A Complete Narrowing Calculus for Higher-Order Functional Logic Programming

Koichi Nakahara,[1]* Aart Middeldorp,[2] Tetsuo Ida[2]

[1] Canon Inc.
Shimomaruko, Ohta-ku, Tokyo 146, Japan

[2] Institute of Information Sciences and Electronics
and
Center for Tsukuba Advanced Research Alliance
University of Tsukuba, Tsukuba 305, Japan

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 SFL, 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]:

$$\begin{array}{lll} \mathtt{plus}\,\mathtt{0}\,y & = y & \qquad \mathtt{map}\,f\,\mathtt{[]} & = \mathtt{[]} \\ \mathtt{plus}\,(\mathtt{S}\,x)\,y & = \mathtt{S}\,(\mathtt{plus}\,x\,y) & \qquad \mathtt{map}\,f\,\mathtt{[}x\,\mathtt{|}\,y\mathtt{]} & = \mathtt{[}f\,x\,\mathtt{|}\,\mathtt{map}\,f\,y\mathtt{]} \\ \mathtt{double}\,x & = \mathtt{plus}\,x\,x & \qquad \mathtt{compose}\,f\,g\,x & = f\,(g\,x) \end{array}$$

* Most of the work reported in this paper was carried out while the first author was at the University of Tsukuba, Doctoral Program of Engineering.

The functions `map` and `compose` are higher-order. Solving the goal

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

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

$$f \mapsto \texttt{compose S double}$$

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, 14]. 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 Sect. 3 we recall the calculus OINC. In Sect. 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 Sect. 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 Sect. 7 we compare the relative efficiency of NCA and OINC on a small example. We conclude in Sect. 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 $\mathcal{F}_\mathcal{D}$ and $\mathcal{F}_\mathcal{C}$ as follows: a function symbol f belongs to $\mathcal{F}_\mathcal{D}$ if there is a rewrite rule $l \to r$ in $\mathcal{R}$ such that $l = f(t_1, \ldots, t_n)$ for some terms $t_1, \ldots, t_n$, otherwise $f \in \mathcal{F}_\mathcal{C}$. Function symbols in $\mathcal{F}_\mathcal{C}$ are called *constructors*, those in $\mathcal{F}_\mathcal{D}$ *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 $(+\,(\mathtt{S}\,\mathtt{0}))\,\mathtt{0}$ and $\mathtt{S}\,(x\,\mathtt{0})$. 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 $(+\,(\mathtt{S}\,\mathtt{0}))\,\mathtt{0}$ and $+\,(\mathtt{S}\,\mathtt{0})\,\mathtt{0}$ denote the same term, which is different from $+\,\mathtt{S}\,\mathtt{0}\,\mathtt{0}$. 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 $+\,(\mathtt{S}\,\mathtt{0})\,\mathtt{0}$ is $+$ and the head-symbol of $x\,\mathtt{0}$ is x. We assume that every constant f is equipped with a natural number $\mathit{arity}(f)$. 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 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 f is not a variable and $t_1, \ldots, t_n$ are patterns. The term $+\,(\mathtt{S}\,\mathtt{0})\,(+\,x)$ is a pattern, but $\mathtt{S}\,(x\,\mathtt{0})$ isn't. Now we are ready to define applicative term rewriting systems.

Definition 2. An *applicative rewrite rule* is a pair $l \to r$ of applicative terms such that the left-hand side l is a pattern of the form $f\,l_1 \cdots l_n$ with $n = \mathit{arity}(f)$, and $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$. An *applicative term rewriting system* ($\mathcal{A}$TRS for short) consists of applicative rewrite rules.

Every $\mathcal{A}$TRS is a TRS. Hence notions defined for TRSs like orthogonality apply to $\mathcal{A}$TRSs. We would like to point out however that the definition of CS doesn't make much sense in the context of $\mathcal{A}$TRSs. The well-known `map` function

from functional programming can be specified as the following $\mathcal{A}$TRS:

$$\begin{cases} \mathtt{map}\, f\, \mathtt{nil} & \to \mathtt{nil} \\ \mathtt{map}\, f\, (\mathtt{:}\, x\, y) & \to \mathtt{:}\, (f\, x)\, (\mathtt{map}\, f\, y) \end{cases}$$

We have $\mathit{arity}(\mathtt{map}) = 2$. This $\mathcal{A}$TRS is not a CS because the arguments of the two left-hand sides contain (hidden) application symbols, which are in $\mathcal{F}_{\mathcal{D}}$. For example, the arguments of the left-hand side $\mathtt{map}\, f\, \mathtt{nil}$ are $\mathtt{map}\, f$ and $\mathtt{nil}$, not f and $\mathtt{nil}$. 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 3. Let $\mathcal{R}$ be an $\mathcal{A}$TRS. 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* ($\mathcal{A}$CS 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 $\mathcal{A}$TRS defining the `map` function is an $\mathcal{A}$CS. We would like to stress that $\mathcal{A}$CSs are not CSs, except in trivial cases. So narrowing strategies that are defined for CSs do not apply to $\mathcal{A}$CSs.

When writing applicative terms we find it convenient to abbreviate $f\, t_1 \cdots t_n$ to $f\, \mathbf{t}_n$. Observe that $\mathbf{t}_n$ is not a term. If $n = 0$ then $f\, \mathbf{t}_n$ denotes the constant f. By the same convention $x\, \mathbf{s}_n\, \mathbf{t}_m$ stands for the term $x\, s_1 \cdots s_n\, t_1 \cdots t_m$. A term of the form $x\, \mathbf{t}_n$ is called a *head-variable* term. In the sequel, when dealing with $\mathcal{A}$TRSs, we usually omit the adjective applicative.

In this paper we consider *untyped* systems since typing does affect neither the design nor the soundness and completeness of our narrowing calculus NCA. However, in Sect. 8 we briefly explain why typing is useful to reduce the search space of NCA.

3 The Outside-In Narrowing Calculus

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

Definition 4. 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):

[on] *outermost narrowing*

$$\frac{f(s_1, \ldots, s_n) \approx t, E}{s_1 \approx l_1, \ldots, s_n \approx l_n, r \approx t, E} \quad t \notin \mathcal{V}$$

if there exists a fresh variant $f(l_1, \ldots, l_n) \to r$ of a rewrite rule in $\mathcal{R}$,

[d] *decomposition*

$$\frac{f(s_1,\ldots,s_n) \approx f(t_1,\ldots,t_n), E}{s_1 \approx t_1,\ldots,s_n \approx t_n, E}$$

[v] *variable elimination*

$$\frac{t \approx x, E}{E\theta} \quad \text{and} \quad \frac{x \approx t, E}{E\theta}\ t \notin \mathcal{V}$$

with $\theta = \{x \mapsto t\}$.

