

Refining Weakly Outermost-Needed Rewriting and Narrowing*

Santiago Escobar

sescobar@dsic.upv.es

Dept. of Computer Systems and Computation
Technical University of Valencia
Camino de Vera, s/n
E-46022 Valencia, Spain

ABSTRACT

Outermost-needed rewriting/narrowing is a sound and complete optimal demand-driven strategy for the class of inductively sequential constructor systems. Its parallel extension (known as *weakly*) deals with non-inductively sequential constructor systems. In this paper, we present *natural rewriting*, a suitable extension of (weakly) outermost-needed rewriting which is based on a refinement of the demandness notion associated to the latter, and we extend it to narrowing. Intuitively, natural rewriting (narrowing) always reduces (narrows) the most often demanded position in a term. We formalize the strategy for left-linear constructor systems though, for the class of inductively sequential constructor systems, natural rewriting (narrowing) behaves even better than outermost-needed rewriting (narrowing) in the avoidance of failing computations. With regard to inductively sequential constructor systems, we introduce a larger class of systems called *inductively sequential preserving* where natural rewriting and narrowing preserve optimality for sequential parts of the program. We also provide a prototype interpreter of natural rewriting and narrowing.

Categories & Subject Descriptors

D.1.1 [Software]: Applicative (Functional) Programming; D.1.6 [Software]: Logic Programming; D.3.1 [Programming Languages]: Formal Definitions and Theory (D.2.1, F.3.1, F.3.2, F.4.2, F.4.3); F.3.1 [Logics and Meanings of Programs]: Specifying and Verifying and Reasoning about Programs (D.2.1, D.2.4, D.3.1, E.1); F.3.2 [Logics and Meanings of Programs]: Semantics of Programming Languages (D.3.1); I.1.3 [Symbolic and Algebraic Manipulation]: Languages and Systems (D.3.2, D.3.3, F.2.2)

*Work partially supported by CICYT TIC2001-2705-C03-01, and MCYT grants HA2001-0059 and HU2001-0019.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PPDP'03, August 27–29, 2003, Uppsala, Sweden.

Copyright 2003 ACM 1-58113-705-2/03/0008 ...\$5.00.

General Terms

Languages, Performance, Theory, Verification

Keywords

Term rewriting, lazy evaluation, demandness, neededness, parallel evaluation

1. INTRODUCTION

A challenging problem in modern programming languages is the discovery of sound and complete evaluation strategies which are ‘optimal’ w.r.t. some efficiency criterion (typically the number of evaluation steps and the avoidance of infinite, failing or redundant derivations) and which are easily implementable.

For orthogonal term rewriting systems (TRSs), Huet and Lévy’s *needed rewriting* is optimal [17]. However, automatic detection of needed redexes is difficult (or even impossible) and Huet and Lévy defined the notion of ‘strong sequentiality’, which provides a formal basis for the mechanization of sequential, normalizing rewriting computations [17]. In [17], Huet and Lévy defined the (computable) notion of strongly needed redexes and showed that, for the (decidable) class of strongly sequential orthogonal TRSs, the steady reduction of strongly needed redexes is normalizing. When strongly sequential TRSs are restricted to constructor systems (CSs), we obtain the class of inductively sequential CSs (see [16]). Intuitively, a CS is inductively sequential if there exists some branching selection structure in the rules. Sekar and Ramakrishnan provided *parallel needed reduction* [24] as the extension of Huet and Lévy needed rewriting for non-inductively sequential CSs (namely, almost orthogonal CSs).

A sound and complete rewrite strategy for the class of inductively sequential CSs is *outermost-needed rewriting* [4]. The extension to narrowing is called *outermost-needed narrowing* [7]. Its optimality properties and the fact that inductively sequential CSs are a subclass of strongly sequential programs explains why outermost-needed narrowing has become useful in functional logic programming as the functional logic counterpart of Huet and Lévy’s strongly needed reduction. *Weakly outermost-needed rewriting* [6] is defined for non-inductively sequential CSs. The extension to narrowing is called *weakly outermost-needed narrowing* [6] and

is considered as the functional logic counterpart of Sekar and Ramakrishnan’s parallel needed reduction.

Unfortunately, weakly outermost-needed rewriting and narrowing do not work appropriately on non-inductively sequential CSs.

