A Complete Narrowing Calculus
for Higher-Order Functional logic Programming

Koichi Nakahara    Aart Middeldorp    Tetsuo Ida

March, 1995

ISE-TR-95-118

Institute of Information Sciences and Electronics,
University of Tsukuba,
Tsukuba, Ibaraki 305, Japan
E-mail: {koh,ami,ida}@scorc.is.tsukuba.ac.jp
A Complete Narrowing Calculus for Higher-Order Functional Logic Programming

Koichi Nakahara  Aart Middeldorp  Tetsuo Ida
Institute of Information Sciences and Electronics
University of Tsukuba, Tsukuba 305, Japan
{koh,ami,ida}@score.is.tsukuba.ac.jp

ABSTRACT
Using higher-order functions is standard practice in functional programming, but most functional logic programming languages that have been described in the literature lack this feature. The natural way to deal with higher-order functions in the framework of (first-order) term rewriting is through so-called applicative term rewriting systems. In this paper we argue that existing calculi for lazy narrowing either do not apply to applicative systems or handle applicative terms very inefficiently. We propose a new lazy narrowing calculus for applicative term rewriting systems and prove its completeness.

1. Introduction
There is a growing interest in combining the functional and logic programming paradigms in a single language, see Hanus [6] for a recent overview of the field. The underlying computational mechanism of most of these integrated languages is (conditional) narrowing. Examples of such languages include BABEL [9] and K-LEAF [4]. Both BABEL and K-LEAF lack higher-order features. Bosco and Giovannetti [2] extended K-LEAF to the higher-order language IDEAL. The semantics of IDEAL is given by means a translation from IDEAL programs into K-LEAF programs. González-Moreno et al. proposed in [5] the language SPL, a higher-order extension of BABEL. The higher-order aspects of these two languages are modeled by means of first-order applicative (conditional) constructor-based term rewriting systems. This means in particular that higher-order unification—like in the higher-order logic programming language λ-PROLOG [11]—is avoided because there are no λ-abstractions around. The following example program is taken from [5]:

\[
\begin{align*}
\text{plus} \ 0 \ y &= y & \text{map} \ f \ [] &= [] \\
\text{plus} \ (S \ x) \ y &= S \ (\text{plus} \ x \ y) & \text{map} \ f \ [x \ y] &= [f \ x \ | \ \text{map} \ f \ y] \\
\text{double} \ x &= \text{plus} \ x \ x & \text{compose} \ f \ x \ y &= f \ (g \ x)
\end{align*}
\]

The functions \text{map} and \text{compose} are higher-order. Solving the goal

\[
\text{map} \ f \ [S \ 0, \ 0, \ S \ 0] = [S \ (S \ (S \ 0)), \ S \ 0, \ S \ (S \ (S \ 0))]
\]

means finding a substitution for the higher-order variable \( f \) such that the value of the left-hand side of the goal equals the right-hand side. One easily verifies that

\[
f \mapsto \ \text{compose} \ S \ \text{double}
\]
is a solution to the goal, but actually computing such solutions is a different matter. The operational semantics of SFL is a particular kind of conditional narrowing and González-Moreno et al. [5] prove its soundness and completeness with respect to a declarative semantics that is based on applicative algebras over Scott domains.

In this paper we are concerned with lazy narrowing strategies for applicative term rewriting systems. Most lazy narrowing strategies that have been proposed in the literature are defined for constructor-based term rewriting systems, e.g. [1, 10, 13]. An easy but important observation is that while every applicative term rewriting system is a particular kind of term rewriting system, not every applicative constructor-based term rewriting system is a constructor-based term rewriting system. Nevertheless, an applicative orthogonal (constructor-based) term rewriting system is an orthogonal term rewriting system, so lazy narrowing strategies that are defined and proved complete for the latter class can be used as a computation model for higher-order functional logic programming.

We analyze the behaviour of OINC—a simple calculus proposed in [7] which realizes lazy narrowing—for applicative orthogonal term rewriting systems. It turns out that OINC handles applicative terms very inefficiently. We transform OINC into a calculus NCA that deals with applicative terms in an efficient way and we prove the completeness of NCA. We would like to stress that the ideas developed in this paper do not depend on OINC. The only reason for choosing OINC is the simplicity of its inference rules.

This paper is organized as follows. In the next section we introduce applicative term rewriting. In Section 3 we recall the calculus OINC. In Section 4 we observe that OINC doesn't manipulate applicative term rewriting systems in a very efficient way. The new calculus NCA is defined to overcome this inefficiency. The completeness of NCA is proved in Section 5. Section 6 is concerned with a further optimization of our calculus, namely we extend NCA with special inference rules for dealing with strict equality in an efficient way. In Section 7 we compare the relative efficiency of NCA and OINC on a small example. We conclude in Section 8 with suggestions for future research.

2. Preliminaries

We assume the reader is familiar with the basics of term rewriting. (See [3] and [8] for extensive surveys.) In this preliminary section we recall only some less common definitions and we introduce the notion of applicative term rewriting.

The set of function symbols $\mathcal{F}$ of a term rewriting system (TRS for short) $(\mathcal{F}, \mathcal{R})$ is partitioned into disjoint sets $\mathcal{F}_D$ and $\mathcal{F}_C$ as follows: a function symbol $f$ belongs to $\mathcal{F}_D$ if there is a rewrite rule $l \rightarrow r$ in $\mathcal{R}$ such that $l = f(t_1, \ldots, t_n)$ for some terms $t_1, \ldots, t_n$, otherwise $f \in \mathcal{F}_C$. Function symbols in $\mathcal{F}_C$ are called constructors, those in $\mathcal{F}_D$ defined symbols. A term built from constructors and variables is called a constructor term. A constructor system (CS for short) is a TRS with the property that the arguments $t_1, \ldots, t_n$ of every left-hand side $f(t_1, \ldots, t_n)$ of a rewrite rule are constructor terms. A left-linear TRS without critical pairs is called orthogonal.

We distinguish a binary function symbol $\approx$, written in infix notation. A term of the form $s \approx t$, where neither $s$ nor $t$ contains any occurrences of $\approx$, is called an equation. Observe that we do not identify the equations $s \approx t$ and $t \approx s$. A goal is a sequence of equations. The empty goal is the empty sequence and denoted by $\Box$.

In applicative term rewriting we deal with applicative terms. Such terms are built from
variables, constants, and a special binary function symbol application, which is denoted by juxtaposition of its two arguments. Examples of applicative terms are \((+ (S 0)) 0\) and \(S (x 0)\). To distinguish constants from variables in applicative terms we denote the former always in typewriter style. Parentheses are omitted under the convention of association to the left, which means that missing parentheses are restored by always taking the leftmost possibility, so \((+ (S 0)) 0\) and \(+ (S 0) 0\) denote the same term, which is different from \(+ S 0\). The head-symbol of an applicative term is the symbol that occurs at the leftmost-innermost position. This symbol is either a constant or a variable. For instance, the head-symbol of \(+ (S 0) 0\) is + and the head-symbol of \(x 0\) is \(x\). We assume that every constant \(f\) is equipped with a natural number \(\text{arity}(f)\). Intuitively this number indicates the number of arguments we have to supply in order to evaluate the function or build the data structure. In the following definition we identify a subclass of applicative terms that is used to define applicative term rewriting systems.