We introduce some useful notations relating to the calculus OINC. If G and G' are the upper and lower goal in the inference rule $[\alpha]$ ($\alpha \in \{\mathrm{on}, \mathrm{d}, \mathrm{v}\}$), 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 $G \Rightarrow_{[\mathrm{on}],\, l\to r} G'$ and $G \Rightarrow_{[\mathrm{v}],\,\theta} G'$. 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 $A\colon G \Rightarrow^* G'$ is denoted by $|A|$. If $|A| \geqslant 1$ 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 5. 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.

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 Sect. 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 OINC. Finally, the absence of the symmetric outermost narrowing rule

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

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

Definition 6. Let $\mathcal{R}$ be a TRS and G a goal. A substitution θ is called a *solution* of G if $s\theta =_\mathcal{R} t\theta$ for every equation $s \approx t$ in G.

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

Theorem 7. *Let $\mathcal{R}$ be an orthogonal TRS and G a right-normal goal. If there exists a successful* OINC-*derivation $G \Rightarrow_\theta^* \Box$ then θ is a solution of G. For every normalizable solution θ of G there exists a successful* OINC-*derivation $G \Rightarrow_{\theta'}^* \Box$ such that $\theta' \leqslant_\mathcal{R} \theta\ [\mathcal{V}ar(G)]$.* $\Box$

If the substitution θ in the second part of Theorem 7 is normalized, the subscript $\mathcal{R}$ can be dropped.

4 A Narrowing Calculus for Applicative Systems

Every $\mathcal{A}$TRS is a TRS, hence OINC is complete (in the sense of the second part of Theorem 7) for orthogonal $\mathcal{A}$TRSs. However, 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 $\mathcal{A}$TRSs 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 1. The (right-normal) goal $\mathtt{c}\,x\,y \approx \mathtt{c}\,\mathtt{a}\,\mathtt{b}$ is solved by the following OINC-derivation:

$$\begin{aligned}
\mathtt{c}\,x\,y \approx \mathtt{c}\,\mathtt{a}\,\mathtt{b} &\Rightarrow_{[\mathrm{d}]} && \mathtt{c}\,x \approx \mathtt{c}\,\mathtt{a}, y \approx \mathtt{b} \\
&\Rightarrow_{[\mathrm{d}]} && \mathtt{c} \approx \mathtt{c}, x \approx \mathtt{a}, y \approx \mathtt{b} \\
&\Rightarrow_{[\mathrm{d}]} && x \approx \mathtt{a}, y \approx \mathtt{b} \\
&\Rightarrow_{[\mathrm{v}],\,\{x \mapsto \mathtt{a}\}} && y \approx \mathtt{b} \\
&\Rightarrow_{[\mathrm{v}],\,\{y \mapsto \mathtt{b}\}} && \Box.
\end{aligned}$$

Recall that the term $\mathtt{c}\,x\,y$ has the (hidden) application symbol as root with $\mathtt{c}\,x$ and y as arguments. Likewise the term $\mathtt{c}\,\mathtt{a}\,\mathtt{b}$ consists of the application of $\mathtt{c}\,\mathtt{a}$ and $\mathtt{b}$. Hence the inference rule [d] decomposes the equation $\mathtt{c}\,x\,y \approx \mathtt{c}\,\mathtt{a}\,\mathtt{b}$ into $\mathtt{c}\,x \approx \mathtt{c}\,\mathtt{a}$ and $y \approx \mathtt{b}$. We need three decomposition steps before we can bind the variables.

Example 2. Consider the orthogonal $\mathcal{A}$TRS

$$\mathcal{R} = \begin{cases} \mathtt{inc} & \to \mathtt{add1} \\ \mathtt{add1}\,x & \to \mathtt{S}\,x \end{cases}$$

The following OINC-derivation computes the solution $\{x \mapsto \mathtt{0}\}$ of the goal $\mathtt{inc}\,x \approx \mathtt{S}\,\mathtt{0}$:

$$\begin{aligned}
\mathtt{inc}\,x \approx \mathtt{S}\,\mathtt{0} &\Rightarrow_{[\mathrm{on}],\,\mathtt{add1}\,x_1 \to \mathtt{S}\,x_1} && \mathtt{inc} \approx \mathtt{add1}, x \approx x_1, \mathtt{S}\,x_1 \approx \mathtt{S}\,\mathtt{0} \\
&\Rightarrow_{[\mathrm{on}],\,\mathtt{inc} \to \mathtt{add1}} && \mathtt{add1} \approx \mathtt{add1}, x \approx x_1, \mathtt{S}\,x_1 \approx \mathtt{S}\,\mathtt{0} \\
&\Rightarrow_{[\mathrm{d}]} && x \approx x_1, \mathtt{S}\,x_1 \approx \mathtt{S}\,\mathtt{0} \\
&\Rightarrow_{[\mathrm{v}],\,\{x_1 \mapsto x\}} && \mathtt{S}\,x \approx \mathtt{S}\,\mathtt{0} \\
&\Rightarrow_{[\mathrm{d}]} && \mathtt{S} \approx \mathtt{S}, x \approx \mathtt{0} \\
&\Rightarrow_{[\mathrm{d}]} && x \approx \mathtt{0} \\
&\Rightarrow_{[\mathrm{v}],\,\{x \mapsto \mathtt{0}\}} && \Box.
\end{aligned}$$

It is essential that we choose the (renamed) rewrite rule $\mathtt{add1}\,x_1 \to \mathtt{S}\,x_1$ for the equation $\mathtt{inc}\,x \approx \mathtt{S}\,\mathtt{0}$ 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 8. Let $\mathcal{R}$ be an orthogonal $\mathcal{A}$TRS. The calculus NCA—Narrowing Calculus for Applicative TRSs—consists of the following five inference rules:

[ona] *outermost narrowing of applicative terms*

$$\frac{f\,\mathbf{s}_n\,\mathbf{t}_m \approx t, E}{s_1 \approx u_1, \ldots, s_n \approx u_n, r\,\mathbf{t}_m \approx t, E} \quad t \notin \mathcal{V}$$

if there exists a fresh variant $f\,\mathbf{u}_n \to r$ of a rewrite rule in $\mathcal{R}$,