EXAMPLE 1. Consider Berry’s program [24] where T and F are constructor symbols and X is a variable:

$$B(T, F, X) = T \quad B(F, X, T) = T \quad B(X, T, F) = T$$

This CS is not inductively sequential since there is no branching selection structure in the rules. Although the CS is not inductively sequential, some terms can be reduced sequentially; where ‘to be reduced sequentially’ is understood as the property of reducing only positions which are unavoidable (or “needed”) when attempting to obtain a normal form [17]. For instance, the term $B(B(T, F, T), B(F, T, T), F)$ has a unique ‘optimal’ rewrite sequence which achieves its associated normal form T :

$$B(B(T, F, T), B(F, T, T), F) \rightarrow B(B(T, F, T), T, F) \rightarrow T$$

However, weakly outermost-needed rewriting is not optimal since, besides the previous optimal sequence, the sequence

$$B(B(T, F, T), B(F, T, T), F) \rightarrow B(T, B(F, T, T), F) \rightarrow B(T, T, F) \rightarrow T$$

is also obtained. The reason is that weakly outermost-needed rewriting partitions the CS into inductively sequential subsets $\mathcal{R}_1 = \{B(X, T, F) = T\}$ and $\mathcal{R}_2 = \{B(T, F, X) = T, B(F, X, T) = T\}$ in such a way that the first step of the former (optimal) rewriting sequence is obtained w.r.t. subset \mathcal{R}_1 whereas the first step of the latter (useless) rewriting sequence is obtained w.r.t. subset \mathcal{R}_2 .

Note that the problem also occurs in weakly outermost-needed narrowing. For instance, term $B(X, B(F, T, T), F)$ has the optimal narrowing sequence

$$B(X, B(F, T, T), F) \rightsquigarrow_{id} B(X, T, F) \rightsquigarrow_{id} T$$

whereas the non-optimal narrowing sequence is

$$B(X, B(F, T, T), F) \rightsquigarrow_{\{X \mapsto T\}} B(T, T, F) \rightsquigarrow_{id} T$$

As Sekar and Ramakrishnan argued in [24], strongly sequential systems have been widely used in programming languages and violations are not frequent. However, it is a cumbersome restriction and its relaxation is quite useful in some contexts. It would be convenient for programmers that such a restriction could be relaxed while optimal evaluation is preserved for strongly sequential parts of a program. The availability of optimal and effective evaluation strategies without such a restriction could encourage programmers to formulate non-inductively sequential programs, which could still be executed in an optimal way.

EXAMPLE 2. Consider the problem of coding the rather simple inequality

$$x + y + z + w \leq h$$

for natural numbers x, y, z, w , and h . A first (and naïve) approach could be to define an inductively sequential program:

$$\begin{aligned} \text{solve}(X, Y, Z, W, H) &= (((X + Y) + Z) + W) \leq H \\ 0 \leq N &= \text{True} & 0 + N &= N \\ s(M) \leq 0 &= \text{False} & s(M) + N &= s(M + N) \\ s(M) \leq s(N) &= M \leq N \end{aligned}$$

Consider the (auxiliary) factorial function $n!$. The pro-

gram is efficient for term¹ $t_1 = \text{solve}(1, Y, 0, 10!, 0)$ since **False** is returned in 5 rewrite steps. However, it is not very efficient for terms $t_2 = \text{solve}(10!, 0, 1, 0+1, 0)$ or $t_3 = \text{solve}(10!, 0+1, 0+1, W, 1)$ since several rewrite steps should be performed on $10!$ before realizing that both terms rewrite to **False**.

When a more effective program is desired for some input terms, an ordinary solution is to include some assertions by hand into the program while preserving determinism of computations. For instance, if terms t_1 , t_2 , and t_3 are part of such input terms of interest, we could obtain the following program

$$\begin{aligned} \text{solve}(X, s(Y), s(Z), W, s(H)) &= \text{solve}(X, Y, s(Z), W, H) \\ \text{solve}(X, 0, s(Z), W, s(H)) &= \text{solve}(X, 0, Z, W, H) \\ \text{solve}(s(X), Y, 0, W, s(H)) &= \text{solve}(X, Y, 0, W, H) \\ \text{solve}(0, s(Y), 0, W, s(H)) &= \text{solve}(0, Y, 0, W, H) \\ \text{solve}(0, 0, 0, 0, H) &= \text{True} \\ \text{solve}(s(X), Y, 0, W, 0) &= \text{False} \\ \text{solve}(X, 0, s(Z), W, 0) &= \text{False} \\ \text{solve}(0, s(Y), Z, 0, 0) &= \text{False} \\ \text{solve}(X, s(Y), s(Z), s(W), 0) &= \text{False} \end{aligned}$$