**Definition 2.1.** A pattern is an applicative term \(t\) with the property that the head-symbol of every non-variable subterm of \(t\) is a constant.

So a pattern is either a variable or a term of the form \(f t_1 \cdots t_n\) where \(t_1, \ldots, t_n\) are patterns. The term \(+ (S 0) (+ x)\) is a pattern, but \(S (x 0)\) isn’t. Now we are ready to define applicative term rewriting systems.

**Definition 2.2.** An applicative rewrite rule is a pair \(l \rightarrow r\) of applicative terms such that the left-hand side \(l\) is a pattern of the form \(f l_1 \cdots l_n\) with \(n = \text{arity}(f)\), and \(\text{Var}(r) \subseteq \text{Var}(l)\). An applicative term rewriting system (ATRS for short) consists of applicative rewrite rules.

Every ATRS is a TRS. Hence notions defined for TRSs like orthogonality apply to ATRSs. We would like to point out however that the definition of CS doesn’t make much sense in the context of ATRSs. The well-known map function from functional programming can be specified as the following ATRS:

\[
\begin{align*}
\text{map} f \text{nil} & \rightarrow \text{nil} \\
\text{map} f (:+ y) & \rightarrow :(f x)(\text{map} f y)
\end{align*}
\]

We have \(\text{arity} (\text{map}) = 2\). This ATRS is not a CS because the arguments of the two left-hand sides contain (hidden) application symbols, which are in \(F_p\). For example, the arguments of the left-hand side \(\text{map} f \text{nil}\) are \(\text{map} f\) and \text{nil}, not \(f\) and \text{nil}. Nevertheless, there is a clear separation between constants that define functions (map) and those that build data structures (nil and :). This suggests the following definition.

**Definition 2.3.** Let \(\mathcal{R}\) be an ATRS. A constant \(f\) is said to be applicatively defined if it is the head-symbol of the left-hand side of some rewrite rule in \(\mathcal{R}\). An applicative constructor is a constant that is not defined. We call \(\mathcal{R}\) an applicative constructor system (ACS for short) if the terms \(t_1, \ldots, t_n\) in the left-hand side \(f t_1 \cdots t_n\) of every rewrite rule do not contain defined symbols.

The ATRS defining the map function is an ACS. We would like to stress that ACSs are not CSs, except in trivial cases. So narrowing strategies that are defined for CSs do not apply to ACSs.

When writing applicative terms we find it convenient to abbreviate \(f t_1 \cdots t_n\) to \(f t_n\). Observe that \(t_n\) is not a term. If \(n = 0\) then \(f t_n\) denotes the constant \(f\). By the same convention \(x s_n t_m\) stands for the term \(x s_1 \cdots s_n t_1 \cdots t_m\). A term of the form \(x t\) is called a head-variable term. In the sequel, when dealing with ATRSs, we usually omit the adjective applicative.
3. The Outside-In Narrowing Calculus

In this section we recall the outside-in narrowing calculus of [7] and state its completeness.

**Definition 3.1.** Let $\mathcal{R}$ be an orthogonal TRS. The outside-in narrowing calculus, OINC for short, consists of the following inference rules ($E$ denotes an arbitrary sequence of equations):

- **outermost narrowing [on]**
  \[
  \frac{f(s_1, \ldots, s_n) \approx t, E}{s_1 \approx l_1, \ldots, s_n \approx l_n, r \approx t, E}
  \]
  if $t \not\in \mathcal{V}$ and there exists a fresh variant $f(l_1, \ldots, l_n) \rightarrow r$ of a rewrite rule in $\mathcal{R}$,

- **decomposition [d]**
  \[
  \frac{f(s_1, \ldots, s_n) \approx f(t_1, \ldots, t_n), E}{s_1 \approx t_1, \ldots, s_n \approx t_n, E}
  \]

- **variable elimination [v]**
  \[
  \frac{t \approx x, E}{E\theta}
  \]
  and
  \[
  \frac{x \approx t, E}{E\theta}
  \]
  with $\theta = \{x \mapsto t\}$. In the second variable elimination rule we require that $t \not\in \mathcal{V}$.

We introduce some useful notations relating to the calculus OINC. If $G$ and $G'$ are the upper and lower goal in the inference rule $\alpha$ ($\alpha \in \{\text{on}, \text{d}, \text{v}\}$), we write $G \Rightarrow_{\alpha} G'$. This is called an OINC-step. The applied rewrite rule or substitution may be supplied as a subscript, that is, we will write things like $G \Rightarrow_{\text{on}, l \mapsto y} G'$ and $G \Rightarrow_{\alpha, \theta} G'$. A finite OINC-derivation $G_1 \Rightarrow_{\theta_1} \cdots \Rightarrow_{\theta_{n-1}} G_n$ may be abbreviated to $G_1 \Rightarrow_{\theta}^* G_n$ with $\theta = \theta_1 \cdots \theta_{n-1}$. A successful OINC-derivation ends in the empty goal $\Box$. The number of steps in an OINC-derivation $A: G \Rightarrow^* G'$ is denoted by $|A|$. If $|A| \geq 1$ then $A_{>1}$ denotes the derivation obtained from $A$ by omitting the first step.

The calculus OINC has been designed with the restriction in mind that initial goals are right-normal.

**Definition 3.2.** A goal $G$ is called right-normal if the right-hand side $t$ of every equation $s \approx t$ in $G$ is a ground normal form. A goal $G'$ is called proper if there exists an OINC-derivation $G \Rightarrow^* G'$ starting from a right-normal goal $G$. 

4
The restriction to proper goals is motivated from the understanding that functional logic programming languages deal with so-called strict equality in order to model non-termination correctly. Every goal consisting of strict equations is right-normal, see Section 6.

It is not difficult to show that the term \( t \) in a proper goal \( E_1, s \approx t, E_2 \) has no variables in common with \( s \) and \( E_1 \). This explains why we don’t need the occur-check in the variable elimination rules. It also explains why there is no imitation rule in \( \mathit{OINC} \). Finally, the absence of the symmetric outermost narrowing rule

\[
\frac{t \approx f(s_1, \ldots, s_n), E \quad s_1 \approx l_1, \ldots, s_n \approx l_n, t \approx r, E}{s_1 \approx l_1, \ldots, s_n \approx l_n, t \approx r, E}
\]

is easily explained by the restriction to orthogonal TRSs and proper goals.

**Definition 3.3.** Let \( \mathcal{R} \) be a TRS and \( G \) a goal. A substitution \( \theta \) is called a solution of \( G \) if \( s\theta = t\theta \) for every equation \( s \approx t \) in \( G \).

Ida and Nakahara [7] obtained the following soundness and completeness result.

**Theorem 3.4.** Let \( \mathcal{R} \) be an orthogonal TRS and \( G \) a right-normal goal.

1. If there exists a successful \( \mathit{OINC} \)-derivation \( G \Rightarrow \gamma \Box \) then \( \theta \) is a solution of \( G \).
2. For every normalizable solution \( \theta \) of \( G \) there exists a successful \( \mathit{OINC} \)-derivation \( G \Rightarrow \gamma', \Box \) such that \( \theta' \leq \mathcal{R} \theta [\mathrm{Var}(G)] \).