[onv] *outermost narrowing of head-variable terms*

$$\frac{x\,\mathbf{s}_n\,\mathbf{t}_m \approx t, E}{(s_1 \approx v_1, \ldots, s_n \approx v_n, r\,\mathbf{t}_m \approx t, E)\theta} \quad t \notin \mathcal{V}$$

if there exists a fresh variant $f\,\mathbf{u}_k\,\mathbf{v}_n \to r$ of a rewrite rule in $\mathcal{R}$, $n > 0$, and $\theta = \{x \mapsto f\,\mathbf{u}_k\}$,

[da] *decomposition of applicative terms*

$$\frac{f\,\mathbf{s}_n \approx f\,\mathbf{t}_n, E}{s_1 \approx t_1, \ldots, s_n \approx t_n, E}$$

[dv] *decomposition of head-variable terms*

$$\frac{x\,\mathbf{s}_n \approx f\,\mathbf{t}_m\,\mathbf{u}_n, E}{(s_1 \approx u_1, \ldots, s_n \approx u_n, E)\theta}$$

if $\theta = \{x \mapsto f\,\mathbf{t}_m\}$,

[v] *variable elimination*

$$\frac{t \approx x, E}{E\theta}$$

if $\theta = \{x \mapsto t\}$.

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

Example 3. The goal $\mathtt{c}\,x\,y \approx \mathtt{c}\,\mathtt{a}\,\mathtt{b}$ is solved by the following NCA-derivation:

$$\begin{aligned} \mathtt{c}\,x\,y \approx \mathtt{c}\,\mathtt{a}\,\mathtt{b} &\Rrightarrow_{[\mathrm{da}]} && x \approx \mathtt{a}, y \approx \mathtt{b} \\ &\Rrightarrow_{[\mathrm{dv}],\,\{x \mapsto \mathtt{a}\}} && y \approx \mathtt{b} \\ &\Rrightarrow_{[\mathrm{dv}],\,\{y \mapsto \mathtt{b}\}} && \square. \end{aligned}$$

With respect to the $\mathcal{A}$TRS $\mathcal{R}$ of Example 2, the goal $\mathtt{inc}\,x \approx \mathtt{S}\,\mathtt{0}$ is solved by the following NCA-derivation:

$$\begin{aligned}
\mathtt{inc}\,x \approx \mathtt{S}\,\mathtt{0} &\Rrightarrow_{[\mathrm{ona}],\,\mathtt{inc}\to\mathtt{add1}} && \mathtt{add1}\,x \approx \mathtt{S}\,\mathtt{0}\\
&\Rrightarrow_{[\mathrm{ona}],\,\mathtt{add1}\,x_1\to\mathtt{S}\,x_1} && x \approx x_1, \mathtt{S}\,x_1 \approx \mathtt{S}\,\mathtt{0}\\
&\Rrightarrow_{[\mathrm{v}],\,\{x_1\mapsto x\}} && \mathtt{S}\,x \approx \mathtt{S}\,\mathtt{0}\\
&\Rrightarrow_{[\mathrm{da}]} && x \approx \mathtt{0}\\
&\Rrightarrow_{[\mathrm{dv}],\,\{x\mapsto\mathtt{0}\}} && \Box.
\end{aligned}$$

The inference rule [onv] of NCA is used to bind higher-order logical variables. This is illustrated in the next example.

Example 4. Consider the orthogonal $\mathcal{A}$TRS

$$\mathcal{R} = \begin{cases}
\mathtt{plus}\,\mathtt{0}\,x & \to x & (1)\\
\mathtt{plus}\,(\mathtt{S}\,x)\,y & \to \mathtt{S}\,(\mathtt{plus}\,x\,y) & (2)\\
\mathtt{map}\,f\,\mathtt{nil} & \to \mathtt{nil} & (3)\\
\mathtt{map}\,f\,(x:y) & \to (f\,x):(\mathtt{map}\,f\,y) & (4)
\end{cases}$$

Here $:$ is a binary constructor, written in infix notation, and $\mathtt{nil}$ is a constant denoting the empty list. We adopt the usual convention of writing $[t_1,\ldots,t_n]$ to denote the list $(t_1:(\cdots(t_n:\mathtt{nil})\cdots))$. The goal $\mathtt{map}\,x\,[\mathtt{S}\,\mathtt{0}] \approx [\mathtt{S}\,\mathtt{0}]$ is solved by the following NCA-derivation:

$$\begin{aligned}
&\mathtt{map}\,x\,[\mathtt{S}\,\mathtt{0}] \approx [\mathtt{S}\,\mathtt{0}]\\
&\quad\Rrightarrow_{[\mathrm{ona}],\,(4)} && x \approx f_1, [\mathtt{S}\,\mathtt{0}] \approx (x_1:y_1), (f_1\,x_1):(\mathtt{map}\,f_1\,y_1) \approx [\mathtt{S}\,\mathtt{0}]\\
&\quad\Rrightarrow_{[\mathrm{v}],\,\{f_1\mapsto x\}} && [\mathtt{S}\,\mathtt{0}] \approx (x_1:y_1), (x\,x_1):(\mathtt{map}\,x\,y_1) \approx [\mathtt{S}\,\mathtt{0}]\\
&\quad\Rrightarrow_{[\mathrm{da}]} && \mathtt{S}\,\mathtt{0} \approx x_1, \mathtt{nil} \approx y_1, (x\,x_1):(\mathtt{map}\,x\,y_1) \approx [\mathtt{S}\,\mathtt{0}]\\
&\quad\Rrightarrow_{[\mathrm{v}],\,\{x_1\mapsto\mathtt{S}\,\mathtt{0}\}} && \mathtt{nil} \approx y_1, (x\,(\mathtt{S}\,\mathtt{0})):(\mathtt{map}\,x\,y_1) \approx [\mathtt{S}\,\mathtt{0}]\\
&\quad\Rrightarrow_{[\mathrm{v}],\,\{y_1\mapsto\mathtt{nil}\}} && (x\,(\mathtt{S}\,\mathtt{0})):(\mathtt{map}\,x\,\mathtt{nil}) \approx [\mathtt{S}\,\mathtt{0}]\\
&\quad\Rrightarrow_{[\mathrm{da}]} && x\,(\mathtt{S}\,\mathtt{0}) \approx \mathtt{S}\,\mathtt{0}, \mathtt{map}\,x\,\mathtt{nil} \approx \mathtt{nil}\\
&\quad\Rrightarrow_{[\mathrm{onv}],\,(1),\,\{x\mapsto\mathtt{plus}\,\mathtt{0}\}} && \mathtt{S}\,\mathtt{0} \approx y_2, y_2 \approx \mathtt{S}\,\mathtt{0}, \mathtt{map}\,(\mathtt{plus}\,\mathtt{0})\,\mathtt{nil} \approx \mathtt{nil}\\
&\quad\Rrightarrow_{[\mathrm{v}],\,\{y_2\mapsto\mathtt{S}\,\mathtt{0}\}} && \mathtt{S}\,\mathtt{0} \approx \mathtt{S}\,\mathtt{0}, \mathtt{map}\,(\mathtt{plus}\,\mathtt{0})\,\mathtt{nil} \approx \mathtt{nil}\\
&\quad\Rrightarrow^{+}_{[\mathrm{da}]} && \mathtt{map}\,(\mathtt{plus}\,\mathtt{0})\,\mathtt{nil} \approx \mathtt{nil}\\
&\quad\Rrightarrow_{[\mathrm{ona}],\,(3)} && \mathtt{plus}\,\mathtt{0} \approx f_3, \mathtt{nil} \approx \mathtt{nil}, \mathtt{nil} \approx \mathtt{nil}\\
&\quad\Rrightarrow_{[\mathrm{v}],\,\{f_3\mapsto\mathtt{plus}\,\mathtt{0}\}} && \mathtt{nil} \approx \mathtt{nil}, \mathtt{nil} \approx \mathtt{nil}\\
&\quad\Rrightarrow^{+}_{[\mathrm{da}]} && \Box.
\end{aligned}$$