Note that this program is not inductively sequential but orthogonal, and some terms can still be reduced (or narrowed) sequentially. For instance, when we consider the previous term t_3 for rewriting:

$$\begin{aligned} \text{solve}(10!, 0+s(0), 0+s(0), W, s(0)) \\ \rightarrow \text{solve}(10!, 0+s(0), s(0), W, s(0)) \\ \rightarrow \text{solve}(10!, s(0), s(0), W, s(0)) \\ \rightarrow \text{solve}(10!, 0, s(0), W, 0) \rightarrow \text{False} \end{aligned}$$

or when we consider the term $t_4 = \text{solve}(X, Y, 0+2, W, 1)$ for narrowing²:

$$\begin{aligned} \text{solve}(X, Y, 0+s(s(0)), W, s(0)) \\ \rightsquigarrow_{id} \text{solve}(X, Y, s(s(0)), W, s(0)) \\ \rightsquigarrow_{\{Y \mapsto 0\}} \text{solve}(X, 0, s(0), W, 0) \\ \quad | \rightsquigarrow_{\{Y \mapsto s(Y')\}} \text{solve}(X, Y', s(s(0)), W, 0) \\ \rightsquigarrow_{id} \text{False} \quad | \rightsquigarrow_{\{Y' \mapsto 0\}} \text{False} \\ \quad | \rightsquigarrow_{\{Y' \mapsto s(Y''), W \mapsto 0, X \mapsto 0\}} \text{False} \\ \quad | \rightsquigarrow_{\{Y' \mapsto s(Y''), W \mapsto s(W'')\}} \text{False} \end{aligned}$$

Indeed, note that even in the case we add the following rule $\text{solve}(X, s(Y), 0, W, 0) = \text{False}$, which makes the program almost orthogonal, t_3 is still sequentially rewritten and t_4 sequentially narrowed.

Modern (multiparadigm) programming languages apply computational strategies which are based on some notion of demandness of a position in a term by a rule (see [8] for a survey discussing this topic). Programs in these languages are commonly modeled by left-linear CSs and computational strategies take advantage of this constructor condition (see [3, 22]). Furthermore, the semantics of these programs normally promotes the computation of values (or constructor head-normal forms) rather than head-normal forms. For instance, (weakly) outermost-needed rewriting and narrowing consider the constructor condition though some refinement is still possible, as shown by the following example.

EXAMPLE 3. Consider the following TRS from [10] defining the symbol \div , which encodes the division function be-

¹Natural numbers $1, 2, \dots$ are used as shorthand for $s(s(\dots s(0)))$, where s is applied $1, 2, \dots$ times.

²The symbol ‘|’ is used as a separator to denote different narrowing steps from a similar term.

tween natural numbers.

$$\begin{aligned} 0 \div \mathfrak{s}(\mathbb{N}) &= 0 & \mathbb{M} - 0 &= \mathbb{M} \\ \mathfrak{s}(\mathbb{M}) \div \mathfrak{s}(\mathbb{N}) &= \mathfrak{s}((\mathbb{M}-\mathbb{N}) \div \mathfrak{s}(\mathbb{N})) & \mathfrak{s}(\mathbb{M}) - \mathfrak{s}(\mathbb{N}) &= \mathbb{M}-\mathbb{N} \end{aligned}$$

Consider the term $t = 10! \div 0$, which is a (non-constructor) head-normal form. Outermost-needed rewriting forces³ the reduction of the first argument and evaluates $10!$, which is useless. The reason is that outermost-needed rewriting uses a data structure called definitional tree which encodes the branching selection structure existing in the rules without testing whether the rules associated to each branch could ever be matched to the term or not. A similar problem occurs when narrowing the term $\mathbb{X} \div 0$, since variable \mathbb{X} is instantiated to 0 or \mathfrak{s} . However, neither instantiation is really necessary.

In this paper, we provide a refinement of the demandness notion associated to outermost-needed rewriting and narrowing. We introduce an improvement w.r.t. outermost-needed rewriting called *natural rewriting* and we extend it to narrowing. Intuitively, natural rewriting (narrowing) always reduces (narrows) the most often demanded position in a term. Our definition applies to left-linear CSs. However, for the class of inductively sequential CSs, natural rewriting (narrowing) behaves even better than outermost-needed rewriting (narrowing) in the avoidance of failing computations. Regarding inductively sequential CSs, we introduce a larger class of CSs called *inductively sequential preserving* where natural rewriting and narrowing preserve optimality for sequential parts of the program. We also provide a prototype interpreter of natural rewriting and narrowing.

2. PRELIMINARIES

We assume some familiarity with term rewriting (see [26] for missing definitions) and narrowing (see [13] for missing definitions). Let $R \subseteq A \times A$ be a binary relation on a set A . We denote the reflexive closure of R by $R^=$, its transitive closure by R^+ , and its reflexive and transitive closure by R^* . An element $a \in A$ is an R -normal form, if there exists no b such that $a R b$. We say that b is an R -normal form of a (written $a R^! b$), if b is an R -normal form and $a R^* b$.