\( \Box \)

If the substitution \( \theta \) in part (2) is normalized, the subscript \( \mathcal{R} \) can be dropped.

**Example 3.5.** Consider the orthogonal TRS

\[
\mathcal{R} = \left\{ \begin{array}{l}
0 + y \rightarrow y \\
S(x) + y \rightarrow S(x + y)
\end{array} \right.
\]

and the right-normal goal \( x + y \approx S(0) \). The solutions \( \{x \mapsto 0, y \mapsto S(0)\} \) and \( \{x \mapsto S(0), y \mapsto 0\} \) are generated by the \( \mathit{OINC} \)-derivations

\[
x + y \approx S(0) \Rightarrow [\text{on}], 0 + y_1 \rightarrow y_1 \quad x \approx 0, y \approx y_1, y_1 \approx S(0)
\]

\[
\Rightarrow [\text{on}], \{x \mapsto 0\} \quad y \approx y_1, y_1 \approx S(0)
\]

\[
\Rightarrow [\text{on}], \{y_1 \mapsto y\} \quad y \approx S(0)
\]

\[
\Rightarrow [\text{on}], \{y \mapsto S(0)\} \quad \Box
\]

and

\[
x + y \approx S(0) \Rightarrow [\text{on}], S(x_1) + y_1 \rightarrow S(x_1 + y_1) \quad x \approx S(x_1), y \approx y_1, S(x_1 + y_1) \approx S(0)
\]

\[
\Rightarrow [\text{on}], \{x \mapsto S(x_1)\} \quad y \approx y_1, S(x_1 + y_1) \approx S(0)
\]

\[
\Rightarrow [\text{on}], \{y_1 \mapsto y\} \quad S(x_1 + y) \approx S(0)
\]

\[
\Rightarrow [\text{on}], \{y \mapsto S(0)\} \quad x_1 + y \approx 0
\]

\[
\Rightarrow [\text{on}], 0 + y_2 \rightarrow y_2 \quad x_1 \approx 0, y \approx y_2, 0 \approx y_2
\]

\[
\Rightarrow [\text{on}], \{x_1 \mapsto 0\} \quad y \approx y_2, 0 \approx y_2
\]

\[
\Rightarrow [\text{on}], \{y_2 \mapsto y\} \quad 0 \approx y
\]

\[
\Rightarrow [\text{on}], \{y \mapsto 0\} \quad \Box
\]

Note that in each outermost narrowing step a fresh variant of a rewrite rule is used.
4. A Narrowing Calculus for Applicative Systems

Every ATRS is a TRS, hence \( \text{oinc} \) is complete (in the sense of Theorem 3.4(2)) for orthogonal ATRSs. However, \( \text{oinc} \) doesn’t handle applicative terms very efficiently. The problem is that the applicable inference rules and (in the case of \([on]\)) rewrite rules are determined by the outermost symbol of the left-hand side of the leftmost equation of the current goal. In the context of ATRSs this outermost symbol is almost always the binary application symbol, which doesn’t carry any useful information for restricting the choice of inference and rewrite rules. Let us consider two examples.

Example 4.1. The (right-normal) goal \( cxy \approx cab \) is solved by the following \( \text{oinc} \)-derivation:

\[
\begin{align*}
  cxy \approx cab & \Rightarrow [d] \quad c \approx ca, y \approx b \\
                 & \Rightarrow [d] \quad c \approx c, x \approx a, y \approx b \\
                 & \Rightarrow [d] \quad x \approx a, y \approx b \\
                 & \Rightarrow [v], \{x \mapsto a\} \quad y \approx b \\
                 & \Rightarrow [v], \{y \mapsto b\} \; \Box.
\end{align*}
\]

We need three decomposition steps before we can bind the variables.

Example 4.2. Consider the orthogonal ATRS

\[
\mathcal{R} = \begin{cases}
  \text{inc} & \rightarrow \text{add1} \\
  \text{add1}x & \rightarrow Sx
\end{cases}
\]

The following \( \text{oinc} \)-derivation computes the solution \( \{x \mapsto 0\} \) of the goal \( \text{inc} \; x \approx S0 \):

\[
\begin{align*}
  \text{inc} \; x \approx S0 & \Rightarrow [on], \text{add1} \; x \overset{\rightarrow}{\rightarrow} S \; x1 \quad \text{inc} \approx \text{add1}, x \approx x1, S \; x1 \approx S0 \\
                         & \Rightarrow [on], \text{inc} \rightarrow \text{add1} \quad \text{add1} \approx \text{add1}, x \approx x1, S \; x1 \approx S0 \\
                         & \Rightarrow [d] \quad x \approx x1, S \; x1 \approx S0 \\
                         & \Rightarrow [v], \{x1 \mapsto x\} \quad S \; x \approx S0 \\
                         & \Rightarrow [d] \quad S \approx S, x \approx 0 \\
                         & \Rightarrow [d] \quad x \approx 0 \\
                         & \Rightarrow [v], \{x \mapsto 0\} \; \Box.
\end{align*}
\]

It is essential that we choose the (renamed) rewrite rule \( \text{add1} \; x1 \rightarrow S \; x1 \) for the equation \( \text{inc} \; x \approx S0 \) in the first outermost narrowing step. However, we have no way to implement this choice. In order to ensure completeness all rewrite rules whose left-hand side is not a single constant must be used in combination with the outermost narrowing rule.

We overcome the problems mentioned above by looking at the head-symbol rather than the outermost symbol of the left-hand side of the equation under consideration. This is natural since the head-symbol of an applicative term corresponds to the outermost symbol of a functional term. The narrowing calculus defined below implements this idea.

Definition 4.3. Let \( \mathcal{R} \) be an orthogonal ATRS. The calculus \( nca \)—Narrowing Calculus for Applicative TRSs—consists of the following five inference rules:

\[
\]
outermost narrowing of applicative terms [ona]
\[
\frac{f \, s_n \, t_m \approx t, E}{s_1 \approx u_1, \ldots, s_n \approx u_n, r \, t_m \approx t, E}
\]
if \( t \notin V \) and there exists a fresh variant \( f \, u_n \to r \) of a rewrite rule in \( \mathcal{R} \),

outermost narrowing of head-variable terms [onv]
\[
\frac{x \, s_n \, t_m \approx t, E}{(s_1 \approx v_1, \ldots, s_n \approx v_n, r \, t_m \approx t, E)\theta}
\]
if \( t \notin V \), there exists a fresh variant \( f \, u_k \, v_n \to r \) of a rewrite rule in \( \mathcal{R} \), \( n > 0 \), and \( \theta = \{x \mapsto f \, u_k\} \),

decomposition of applicative terms [da]
\[
\frac{f \, s_n \approx f \, t_n, E}{s_1 \approx t_1, \ldots, s_n \approx t_n, E}
\]

decomposition of head-variable terms [dv]
\[
\frac{x \, s_n \approx f \, t_m \, u_n, E}{(s_1 \approx u_1, \ldots, s_n \approx u_n, E)\theta}
\]
if \( \theta = \{x \mapsto f \, t_m\} \),

variable elimination [v]
\[
\frac{t \approx x, E}{E\theta}
\]
if \( \theta = \{x \mapsto t\} \).