Note that in the $\Rrightarrow_{[\mathrm{onv}]}$-step the variable x is bound to the higher-order term $(\mathtt{plus}\,\mathtt{0})$.

Soundness of NCA is expressed in the following theorem.

Theorem 9. *Let $\mathcal{R}$ be an orthogonal TRS and G a right-normal goal. If there exists a successful* NCA*-derivation $G \Rrightarrow^{*}_{\theta} \Box$ then θ is a solution of G.*

Proof. Straightforward induction on the length of the successful NCA-derivation $G \Rrightarrow^{*}_{\theta} \Box$. ◻

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 $A\colon G \Rightarrow_\theta^+ \Box$ there exist an NCA-step $G \Rrightarrow_\sigma G'$ and an OINC-derivation $A'\colon G' \Rightarrow_{\theta'}^* \Box$ 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 10. Let $\mathcal{R}$ be an $\mathcal{A}$TRS. In the OINC-derivation

$$E \Rightarrow^* s_1\, s_2 \approx t_1\, t_2, E_1 \Rightarrow_{[\mathrm{d}]} s_1 \approx t_1, s_2 \approx t_2, E_1 \Rightarrow^* E_2$$

the equation $s_1 \approx t_1$ is called an *immediate descendant* of the equation $s_1\, s_2 \approx t_1\, t_2$. In the OINC-derivation

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

where $l_1\, l_2 \to r$ is a fresh variant of a rewrite rule in $\mathcal{R}$, the equation $s_1 \approx l_1$ is called an *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 11–16 we observe basic properties of successful OINC-derivations.

Lemma 11. *Let $G = f\,\mathbf{s}_n \approx g\,\mathbf{t}_m, E$ 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 $f\,\mathbf{s}_n \approx g\,\mathbf{t}_m$.*

Proof. Obvious. □

Lemma 12. *Let $G = x\,\mathbf{s}_n \approx g\,\mathbf{t}_m, E$ be a goal such that $m < n$. In all successful* OINC*-derivations starting from G the rule* [on] *is applied to a descendant of $x\,\mathbf{s}_n \approx g\,\mathbf{t}_m$.*

Proof. Obvious. □

Lemma 13. *Let $G = f\,\mathbf{s}_n \approx t, E$ be a goal such that $n < \mathit{arity}(f)$. There exist no successful* OINC*-derivations starting from G in which the rule* [on] *is applied to a descendant of $f\,\mathbf{s}_n \approx t$.*

Proof. We use induction on n. If $n = 0$ then there are no rewrite rules of the form $f \to r$ because $\mathit{arity}(f) > 0$ 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
$$f\,\mathbf{s}_n \approx t, E \Rightarrow_{[\mathrm{d}]} f\,\mathbf{s}_{n-1} \approx t_1, s_n \approx t_2, E \Rightarrow^* \Box$$
with $t = t_1\, t_2$. According to the induction hypothesis the inference rule [on] is not applied to a descendant of $f\,\mathbf{s}_{n-1} \approx t_1$ in the subderivation $A_{>1}$. Therefore [on] is not applied to a descendant of $f\,\mathbf{s}_n \approx t$ in A.

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

$$f\,\mathbf{s}_n \approx t, E \Rightarrow_{[\mathrm{on}],\, l_1\, l_2 \to r} f\,\mathbf{s}_{n-1} \approx l_1, s_n \approx l_2, r \approx t, E \Rightarrow^* \Box$$

for some rewrite rule $l_1\, l_2 \to r$. We have to show that this is impossible. According to the induction hypothesis the inference rule [on] is not applied to a descendant of $f\,\mathbf{s}_{n-1} \approx l_1$ 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 $l_1 = g\,\mathbf{u}_m$ with $m = \mathit{arity}(g) - 1$. We have either $f \neq g$ or $n - 1 \neq m$. Now Lemma 11 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 $f\,\mathbf{s}_n \approx t$ left in $A_{>1}$ to which [on] can be applied. Therefore [on] is not applied to a descendant of $f\,\mathbf{s}_n \approx t$ in A.

□

Lemma 14. *Let $G = f\,\mathbf{s}_n \approx f\,\mathbf{t}_n, E$ be a goal such that $n < \mathit{arity}(f)$. In every successful* OINC*-derivation starting from G the rule* [d] *is applied to all descendants of $f\,\mathbf{s}_n \approx f\,\mathbf{t}_n$.*

Proof. Easy consequence of Lemma 13. □

Lemma 15. *Let $G = f\,\mathbf{s}_n \approx t, E$ be a goal such that $n = \mathit{arity}(f)$. 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 $f\,\mathbf{u}_n \to r$.*

Proof. Easy consequence of Lemmata 11 and 13. □

Lemma 16. *Let $G = f\,\mathbf{s}_n\,\mathbf{t}_m \approx t, E$ be a goal such that $n = \mathit{arity}(f)$ and $m > 0$. If there exists a successful* OINC*-derivation A starting from G in which the rule* [on] *is applied to a descendant $f\,\mathbf{s}_n\,\mathbf{t}_k \approx t'$ $(1 \leqslant k \leqslant m)$ then* [on] *is applied to the descendant $f\,\mathbf{s}_n \approx t''$ of $f\,\mathbf{s}_n\,\mathbf{t}_m \approx t$.*