Observe that the second variable elimination rule of ONC is subsumed by the inference rule [dv] of NCA. In order to distinguish NCA-derivations from ONC-derivations, we use \( \Rightarrow \) instead of \( \Rightarrow \) for the former.

**Example 4.4.** The goal \( c \, x \, y \approx c \, a \, b \) is solved by the following NCA-derivation:

\[
\begin{align*}
    c \, x \, y \approx c \, a \, b & \Rightarrow [da] \quad x \approx a, y \approx b \\
    & \Rightarrow [dv], \{x \mapsto a\} \quad y \approx b \\
    & \Rightarrow [dv], \{y \mapsto b\} \quad \Box.
\end{align*}
\]

With respect to the ATRS \( \mathcal{R} \) of Example 4.2, the goal \( \text{inc} \, x \approx S \, 0 \) is solved by the following NCA-derivation:

\[
\begin{align*}
    \text{inc} \, x \approx S \, 0 & \Rightarrow [ona], \text{inc} \to \text{add}1 \quad \text{add}1 \, x \approx S \, 0 \\
    & \Rightarrow [ona], \text{add}1 \, x_1 \to S \, x_1 \quad x \approx x_1, S \, x_1 \approx S \, 0 \\
    & \Rightarrow [v], \{x_1 \mapsto x\} \quad S \, x \approx S \, 0 \\
    & \Rightarrow [da] \quad x \approx 0 \\
    & \Rightarrow [dv], \{x \mapsto 0\} \quad \Box.
\end{align*}
\]
The inference rule $[onv]$ of nca is used to bind higher-order logical variables. This is illustrated in the next example.

**Example 4.5.** Consider the orthogonal $\mathcal{ATRS}$

$$\mathcal{R} = \begin{cases} 
\text{plus } 0 \, x & \rightarrow \, x \\
\text{plus } (S \, x) \, y & \rightarrow \, S (\text{plus } x \, y) \\
\text{map } f \, \text{nil} & \rightarrow \, \text{nil} \\
\text{map } f \, (x : y) & \rightarrow \, (f \, x) \, (\text{map } f \, y) 
\end{cases}$$

Here $:$ is a binary constructor, written in infix notation, and nil is a constant denoting the empty list. We adopt the usual convention of writing $[t_1, \ldots, t_n]$ to denote the list $(t_1 : (\cdots (t_n : \text{nil}) \cdots))$.

The goal map $x \, [S \, O] \approx [S \, O]$ is solved by the following nca-derivation:

$$\text{map } x \, [S \, O] \approx [S \, O] \quad \Rightarrow_{\text{onv}} \quad x \approx f_1, [S \, O] \approx (x_1 : y_1), (f_1 \, x_1) : (\text{map } f_1 \, y_1) \approx [S \, O]$$

$$\Rightarrow_{[\text{t}]}, \{f_1 \rightarrow x\} \quad \Rightarrow_{[\text{da}]}, \{x_1 \rightarrow \text{nil}\} \quad \Rightarrow_{[\text{v}]} \quad S_0 \approx x_1, \text{nil} \approx y_1, (x_1 \, x_1) : (\text{map } x_1 \, y_1) \approx [S \, O]$$

$$\Rightarrow_{[\text{da}]} \quad \text{nil} \approx y_1, (x (S_0)) : (\text{map } x \, y_1) \approx [S \, O]$$

$$\Rightarrow_{[\text{v}]} \quad (x (S_0)) : (\text{map } x \, \text{nil}) \approx [S \, O]$$

$$\Rightarrow_{[\text{da}]} \quad x (S_0) \approx S_0, \text{map } x \, \text{nil} \approx \text{nil}$$

$$\Rightarrow_{[\text{onv}], (1)}, \{(x \rightarrow \text{plus } 0)\} \quad S_0 \approx y_2, y_2 \approx S_0, \text{map } (\text{plus } 0) \, \text{nil} \approx \text{nil}$$

$$\Rightarrow_{[\text{v}]} \quad S_0 \approx S_0, \text{map } (\text{plus } 0) \, \text{nil} \approx \text{nil}$$

$$\Rightarrow_{[\text{da}]} \quad \text{map } (\text{plus } 0) \, \text{nil} \approx \text{nil}$$

$$\Rightarrow_{[\text{onv}], (3)} \quad \text{plus } 0 \approx f_2, \text{nil} \approx \text{nil}, \text{nil} \approx \text{nil}$$

$$\Rightarrow_{[\text{v}]} \quad \text{plus } 0 \approx f_2, \text{nil} \approx \text{nil}, \text{nil} \approx \text{nil}$$

$$\Rightarrow_{[\text{da}]} \quad \square$$

Note that in the $\Rightarrow_{\text{onv}}$ step the variable $x$ is bound to the higher-order term $(\text{plus } 0)$.

Soundness of nca is expressed in the following theorem.

**Theorem 4.6.** Let $\mathcal{R}$ be an orthogonal TRS and $G$ a right-normal goal. If there exists a successful nca-derivation $G \Rightarrow^* \square$ then $\theta$ is a solution of $G$.

**Proof.** Straightforward induction on the length of the successful nca-derivation $G \Rightarrow^* \square$.

5. **Completeness**

In this section we establish the completeness of nca. The idea behind the proof is straightforward. We show that for every non-empty oinc-derivation $A : G \Rightarrow^* \square$ there exist an nca-step $G \Rightarrow^* G'$ and an oinc-derivation $A' : G' \Rightarrow^* \square$ such that $\theta = \sigma \theta'$ and $|A'| < |A|$. Completeness of nca is then reduced to the completeness of oinc by a routine induction argument. First we define an appropriate notion of descendant for oinc-derivations.

**Definition 5.1.** Let $\mathcal{R}$ be an ATRS. In the oinc-derivation

$$E \Rightarrow^* s_1 \, s_2 \approx t_1 \, t_2, E_1 \Rightarrow[\theta] \approx s_1 \approx t_1, s_2 \approx t_2, E_1 \Rightarrow^* E_2$$
the equation $s_1 \approx t_1$ is called an \textit{immediate descendant} of the equation $s_1 s_2 \approx t_1 t_2$. In the oINC-derivation

$$E \Rightarrow^* s_1 s_2 \approx t, E_1 \Rightarrow_{[on]} s_1 \approx l_1, s_2 \approx l_2, r \approx t, E_1 \Rightarrow^* E_2$$

where $l_1 l_2 \rightarrow r$ is a fresh variant of a rewrite rule in $\mathcal{R}$, the equation $s_1 \approx l_1$ is called an \textit{immediate descendant} of the equation $s_1 s_2 \approx t$. The notion of immediate descendants generalizes to a notion of descendant by reflexivity and transitivity.

In Lemmata 5.2–5.7 we observe basic properties of successful oINC-derivations.

**Lemma 5.2.** Let $G = f s_n \approx g t_m, E$ be a goal such that $f \neq g$ or $n \neq m$. In all successful oINC-derivations starting from $G$ the rule $[on]$ is applied to a descendant of $f s_n \approx g t_m$.

**Proof.** Obvious. □

**Lemma 5.3.** Let $G = x s_n \approx g t_m, E$ be a goal such that $m < n$. In all successful oINC-derivations starting from $G$ the rule $[on]$ is applied to a descendant of $x s_n \approx g t_m$.

**Proof.** Obvious. □

**Lemma 5.4.** Let $G = f s_n \approx t, E$ be a goal such that $n < \text{arity}(f)$. There exist no successful oINC-derivations starting from $G$ in which the rule $[on]$ is applied to a descendant of $f s_n \approx t$.

**Proof.** We use induction on $n$. If $n = 0$ then there are no rewrite rules of the form $f \rightarrow r$ because $\text{arity}(f) > 0$ and hence $[on]$ is not applicable. Suppose $n > 0$. Let $A$ be an arbitrary successful oINC-derivation starting from $G$. We distinguish the following three cases:

1. Suppose the inference rule $[d]$ is applied in the first step of $A$, so $A$ can be written as

$$f s_n \approx t, E \Rightarrow [d] f s_{n-1} \approx t_1, s_n \approx t_2, E \Rightarrow^* □$$

with $t = t_1 t_2$. According to the induction hypothesis the inference rule $[on]$ is not applied to a descendant of $f s_{n-1} \approx t_1$ in the subderivation $A_{>1}$. Therefore $[on]$ is not applied to a descendant of $f s_n \approx t$ in $A$.

2. Suppose the inference rule $[on]$ is applied in the first step of $A$, so $A$ can be written as

$$f s_n \approx t, E \Rightarrow [on], l_1 l_2 \rightarrow r f s_{n-1} \approx l_1, s_n \approx l_2, r \approx t, E \Rightarrow^* □$$

for some rewrite rule $l_1 l_2 \rightarrow r$. We have to show that this is impossible. According to the induction hypothesis the inference rule $[on]$ is not applied to a descendant of $f s_{n-1} \approx l_1$ in the subderivation $A_{>1}$. Because $l_1 l_2$ is a pattern, $l_1$ is not a head-variable term, so we may write $l_1 = g u_m$ with $m = \text{arity}(g) - 1$. We have either $f \neq g$ or $n - 1 \neq m$. Now Lemma 5.2 yields the desired contradiction.

3. If the inference rule $[v]$ is applied in the first step of $A$ then by definition there are no descendants of $f s_n \approx t$ left in $A_{>1}$ to which $[on]$ can be applied. Therefore $[on]$ is not applied to a descendant of $f s_n \approx t$ in $A$.

□

**Lemma 5.5.** Let $G = f s_n \approx f t_n, E$ be a goal such that $n < \text{arity}(f)$. In every successful oINC-derivation starting from $G$ the rule $[d]$ is applied to all descendants of $f s_n \approx f t_n$.

**Proof.** Easy consequence of Lemma 5.4. □
Lemma 5.6. Let $G = f s_n \approx t, E$ be a goal such that $n = \text{arity}(f)$. If there exists a successful oinc-derivation starting from $G$ in which the first step is an application of the rule $[on]$, then the rewrite rule used in this step is of the form $f u_n \rightarrow r$.

Proof. Easy consequence of Lemmata 5.2 and 5.4. □

Lemma 5.7. Let $G = f s_n t_m \approx t, E$ be a goal such that $n = \text{arity}(f)$ and $m > 0$. If there exists a successful oinc-derivation $A$ starting from $G$ in which the rule $[on]$ is applied to a descendant $f s_n t_k \approx t'$ ($1 \leq k \leq m$) then $[on]$ is applied to the descendant $f s_n \approx t''$ of $f s_n t_m \approx t$.

Proof. Without loss of generality we may write $A$ as

\[
G \Rightarrow \quad f s_n t_k \approx t', E' \\
\Rightarrow_{[on], g u_{\ell} \rightarrow r} f s_n t_{k-1} \approx g u_{\ell-1}, t_k \approx u_{\ell}, r \approx t', E' \\
\Rightarrow_{[d]} f s_n \approx g u_{\ell-k}, E'' \\
\Rightarrow_{\theta} 
\]

where $g u_{\ell} \rightarrow r$ is a fresh variant of a rewrite rule in $\mathcal{R}$. Note that $\ell = n$ if $f = g$. So we have $f \neq g$ or $n \neq \ell - k$. According to Lemmata 5.2 and 5.4 the inference rule $[on]$ is applied to the equation $f s_n \approx g u_{\ell-k}$. □

In Lemmata 5.8–5.12 we prove that for certain successful oinc-derivation $A: G \Rightarrow_{\theta} \square$ there exists an oinc-derivation $A': G' \Rightarrow_{\theta'} \square$ such that $G \Rightarrow_{\theta} G'$, $\theta = \sigma \theta'$, and $|A'| < |A|$. In Lemma 5.13 we show that there are no other cases to consider.

Lemma 5.8. Let $A: s \approx x, E \Rightarrow_{\theta} \square$ be an oinc-derivation. There exists an oinc-derivation $A': E \sigma \Rightarrow_{\theta'} \square$

with $\sigma = \{x \mapsto s\}$ such that $\theta = \sigma \theta'$ and $|A'| < |A|$.

Proof. The first step of $A$ must be an application of the inference rule $[v]$:

\[A: s \approx x, E \Rightarrow_{[v], \sigma} E \sigma \Rightarrow_{\theta} \square\]

with $\theta = \sigma \theta'$ and $\sigma = \{x \mapsto s\}$. Define $A' = A_{>1}$. We clearly have $|A'| = |A| - 1 < |A|$. □

The initial goals of $A$ and $A'$ in Lemma 5.8 are connected by a $\Rightarrow_{[v], \sigma}$-step.

Lemma 5.9. Let $A: f s_n \approx f t_n, E \Rightarrow_{\theta} \square$ be an oinc-derivation. If $[on]$ is never applied to a descendant of $f s_n \approx f t_n$ then there exists an oinc-derivation $A': s_1 \approx t_1, \ldots, s_n \approx t_n, E \Rightarrow_{\theta} \square$

such that $|A'| < |A|$.

Proof. By induction on $n$. If $n = 0$ then we take $A' = A_{>1}$. In this case we clearly have $|A'| < |A|$. Suppose $n > 0$. The first step of $A$ must be an application of $[d]$, so we may write $A$ as

\[f s_n \approx f t_n, E \Rightarrow_{[d]} f s_{n-1} \approx f t_{n-1}, s_n \approx t_n, E \Rightarrow_{\theta} \square\]

An application of the induction hypothesis to the oinc-derivation $A_{>1}$ yields an oinc-derivation $A': s_1 \approx t_1, \ldots, s_{n-1} \approx t_{n-1}, s_n \approx t_n, E \Rightarrow_{\theta} \square$

with $|A'| < |A_{>1}| = |A| - 1 < |A|$. □
Note that the initial goals of $A$ and $A'$ in Lemma 5.10 are connected by a $\Rightarrow_{\{da\}}$-step.