Proof. Without loss of generality we may write A as

$$\begin{aligned} G \Rightarrow^* & \quad f\,\mathbf{s}_n\,\mathbf{t}_k \approx t', E' \\ \Rightarrow_{[\mathrm{on}],\, g\,\mathbf{u}_\ell \to r} & \quad f\,\mathbf{s}_n\,\mathbf{t}_{k-1} \approx g\,\mathbf{u}_{\ell-1}, t_k \approx u_\ell, r \approx t', E' \\ \Rightarrow^*_{[\mathrm{d}]} & \quad f\,\mathbf{s}_n \approx g\,\mathbf{u}_{\ell-k}, E'' \\ \Rightarrow^*_{\theta} & \quad \Box \end{aligned}$$

where $g\,\mathbf{u}_\ell \to r$ 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 11 and 13 the inference rule [on] is applied to the equation $f\,\mathbf{s}_n \approx g\,\mathbf{u}_{\ell-k}$. □

In Lemmata 17–21 we prove that for certain successful OINC-derivation $A\colon G \Rightarrow^+_\theta \Box$ there exists an OINC-derivation $A'\colon G' \Rightarrow^*_{\theta'} \Box$ such that $G \Rrightarrow_\sigma G'$, $\theta = \sigma\theta'$, and $|A'| < |A|$. In Lemma 22 we show that there are no other cases to consider.

Lemma 17. *Let* $A\colon s \approx x, E \Rightarrow_\theta^+ \Box$ *be an* OINC*-derivation. There exists an* OINC*-derivation* $A'\colon E\sigma \Rightarrow_{\theta'}^* \Box$ *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]:

$$A\colon s \approx x, E \Rightarrow_{[\mathrm{v}],\,\sigma} E\sigma \Rightarrow_{\theta'}^* \Box$$

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 17 are connected by a $\Rrightarrow_{[\mathrm{v}],\,\sigma}$-step.

Lemma 18. *Let* $A\colon f\,\mathbf{s}_n \approx f\,\mathbf{t}_n, E \Rightarrow_\theta^+ \Box$ *be an* OINC*-derivation. If* [on] *is never applied to a descendant of* $f\,\mathbf{s}_n \approx f\,\mathbf{t}_n$ *then there exists an* OINC*-derivation* $A'\colon s_1 \approx t_1, \ldots, s_n \approx t_n, E \Rightarrow_\theta^* \Box$ *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

$$f\,\mathbf{s}_n \approx f\,\mathbf{t}_n, E \Rightarrow_{[\mathrm{d}]} f\,\mathbf{s}_{n-1} \approx f\,\mathbf{t}_{n-1}, s_n \approx t_n, E \Rightarrow_\theta^+ \Box.$$

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

$$A'\colon s_1 \approx t_1, \ldots, s_{n-1} \approx t_{n-1}, s_n \approx t_n, E \Rightarrow_\theta^* \Box$$

with $|A'| < |A_{>1}| = |A| - 1 < |A|$. □

Note that the initial goals of A and A' in Lemma 18 are connected by a $\Rrightarrow_{[\mathrm{da}]}$-step.

Lemma 19. *Let* $A\colon x\,\mathbf{s}_n \approx f\,\mathbf{t}_m\,\mathbf{u}_n \Rightarrow_\theta^+ \Box$ *be an* OINC*-derivation. If* [on] *is never applied to a descendant of* $x\,\mathbf{s}_n \approx f\,\mathbf{t}_m\,\mathbf{u}_n$ *then there exists an* OINC*-derivation* $A'\colon (s_1 \approx u_1, \ldots, s_n \approx u_n, E)\sigma \Rightarrow_{\theta'}^* \Box$ *with* $\sigma = \{x \mapsto f\,\mathbf{t}_m\}$ *such that* $\theta = \sigma\theta'$ *and* $|A'| < |A|$.

Proof. Similar to the proof of Lemma 18. □

The initial goals of A and A' in Lemma 19 are connected by a $\Rrightarrow_{[\mathrm{dv}],\,\sigma}$-step.

Lemma 20. *Let* $A\colon f\,\mathbf{s}_n\,\mathbf{t}_m \approx t \Rightarrow_\theta^+ \Box$ *be an* OINC*-derivation with* $n = \mathit{arity}(f)$. *If* [on] *is applied to the descendant* $f\,\mathbf{s}_n \approx t'$ *of* $f\,\mathbf{s}_n\,\mathbf{t}_m \approx t$ *using the rewrite rule* $f\,\mathbf{u}_n \to r$ *then there exists an* OINC*-derivation*

$$A'\colon s_1 \approx u_1, \ldots, s_n \approx u_n, r\,\mathbf{t}_m \approx t, E \Rightarrow_\theta^* \Box$$

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

Proof. We use induction on m. If $m = 0$ then the inference rule [on] with rewrite rule $f\,\mathbf{u}_n \to r$ 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

$$f\,\mathbf{s}_n \approx t, E \Rightarrow_{[\mathrm{on}]} f\,\mathbf{s}_{n-1} \approx f\,\mathbf{u}_{n-1}, s_n \approx u_n, r \approx t, E \Rightarrow_\theta^+ \Box.$$

According to Lemma 14 the inference rule [on] is not applied to descendants of $f\,\mathbf{s}_{n-1} \approx f\,\mathbf{u}_{n-1}$ in the subderivation $A_{>1}$. Hence we can apply Lemma 18 to $A_{>1}$. This yields an OINC-derivation

$$A'\colon s_1 \approx u_1, \ldots, s_{n-1} \approx u_{n-1}, s_n \approx u_n, r \approx t, E \Rightarrow_\theta^* \Box$$

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 $s \approx t, E \Rightarrow_{[\mathrm{d}]} f\,\mathbf{s}_n\,\mathbf{t}_{m-1} \approx v_1, t_m \approx v_2, E \Rightarrow_\theta^+ \Box$ with $t = v_1\,v_2$. An application of the induction hypothesis to the subderivation $A_{>1}$ yields an OINC-derivation

   $$B\colon E', r\,\mathbf{t}_{m-1} \approx v_1, t_m \approx v_2, E \Rightarrow_\theta^* \Box$$

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

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

   and

   $$B_2\colon (r\,\mathbf{t}_{m-1} \approx v_1, t_m \approx v_2, E)\theta_1 \Rightarrow_{\theta_2}^* \Box$$

   with $\theta = \theta_1\theta_2$. The OINC-derivation B_1 can easily be transformed into the OINC-derivation

   $$C\colon E', r\,\mathbf{t}_m \approx v_1\,v_2, E \Rightarrow_{\theta_1}^* (r\,\mathbf{t}_m \approx v_1\,v_2, E)\theta_1.$$