**Lemma 5.10.** Let $A: x s_n \approx f t_m u_n \Rightarrow \phi \sqcap$ be an \textsc{oinc}-derivation. If $[on]$ is never applied to a descendant of $x s_n \approx f t_m u_n$ then there exists an \textsc{oinc}-derivation

$$A': (s_1 \approx u_1, \ldots, s_n \approx u_n, E) \sigma \Rightarrow \phi' \sqcap$$

with $\sigma = \{x \mapsto f t_m\}$ such that $\theta = \sigma \theta'$ and $|A'| < |A|$.

**Proof.** Similar to the proof of Lemma 5.9. $\square$

The initial goals of $A$ and $A'$ in Lemma 5.10 are connected by a $\Rightarrow_{\{da\}, \sigma}$-step.

**Lemma 5.11.** Let $A: f s_n t_m \approx t \Rightarrow \phi$ be an \textsc{oinc}-derivation with $n = \text{arity}(f)$. If $[on]$ is applied to the descendant $f s_n \approx t'$ of $f s_n t_m \approx t$ using the rewrite rule $f u_n \rightarrow r$ then there exists an \textsc{oinc}-derivation

$$A': s_1 \approx u_1, \ldots, s_n \approx u_n, r t_m \approx t, E \Rightarrow \phi \sqcap$$

such that $|A'| < |A|$.

**Proof.** We use induction on $m$. If $m = 0$ then the inference rule $[on]$ with rewrite rule $f u_n \rightarrow r$ is used in the first step of $A$. If $n = 0$ then we take $A' = A_{>1}$. If $n > 0$ then we may write $A$ as

$$f s_n \approx t, E \Rightarrow [on] f s_{n-1} \approx f u_{n-1}, s_n \approx u_n, r \approx t, E \Rightarrow \phi \sqcap.$$ 

According to Lemma 5.5 the inference rule $[on]$ is not applied to descendants of $f s_{n-1} \approx f u_{n-1}$ in the subderivation $A_{>1}$. Hence we can apply Lemma 5.9 to $A_{>1}$. This yields an \textsc{oinc}-derivation

$$A': s_1 \approx u_1, \ldots, s_n-1 \approx u_{n-1}, s_n \approx u_n, r \approx t, E \Rightarrow \phi \sqcap$$

such that $|A'| < |A_{>1}| = |A| - 1 < |A|$. For the induction step, suppose $m > 0$. Let us abbreviate $s_1 \approx u_1, \ldots, s_n \approx u_n$ to $E'$. We distinguish the following cases:

(1) Suppose the inference rule $[d]$ is used in the first step of $A$. This means that we may write $A$ as

$$s \approx t, E \Rightarrow [d] f s_n t_{m-1} \approx v_1, t_m \approx v_2, E \Rightarrow \phi \sqcap$$

with $t = v_1 v_2$. An application of the induction hypothesis to the subderivation $A_{>1}$ yields an \textsc{oinc}-derivation

$$B: E', r t_{m-1} \approx v_1, t_m \approx v_2, E \Rightarrow \phi \sqcap$$

such that $|B| < |A_{>1}| < |A|$. The \textsc{oinc}-derivation $B$ can be split into

$$B_1: E', r t_{m-1} \approx v_1, t_m \approx v_2, E \Rightarrow \phi_1 (r t_{m-1} \approx v_1, t_m \approx v_2, E) \theta_1$$

and

$$B_2: (r t_{m-1} \approx v_1, t_m \approx v_2, E) \theta_1 \Rightarrow \phi_2 \sqcap$$

with $\theta = \theta_1 \theta_2$. The \textsc{oinc}-derivation $B_1$ can easily be transformed into the \textsc{oinc}-derivation

$$C: E', r t_m \approx v_1 v_2, E \Rightarrow \phi_1 (r t_m \approx v_1 v_2, E) \theta_1.$$ 

Because $m > 0$ we can apply the inference rule $[d]$ to the final goal $(r t_m \approx v_1 v_2, E) \theta_1$ of $C$, yielding the \textsc{oinc}-step

$$D: (r t_m \approx v_1 v_2, E) \theta_1 \Rightarrow [d] (r t_{m-1} \approx v_1, t_m \approx v_2, E) \theta_1.$$
Concatenating the three \textit{oinc}-derivations $C$, $D$, and $B_2$ yields the desired \textit{oinc}-derivation $A': E', \text{rt}_m \approx t, E \Rightarrow^*_\theta \Box$.

Note that $|A'| = |C| + |D| + |B_2| = |B| + 1 < |A|$.

(2) Suppose the inference rule \text{[on]} is used in the first step of $A$. This means that there exists a fresh variant $v_1 v_2 \rightarrow r'$ of a rewrite rule in $\mathcal{R}$ such that $A$ can be written as

$$s \approx t, E \Rightarrow_{[\text{on}]} f s_n \text{rt}_{m-1} \approx v_1, t_m \approx v_2, r' \approx t, E \Rightarrow^*_\theta \Box$$

An application of the induction hypothesis to $A_{>1}$ yields an \textit{oinc}-derivation

$$B: E', \text{rt}_{m-1} \approx v_1, t_m \approx v_2, r' \approx t, E \Rightarrow^*_\theta \Box$$

such that $|B| < |A_{>1}| < |A|$. The \textit{oinc}-derivation $B$ can be split into

$$B_1: E', \text{rt}_{m-1} \approx v_1, t_m \approx v_2, r' \approx t, E \Rightarrow_{\theta_1}^* (\text{rt}_{m-1} \approx v_1, t_m \approx v_2, r' \approx t, E)\theta_1$$
and

$$B_2: (\text{rt}_{m-1} \approx v_1, t_m \approx v_2, r' \approx t, E)\theta_1 \Rightarrow_{\theta_2}^* \Box$$

with $\theta = \theta_1 \theta_2$. We transform $B_1$ into the \textit{oinc}-derivation

$$C: E', \text{rt}_m \approx t, E \Rightarrow_{\theta_1}^* (\text{rt}_m \approx t, E)\theta_1.$$  

Next we apply the inference rule \text{[on]} to the final goal $\text{rt}_m \approx t, E)\theta_1$. Using exactly the same variant $v_1 v_2 \rightarrow r'$. This yields the \textit{oinc-step}

$$D: (\text{rt}_m \approx t, E)\theta_1 \Rightarrow_{[\text{on}]} (\text{rt}_{m-1} \approx v_1, t_m \approx v_2, r' \approx t, E)\theta_1.$$  

Also in this case the desired \textit{oinc}-derivation $A'$ is obtained by concatenating $C$, $D$, and $B_2$.

(3) It is not possible that the first step of $A$ is an application of the variable elimination rule \text{[v]} because then there is no descendant left to which \text{[on]} can be applied.

$\Box$

Observe that the initial goals of $A$ and $A'$ in Lemma 5.11 are connected by a $\Rightarrow_{[\text{ons}]}$-step.