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

   $$D\colon (r\,\mathbf{t}_m \approx v_1\,v_2, E)\theta_1 \Rightarrow_{[\mathrm{d}]} (r\,\mathbf{t}_{m-1} \approx v_1, t_m \approx v_2, E)\theta_1.$$

   Concatenating the three OINC-derivations C, D, and B_2 yields the desired OINC-derivation

   $$A'\colon E', r\,\mathbf{t}_m \approx t, E \Rightarrow_\theta^* \Box.$$

   Note that $|A'| = |C| + |D| + |B_2| = |B| + 1 < |A|$.
2. Suppose the inference rule [on] is used in the first step of A. This means that there exists a fresh variant $v_1\,v_2 \to r'$ of a rewrite rule in $\mathcal{R}$ such that A can be written as

   $$s \approx t, E \Rightarrow_{[\mathrm{on}]} f\,\mathbf{s}_n\,\mathbf{t}_{m-1} \approx v_1, t_m \approx v_2, r' \approx t, E \Rightarrow_\theta^+ \Box$$

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

   $$B\colon E', r\,\mathbf{t}_{m-1} \approx v_1, t_m \approx v_2, r' \approx t, E \Rightarrow_\theta^* \Box$$

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

$$B_1\colon E', r\,\mathbf{t}_{m-1} \approx v_1, t_m \approx v_2, r' \approx t, E \Rrightarrow^*_{\theta_1} (r\,\mathbf{t}_{m-1} \approx v_1, t_m \approx v_2, r' \approx t, E)\theta_1$$

and

$$B_2\colon (r\,\mathbf{t}_{m-1} \approx v_1, t_m \approx v_2, r' \approx t, E)\theta_1 \Rrightarrow^*_{\theta_2} \Box$$

with $\theta = \theta_1\theta_2$. We transform B_1 into the OINC-derivation

$$C\colon E', r\,\mathbf{t}_m \approx t, E \Rrightarrow^*_{\theta_1} (r\,\mathbf{t}_m \approx t, E)\theta_1.$$

Next we apply the inference rule [on] to the final goal $(r\,\mathbf{t}_m \approx t, E)\theta_1$ of C, using exactly the same variant $v_1\,v_2 \to r'$. This yields the OINC-step

$$D\colon (r\,\mathbf{t}_m \approx t, E)\theta_1 \Rrightarrow_{[\mathrm{on}]} (r\,\mathbf{t}_{m-1} \approx v_1, t_m \approx v_2, r' \approx t, E)\theta_1.$$

Also in this case the desired 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 [v] because then there is no descendant left to which [on] can be applied.

□

Observe that the initial goals of A and A' in Lemma 20 are connected by a $\Rrightarrow_{[\mathrm{ona}]}$-step.

Lemma 21. *Let $A\colon x\,\mathbf{s}_n\,\mathbf{t}_m \approx t, E \Rrightarrow^+_\theta \Box$ be an* OINC*-derivation with $n > 0$. If* [on] *is applied to the descendant $x\,\mathbf{s}_n \approx t'$ of $x\,\mathbf{s}_n\,\mathbf{t}_m \approx t$ using the rewrite rule $f\,\mathbf{u}_k\,\mathbf{v}_n \to r$ and $x\,\mathbf{s}_n \approx t'$ is the last descendant to which* [on] *is applied then there exists an* OINC*-derivation $A'\colon (s_1 \approx v_1, \ldots, s_n \approx v_n, r\,\mathbf{t}_m \approx t, E)\sigma \Rrightarrow^*_{\theta'} \Box$ with $\sigma = \{x \mapsto f\,\mathbf{u}_k\}$ such that $\theta = \sigma\theta'$ and $|A'| < |A|$.*

Proof. Similar to the proof of Lemma 20. □

In Lemma 21 the initial goals of A and A' are connected by a $\Rrightarrow_{[\mathrm{onv}],\,\sigma}$-step.

Lemma 22. *Let G be a proper goal. For every* OINC*-derivation $A\colon G \Rrightarrow^+_\theta \Box$ there exist an* NCA*-step $G \Rrightarrow_\sigma G'$ and an* OINC*-derivation $A'\colon G' \Rrightarrow^*_{\theta'} \Box$ such that $\theta = \sigma\theta'$ and $|A'| < |A|$.*

Proof. We have to show that Lemmata 17–21 cover all possible cases. Let G be the goal $s \approx t, E$. The case that t is a variable is covered by Lemma 17, 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 [on] is not applied to a descendant of $s \approx t$.
   (a) If the head-symbol of s is a constant f then, according to Lemma 11, we must have $s = f\,\mathbf{s}_n$ and $t = f\,\mathbf{t}_n$. This case is covered by Lemma 18.
   (b) If the head-symbol of s is a variable x then, according to Lemma 12, $s = x\,\mathbf{s}_n$ and $t = f\,\mathbf{t}_m\,\mathbf{u}_n$. This case is covered by Lemma 19.

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 13, we have $s = f\,\mathbf{s}_n\,\mathbf{t}_m$ with $n = \mathit{arity}(f)$. From Lemma 13 we also infer that [on] is never applied to descendants of the form $f\,\mathbf{s}_k \approx t'$ with $k < \mathit{arity}(f)$. Hence the application of [on] is to a descendant of the form $f\,\mathbf{s}_n\,\mathbf{t}_k \approx t''$ with $0 \leqslant k \leqslant m$. According to Lemma 16 [on] is applied to the descendant $f\,\mathbf{s}_n \approx t'''$. Lemma 15 states that the employed rewrite rule is of the form $f\,\mathbf{u}_n \to r$. Hence this case is covered by Lemma 20.
   (b) The case that the head-symbol of s is a variable x is covered by Lemma 21, using similar reasoning as in the previous case.

□

The completeness of NCA is an easy consequence of the previous lemma and the completeness of OINC (Theorem 7).