**Lemma 5.12.** Let $A: x s_n \text{rt}_m \approx t, E \Rightarrow^*_\theta \Box$ be an \textit{oinc}-derivation with $n > 0$. If \text{[on]} is applied to the descendant $x s_n \approx t'$ of $x s_n \text{rt}_m \approx t$ using the rewrite rule $f u_k v_n \rightarrow r$ and $x s_n \approx t'$ is the last descendant to which \text{[on]} is applied then there exists an \textit{oinc}-derivation

$$A': (s_1 \approx v_1, \ldots, s_n \approx v_n, r \text{rt}_m \approx t, E)\sigma \Rightarrow_{\theta'}^* \Box$$

with $\sigma = \{x \mapsto f u_k\}$ such that $\theta = \sigma \theta'$ and $|A'| < |A|$.

**Proof.** Similar to the proof of Lemma 5.11. $\Box$

In Lemma 5.12 the initial goals of $A$ and $A'$ are connected by a $\Rightarrow_{[\text{ons}]}$-step.

**Lemma 5.13.** Let $G$ be a proper goal. For every \textit{oinc}-derivation $A: G \Rightarrow^*_\theta \Box$ there exist an nca-step $G \Rightarrow_{\sigma} G'$ and an \textit{oinc}-derivation $A': G' \Rightarrow_{\theta'}^* \Box$ such that $\theta = \sigma \theta'$ and $|A'| < |A|$.

**Proof.** We have to show that Lemmata 5.8–5.12 cover all possible cases. Let $G$ be the goal $s \approx t, E$. The case that $t$ is a variable is covered by Lemma 5.8, so we may assume that $t$ is not a variable. Because $G$ is proper the right-hand side $t$ is a pattern. Hence $t$ is not a head-variable term. We distinguish two cases.

(1) Suppose \text{[on]} is not applied to a descendant of $s \approx t$.  

12
(a) If the head-symbol of \( s \) is a constant \( f \) then, according to Lemma 5.2, we must have \( s = f s_n \) and \( t = f t_n \). This case is covered by Lemma 5.9.

(b) If the head-symbol of \( s \) is a variable \( x \) then, according to Lemma 5.3, \( s = x s_n \) and \( t = f t_m \). This case is covered by Lemma 5.10.

(2) Suppose \([on]\) is applied to a descendant of \( s \approx t \).

(a) If the head-symbol of \( s \) is a constant \( f \) then, according to Lemma 5.4, we have \( s = f s_n t_m \) with \( n = \text{arity}(f) \). From Lemma 5.4 we infer that \([on]\) is never applied to descendants of the form \( f s_k \approx t' \) with \( k < \text{arity}(f) \). Hence the application of \([on]\) is to a descendant of the form \( f s_n t_k \approx t'' \) with \( 0 \leq k \leq m \). According to Lemma 5.7 \([on]\) is applied to the descendant \( f s_n \approx t'' \). Lemma 5.6 states that the employed rewrite rule is of the form \( f u_n \rightarrow r \). Hence this case is covered by Lemma 5.11.

(b) The case that the head-symbol of \( s \) is a variable \( x \) is covered by Lemma 5.12, using similar reasoning as in the previous case.

\[ \square \]

The completeness of \( \text{NCA} \) is an easy consequence of the previous lemma and the completeness of \( \text{oINC} \) (Theorem 3.4(2)).

**Theorem 5.14.** Let \( \mathcal{R} \) be an orthogonal \( \text{ATRS} \) and \( G \) a right-normal goal. For every normalizable solution \( \theta \) of \( G \) there exists a successful \( \text{NCA-derivation} \ G \Rightarrow^*_\theta \) such that \( \theta' \leq_{\mathcal{R}} \theta \). [Var](G). 

**Proof.** According to Theorem 3.4(2) there exists a successful \( \text{oINC-derivation} \ A : G \Rightarrow^*_\theta \) such that \( \theta' \leq_{\mathcal{R}} \theta \). By induction on \( |A| \) we show the existence of a successful \( \text{NCA-derivation} \ G \Rightarrow^*_\theta \). The case \( |A| = 0 \) is trivial. Suppose \( |A| > 0 \). According to Lemma 5.13 there exist an \( \text{NCA-step} \ G \Rightarrow^*_\theta G' \) and an \( \text{oINC-derivation} \ A' : G' \Rightarrow^*_\theta \) such that \( \theta' = \sigma \theta'' \) and \( |A'| < |A| \). The induction hypothesis yields an successful \( \text{NCA-derivation} \ G' \Rightarrow^*_\theta \). Combining this derivation with the \( \text{NCA-step} \ G \Rightarrow^*_\theta G' \) yields the desired \( \text{NCA-derivation} \ G \Rightarrow^*_\theta \). \[ \square \]

6. **Incorporating Strict Equality into \( \text{NCA} \)**

In functional logic programming languages like \( \text{K-LEAF} \) \( [4] \) and \( \text{BABEL} \) \( [9] \) two expressions are considered to be equal if and only if they reduce to the same ground constructor normal form. This so-called \( \text{strict equality} \) is adopted to model non-termination correctly. In the framework of applicative term rewriting, strict equality is realized by adding the rewrite rules

\[
\begin{align*}
\{ & c \equiv c \quad \rightarrow \quad \text{true} & \text{if } c \text{ is a nullary constructor}, \\
& c x_n \equiv c y_n \quad \rightarrow \quad x_1 \equiv y_1 \land \cdots \land x_n \equiv y_n & \text{if } c \text{ is an } n\text{-ary constructor with } n > 0, \\
& \text{true} \land x \quad \rightarrow \quad x
\end{align*}
\]

to a given \( \text{ATRS} \mathcal{R} \), resulting in the \( \text{ATRS} \mathcal{R}_s \). Here \( \equiv \) denotes strict equality and \( \land \) is a binary right-associative symbol, written in infix notation, denoting logical conjunction. In our framework a strict equation is an equation of the form \( s \equiv t \approx \text{true} \), which we abbreviate to \( s \approx t \). A goal consisting of strict equations is trivially right-normal.

Since \( \mathcal{R}_s \) inherits orthogonality from \( \mathcal{R} \), we can solve (strict) equations with respect to the calculus \( \text{NCA} \) and the \( \text{ATRS} \mathcal{R}_s \). However, as observed by Ida and Nakahara \( [7] \) in the context of \( \text{oINC} \), it is much more efficient to add special inference rules for the rewrite rules in \( \mathcal{R}_s \backslash \mathcal{R} \). Based on their ideas, we extend \( \text{NCA} \) in the following definition.
DEFINITION 6.1. Let $\mathcal{R}$ be an orthogonal ATRS. The calculus $\text{NCA}_s$ is obtained by adding the following inference rules to $\text{NCA}$:

- **outermost narrowing of applicative terms for strict equations [onas]**

  $$f s_n t_m \simeq_s t, E$$

  $$s_1 \approx u_1, \ldots, s_n \approx u_n, r t_m \simeq_s t, E$$

  if there exists a fresh variant $f u_n \rightarrow r$ of a rewrite rule in $\mathcal{R}$,

- **outermost narrowing of head-variable terms for strict equations [onvs]**

  $$x s_n t_m \simeq_s t, E$$

  $$\left( s_1 \approx v_1, \ldots, s_n \approx v_n, r t_m \simeq_s t, E \right) \theta$$

  if there exists a fresh variant $f u_k v_n \rightarrow r$ of a rewrite rule in $\mathcal{R}$, $n > 0$, and $\theta = \{ x \mapsto f u_k \}$,

- **decomposition of applicative terms for strict equations [das]**

  $$c s_n \approx_s c t_n, E$$

  $$s_1 \approx s t_1, \ldots, s_n \approx s t_n, E$$

  if $c$ is an $n$-ary constructor symbol,

- **imitation for strict equations [imas]**

  $$x s_n \simeq_s c t_m u_n, E$$

  $$\left( x_1 \approx_s t_1, \ldots, x_m \approx_s t_m, s_1 \approx s u_1, \ldots, s_n \approx s u_n, E \right) \theta$$

  if $c$ is an $(m + n)$-ary constructor symbol and $\theta = \{ x \mapsto c x_m \}$ with $x_m$ fresh variables,

- **decomposition of head-variable terms for strict equations [dvs]**

  $$x s_n \simeq_s y t_m u_n, E$$

  $$\left( x_1 \approx_s t_1, \ldots, x_m \approx_s t_m, s_1 \approx s u_1, \ldots, s_n \approx s u_n, E \right) \theta$$

  if either $x = y$, $m = 0$, and $\theta$ is the empty substitution, or $x \neq y$ and $\theta = \{ x \mapsto y x_m \}$ with $x_m$ fresh variables.

Here $s \simeq_s t$ stands for $s \approx_s t$ or $t \approx_s s$.

Observe that the rewrite rules in $\mathcal{R}_s \setminus \mathcal{R}$ for strict equality are no longer needed in $\text{NCA}_s$.

EXAMPLE 6.2. Let $\mathcal{R} = \{ a \rightarrow b \}$. The following $\text{NCA}_s$-derivation, starting from the goal $G = x a a \approx_s y a$, produces the substitution $\sigma = \{ y \mapsto x b \}$:

$$G \Rightarrow_{\text{dvs}, \{ y = x x_1 \}} a \approx_s x_1, a \approx_s a$$

$$\Rightarrow_{\text{onas}, \{ a \rightarrow b \}} b \approx_s x_1, a \approx_s a$$

$$\Rightarrow_{\text{imas}, \{ x_1 = b \}} a \approx_s a$$

$$\Rightarrow_{\text{onas}, \{ a \rightarrow b \}} b \approx_s a$$

$$\Rightarrow_{\text{onas}, \{ a \rightarrow b \}} b \approx_s b$$

$$\Rightarrow_{\text{das}} \square$$

14
Note that $\sigma$ is not a solution of $G$ since the terms $x \ a \ a$ and $x \ b \ a$ do not reduce to the same ground constructor normal form. However, we would like to stress that $\sigma$ represents (all) solutions of $G$ in the sense that $\sigma \theta$ is a solution of $G$ for all $\theta = \{x \mapsto c \ t_n \mid c \text{ is an } (n + 2)\text{-ary constructor and } t_1, \ldots, t_n \text{ are ground constructor terms}\}$. 

Below we state the completeness of $\text{nca}_s$. The proof, which is essentially the same as the completeness proof of the calculus $\text{oinc}$ studied in [7], is omitted.

**Theorem 6.3.** Let $\mathcal{R}$ be an orthogonal ATRS and $G$ a right-normal goal. For every normalizable solution $\theta$ of $G$ there exists a successful $\text{nca}_s$-derivation $G \Rightarrow_{\theta'}$, $\square$ such that $\theta' \leq_{\mathcal{R}} \theta [\text{Var}(G)]$. $\square$

### 7. Experimental Results

In this section we compare the performance of $\text{nca}$ and $\text{oinc}$ on a small example. We have implemented both calculi in Sicstus Prolog 2.1. We solved goals of the form

$$G_n = \text{map} \ f [S^n \ 0, S^{n-1} \ 0, \ldots, 0] \approx [S^{2n+1} \ 0, S^{2n-1} \ 0, \ldots, 0]$$

with respect to the example program in Section 1. Here $S^n \ 0$ denotes the term $\underbrace{S(S(\cdots(S\ 0)\cdots))}_n$.

Since for each $n$ there are infinitely many normalized solutions of $G_n$, we measured the runtime of the two programs to compute the first solution $\{f \mapsto \text{compose} \ S \ \text{double}\}$. Table 1 shows, for several values of $n$, these times in milliseconds as well as the length of the resulting successful derivation. It is apparent that $\text{nca}$ has a much better performance than $\text{oinc}$.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$\text{OINC}$</th>
<th>$\text{NCA}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3800 msec.</td>
<td>120 msec.</td>
</tr>
<tr>
<td></td>
<td>(73 steps)</td>
<td>(42 steps)</td>
</tr>
<tr>
<td>2</td>
<td>5448 msec.</td>
<td>179 msec.</td>
</tr>
<tr>
<td></td>
<td>(136 steps)</td>
<td>(77 steps)</td>
</tr>
<tr>
<td>3</td>
<td>7401 msec.</td>
<td>250 msec.</td>
</tr>
<tr>
<td></td>
<td>(210 steps)</td>
<td>(118 steps)</td>
</tr>
<tr>
<td>4</td>
<td>8799 msec.</td>
<td>305 msec.</td>
</tr>
<tr>
<td></td>
<td>(295 steps)</td>
<td>(165 steps)</td>
</tr>
<tr>
<td>5</td>
<td>10769 msec.</td>
<td>380 msec.</td>
</tr>
<tr>
<td></td>
<td>(391 steps)</td>
<td>(218 steps)</td>
</tr>
</tbody>
</table>

**Table 1.**

### 8. Concluding Remarks

We have presented complete narrowing calculi for applicative term rewriting. Applicative term rewriting is a natural first-order framework for dealing with higher-order functions in the framework of functional (logic) programming with lazy semantics.

Although $\text{nca}$ and $\text{nca}_s$ have been designed to deal efficiently with applicative terms, there remains some room for improvement. In order to ensure completeness of the calculi, the various
inference rules have to be applied don't know non-deterministically. In general more than one inference rule is applicable to a given goal. For instance, both \([on]+\) and \([da]\) apply to certain goals. One way to remove this particular non-determinism is by restricting ourselves to ACSs. The restriction to ACSs also enables us to add failure rules which can be used to prune unsuccessful derivations at an early stage. Another source of inefficiency in our calculi is in the inference rules \([on]+\) and \([on]+\) themselves. In the worst case there are \(arity(f) - 1\) different ways to apply the two inference rules with respect to a given rewrite rule \(f u_k v_n \rightarrow r\). A practical restriction to remove this non-determinism is by adding types to applicative term rewriting systems. In typed systems we can associate a type with every (head-)variable. This implies that we can uniquely determine the number \(k\) for the rewrite rule \(f u_k v_n \rightarrow r\) such that the type of the head-variable \(x\) is (an instance of) the type of the term \(f u_k\).

As a final remark, we emphasize that the basic ideas in this paper do not depend on the calculus \(\text{OINC}\). For example, it is only a matter of diligence to extend \(\text{NCA}\) to a calculus based on the calculus \(\text{LNC}\) studied in [12]. Because the inference rules of \(\text{LNC}\) are more complex than those of \(\text{OINC}\), the former calculus is complete for arbitrary confluent term rewriting systems and arbitrary initial goals.

References