Theorem 23. *Let $\mathcal{R}$ be an orthogonal $\mathcal{A}$TRS and G a right-normal goal. For every normalizable solution θ of G there exists a successful* NCA*-derivation $G \Rrightarrow^*_{\theta'} \Box$ such that $\theta' \leqslant_{\mathcal{R}} \theta\ [\mathcal{V}ar(G)]$.*

Proof. According to the second part of Theorem 7 there exists a successful OINC-derivation $A\colon G \Rightarrow^*_{\theta'} \Box$ such that $\theta' \leqslant_{\mathcal{R}} \theta\ [\mathcal{V}ar(G)]$. By induction on $|A|$ we show the existence of a successful NCA-derivation $G \Rrightarrow^*_{\theta'} \Box$. The case $|A| = 0$ is trivial. Suppose $|A| > 0$. According to Lemma 22 there exist an NCA-step $G \Rrightarrow_\sigma G'$ and an OINC-derivation $A'\colon G' \Rightarrow^*_{\theta''} \Box$ such that $\theta' = \sigma\theta''$ and $|A'| < |A|$. The induction hypothesis yields an successful NCA-derivation $G' \Rrightarrow^*_{\theta''} \Box$. Combining this derivation with the NCA-step $G \Rrightarrow_\sigma G'$ yields the desired NCA-derivation $G \Rrightarrow^*_{\theta'} \Box$. □

Inspection of the above proofs reveals that the length of the resulting NCA-derivation $G \Rrightarrow^* \Box$ never exceeds the length of the OINC-derivation $G \Rightarrow^* \Box$. Moreover, it is not difficult to see that for every application of Lemmata 18–21 in the transformation from $G \Rightarrow^* \Box$ to $G \Rightarrow^* \Box$ we gain n steps (corresponding to applications of the inference rule [d] in the OINC-derivation $G \Rightarrow^* \Box$).

6 Incorporating Strict Equality into NCA

In functional logic programming languages like K-LEAF [4] and 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 *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

$$\begin{cases} c \equiv c & \to \mathtt{true} & \text{if } c \text{ is a nullary constructor,} \\ c\,\mathbf{x}_n \equiv c\,\mathbf{y}_n & \to x_1 \equiv y_1 \wedge \cdots \wedge x_n \equiv y_n & \text{if } c \text{ is an } n\text{-ary constructor with } n > 0, \\ \mathtt{true} \wedge x & \to x \end{cases}$$

to a given $\mathcal{A}$TRS $\mathcal{R}$, resulting in the $\mathcal{A}$TRS $\mathcal{R}_s$. Here $\equiv$ denotes strict equality and $\wedge$ 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 $(s \equiv t) \approx \mathtt{true}$, which we abbreviate to $s \approx_s t$. 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 NCA and the $\mathcal{A}$TRS $\mathcal{R}_s$. However, as observed by Ida and Nakahara [7] in the context of 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 NCA in the following definition.

Definition 24. Let $\mathcal{R}$ be an orthogonal $\mathcal{A}$TRS. The calculus NCA_s is obtained by adding the following inference rules to NCA:

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

$$\frac{f\,\mathbf{s}_n\,\mathbf{t}_m \simeq_s t, E}{s_1 \approx u_1, \ldots, s_n \approx u_n, r\,\mathbf{t}_m \approx_s t, E}$$

if there exists a fresh variant $f\,\mathbf{u}_n \to r$ of a rewrite rule in $\mathcal{R}$,

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

$$\frac{x\,\mathbf{s}_n\,\mathbf{t}_m \simeq_s t, E}{(s_1 \approx v_1, \ldots, s_n \approx v_n, r\,\mathbf{t}_m \approx_s t, E)\theta}$$

if there exists a fresh variant $f\,\mathbf{u}_k\,\mathbf{v}_n \to r$ of a rewrite rule in $\mathcal{R}$, $n > 0$, and $\theta = \{x \mapsto f\,\mathbf{u}_k\}$,

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

$$\frac{c\,\mathbf{s}_n \approx_s c\,\mathbf{t}_n, E}{s_1 \approx_s t_1, \ldots, s_n \approx_s t_n, E}$$

if c is an n-ary constructor symbol,

[imas] *imitation for strict equations*

$$\frac{x\,\mathbf{s}_n \simeq_s c\,\mathbf{t}_m\,\mathbf{u}_n, E}{(x_1 \approx_s t_1, \ldots, x_m \approx_s t_m, s_1 \approx_s u_1, \ldots, s_n \approx_s u_n, E)\theta}$$

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

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

$$\frac{x\,\mathbf{s}_n \simeq_s y\,\mathbf{t}_m\,\mathbf{u}_n, E}{(x_1 \approx_s t_1, \ldots, x_m \approx_s t_m, s_1 \approx_s u_1, \ldots, s_n \approx_s u_n, E)\theta}$$

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

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

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

Example 5. Let $\mathcal{R} = \{\mathtt{a} \to \mathtt{b}\}$. The following NCA_s-derivation, starting from the goal $G = x\,\mathtt{a}\,\mathtt{a} \approx_s y\,\mathtt{a}$, produces the substitution $\sigma = \{y \mapsto x\,\mathtt{b}\}$:

$$\begin{array}{lll} G \Rrightarrow_{[\text{dvs}],\,\{y \mapsto x\,x_1\}} \mathtt{a} \approx_s x_1, \mathtt{a} \approx_s \mathtt{a} & \Rrightarrow_{[\text{onas}],\,\mathtt{a}\to\mathtt{b}} \mathtt{b} \approx_s x_1, \mathtt{a} \approx_s \mathtt{a} & \\ \quad \Rrightarrow_{[\text{imas}],\,\{x_1 \mapsto \mathtt{b}\}} \mathtt{a} \approx_s \mathtt{a} & \Rrightarrow_{[\text{onas}],\,\mathtt{a}\to\mathtt{b}} \mathtt{b} \approx_s \mathtt{a} & \\ \quad \Rrightarrow_{[\text{onas}],\,\mathtt{a}\to\mathtt{b}} \mathtt{b} \approx_s \mathtt{b} & \Rrightarrow_{[\text{das}]} \Box & \end{array}$$

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

Below we state the completeness of NCA_s. The proof, which is essentially the same as the completeness proof of the calculus S-OINC studied in [7], is omitted.

Theorem 25. *Let $\mathcal{R}$ be an orthogonal $\mathcal{A}$TRS and G a right-normal goal. For every normalizable solution θ of G there exists a successful NCA_s-derivation $G \Rrightarrow^*_{\theta'} \Box$ such that $\theta' \leqslant_{\mathcal{R}} \theta\ [\mathcal{V}ar(G)]$.* □

7 Experimental Results

In this section we compare the performance of NCA and OINC on a small example. We have implemented both calculi in Sicstus Prolog 2.1. We solved goals of the form

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

with respect to the example program in Sect. 1. Here $\mathtt{S}^n\,\mathtt{0}$ denotes the term

$$\underbrace{\mathtt{S}\,(\mathtt{S}\,(\cdots(\mathtt{S}}_{n}\ \mathtt{0})\cdots)).$$

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 $\{f \mapsto \mathtt{compose}\,\mathtt{S}\,\mathtt{double}\}$. 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 NCA has a much better performance than OINC.

Table 1. Comparison between OINC and NCA

n	OINC	NCA
1	3800 msec. (73 steps)	120 msec. (42 steps)
2	5448 msec. (136 steps)	179 msec. (77 steps)
3	7401 msec. (210 steps)	250 msec. (118 steps)
4	8799 msec. (295 steps)	305 msec. (165 steps)
5	10769 msec. (391 steps)	380 msec. (218 steps)

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 setting of functional (logic) programming with lazy semantics. Because of the absence of (λ-)abstraction, applicative term rewriting cannot express all higher-order features. Prehofer [13] describes a full higher-order lazy narrowing calculus.

Although NCA and 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 [ona] and [da] apply to certain goals. One way to remove this particular non-determinism is by restricting ourselves to $\mathcal{A}$CSs. The restriction to $\mathcal{A}$CSs 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 [onv] and [onvs] themselves. In the worst case there are $\mathit{arity}(f)-1$ different ways to apply the two inference rules with respect to a given rewrite rule $f\,\mathbf{u}_k\,\mathbf{v}_n \to r$. 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 $f\,\mathbf{u}_k\,\mathbf{v}_n \to r$ such that the type of the head-variable x is (unifiable with) the type of the term $f\,\mathbf{u}_k$. For example, consider the $\mathcal{A}$CS $\{\mathtt{plus}\,\mathtt{0}\,x \to x\ (1), \mathtt{plus}\,(\mathtt{S}\,x)\,y \to \mathtt{S}\,(\mathtt{plus}\,x\,y)\ (2)\}$ and the goal $z\,\mathtt{0}\,(\mathtt{S}\,\mathtt{0}) \approx \mathtt{S}\,\mathtt{0}$. In the untyped system presented in this paper there are four different ways to apply the inference rule [onv]. Only one of them leads to a successful derivation. One of the three unsuccessful applications of [onv] is

$$z\,\mathtt{0}\,(\mathtt{S}\,\mathtt{0}) \approx \mathtt{0} \Rrightarrow_{[\mathrm{onv}],\,(2),\,\{z \mapsto \mathtt{plus}\,(\mathtt{S}\,x)\}} \mathtt{0} \approx y, \mathtt{S}\,(\mathtt{plus}\,x\,y)\,(\mathtt{S}\,\mathtt{0}) \approx \mathtt{S}\,\mathtt{0}.$$

Notice that the term $\mathtt{S}\,(\mathtt{plus}\,x\,y)\,(\mathtt{S}\,\mathtt{0})$ cannot be typed. In a (polymorphically) typed system we can actually avoid the above [onv] step by letting the type of the variable z be $\mathbf{Nat} \to \mathbf{Nat} \to \mathbf{Nat}$ and observing that the type $\mathbf{Nat} \to \mathbf{Nat}$ of the term $\mathtt{plus}\,(\mathtt{S}\,x)$ isn't unifiable with the type of z. So by type-checking the substitution $\theta = \{x \mapsto f\,\mathbf{u}_k\}$ in [onv] steps we can avoid invalid ones and hence (significantly) reduce the search space.

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

Acknowledgements. We thank the referees for their constructive comments on an earlier version of this paper. This work is partially supported by the Grant-in-Aid for Scientific Research (C) 06680300 and the Grant-in-Aid for Encouragement of Young Scientists 06780229 of the Ministry of Education, Science and Culture of Japan.

References

1. S. Antoy, R. Echahed, and M. Hanus, *A Needed Narrowing Strategy*, Proceedings of the 21st ACM Symposium on Principles of Programming Languages, Portland, pp. 268–279, 1994.
2. P.G. Bosco and E. Giovannetti, *IDEAL: An Ideal Deductive Applicative Language*, Proceedings of the IEEE International Symposium on Logic Programming, pp. 89–94, 1986.
3. N. Dershowitz and J.-P. Jouannaud, *Rewrite Systems*, in: Handbook of Theoretical Computer Science, Vol. B (ed. J. van Leeuwen), North-Holland, pp. 243–320, 1990.
4. E. Giovannetti, G. Levi, C. Moiso, and C. Palamidessi, *Kernel-LEAF: A Logic plus Functional Language*, Journal of Computer and System Sciences **42**(2), pp. 139–185, 1991.
5. Juan Carlos González-Moreno, M.T. Hortalá-González, and M. Rodríguez-Artalejo, *On the Completeness of Narrowing as the Operational Semantics of Functional Logic Programming*, Proceedings of the 6th Workshop on Computer Science Logic, San Miniato, Lecture Notes in Computer Science **702**, pp. 216–230, 1992.
6. M. Hanus, *The Integration of Functions into Logic Programming: From Theory to Practice*, Journal of Logic Programming **19 & 20**, pp. 583–628, 1994.
7. T. Ida and K. Nakahara, *Leftmost Outside-In Narrowing Calculi*, report ISE-TR-94-107, University of Tsukuba, 1994. To appear in the Journal of Functional Programming.
8. J.W. Klop, *Term Rewriting Systems*, in: Handbook of Logic in Computer Science, Vol. II (eds. S. Abramsky, D. Gabbay and T. Maibaum), Oxford University Press, pp. 1–116, 1992.
9. J.J. Moreno-Navarro and M. Rodrìguez-Artalejo, *Logic Programming with Functions and Predicates: The Language BABEL*, Journal of Logic Programming **12**, pp. 191–223, 1992.
10. J.J. Moreno-Navarro, H. Kuchen, R. Loogen, and M. Rodrìguez-Artalejo, *Lazy Narrowing in a Graph Machine*, Proceedings of the 2nd International Conference on Algebraic and Logic Programming, Nancy, Lecture Notes in Computer Science **463**, pp. 298–317, 1990.
11. G. Nadathur and D. Miller, *An Overview of λ-Prolog*, Proceedings of the 5th International Conference on Logic Programming, MIT Press, pp. 810–827, 1988.

12. S. Okui, A. Middeldorp, and T. Ida, *Lazy Narrowing: Strong Completeness and Eager Variable Elimination*, Proceedings of the 20th Colloquium on Trees in Algebra and Programming, Aarhus, Lecture Notes in Computer Science **915**, pp. 394–408, 1995.
13. C. Prehofer, *Higher-Order Narrowing*, Proceedings of the 9th IEEE Symposium on Logic in Computer Science, Paris, pp. 507–516, 1994.
14. U.S. Reddy, *Narrowing as the Operational Semantics of Functional Languages*, Proceedings of the IEEE International Symposium on Logic Programming, Boston, pp. 138–151, 1985.