Throughout the paper, \mathcal{X} denotes a countable set of variables $\{x, y, \dots\}$ and \mathcal{F} denotes a signature, i.e. a set of function symbols $\{f, g, \dots\}$. We denote the set of terms built from \mathcal{F} and \mathcal{X} by $\mathcal{T}(\mathcal{F}, \mathcal{X})$. A term is said to be linear if it has no multiple occurrences of a single variable. Let $\text{Subst}(\mathcal{T}(\mathcal{F}, \mathcal{X}))$ denote the set of substitutions. We denote by id the “identity” substitution: $id(x) = x$ for all $x \in \mathcal{X}$. Terms are ordered by the preorder \leq of “relative generality”, i.e. $s \leq t$ if there exists σ s.t. $\sigma(s) = t$. Term t is a variant of s if $t \leq s$ and $s \leq t$. A substitution σ is more general than θ , denoted by $\sigma \leq \theta$, if $\theta = \sigma\gamma$ for some substitution γ .

By $\text{Pos}(t)$ we denote the set of positions of a term t . Given a set $S \subseteq \mathcal{F} \cup \mathcal{X}$, $\text{Pos}_S(t)$ denotes positions in t where symbols in S occur. We denote the root position by Λ . Given positions p, q , we denote its concatenation as $p.q$. Positions are ordered by the standard prefix ordering \leq . The subterm at position p of t is denoted as $t|_p$, and $t[s]_p$ is the term t

³Indeed, this specific fact depends on how a real implementation builds definitional trees. However, as shown in Example 21 below, the problem persists for other terms.

with the subterm at position p replaced by s . The symbol labeling the root of t is denoted as $\text{root}(t)$.

A rewrite rule is an ordered pair (l, r) , written $l \rightarrow r$, with $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $l \notin \mathcal{X}$. The left-hand side (*lhs*) of the rule is l and r is the right-hand side (*rhs*). A TRS is a pair $\mathcal{R} = (\mathcal{F}, R)$ where R is a set of rewrite rules. $L(\mathcal{R})$ denotes the set of *lhs*'s of \mathcal{R} . A TRS \mathcal{R} is left-linear if for all $l \in L(\mathcal{R})$, l is a linear term. Given $\mathcal{R} = (\mathcal{F}, R)$, we take \mathcal{F} as the disjoint union $\mathcal{F} = \mathcal{C} \uplus \mathcal{D}$ of symbols $c \in \mathcal{C}$, called *constructors* and symbols $f \in \mathcal{D}$, called *defined functions*, where $\mathcal{D} = \{\text{root}(l) \mid l \rightarrow r \in R\}$ and $\mathcal{C} = \mathcal{F} - \mathcal{D}$. A pattern is a term $f(l_1, \dots, l_k)$ where $f \in \mathcal{D}$ and $l_i \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, for $1 \leq i \leq k$. A TRS $\mathcal{R} = (\mathcal{C} \uplus \mathcal{D}, R)$ is a constructor system (CS) if all *lhs*'s are patterns. A term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ rewrites to s (at position p), written $t \xrightarrow{p}_{l \rightarrow r} s$ (or just $t \rightarrow s$), if $t|_p = \sigma(l)$ and $s = t[\sigma(r)]_p$, for some rule $l \rightarrow r \in R$, $p \in \text{Pos}(t)$ and substitution σ . The subterm $\sigma(l)$ in t is called a *redex*. On the other hand, a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ narrows to s (at position p with substitution σ), written $t \rightsquigarrow_{\{p, \sigma, l \rightarrow r\}} s$ (or just $t \rightsquigarrow_\sigma s$) if p is a non-variable position in t and $\sigma(t) \xrightarrow{p}_{l \rightarrow r} s$. A term t is a head-normal form (or root-stable) if it cannot be reduced to a redex. Similarly, t is a head-normal form in the context of narrowing if it cannot be narrowed to a redex. We say that a rewriting (or narrowing) relation \rightarrow is root-normalizing if all infinite \rightarrow -sequences eventually reach a root-stable form.

Two (possibly renamed) rules $l \rightarrow r$ and $l' \rightarrow r'$ *overlap* if there is a non-variable position $p \in \text{Pos}(l)$ and a most-general unifier σ such that $\sigma(l|_p) = \sigma(l')$ (where $l' \rightarrow r'$ is not a variant of $l \rightarrow r$ in case of $p = \Lambda$). The pair $\langle \sigma(l)[\sigma(r')]_p, \sigma(r) \rangle$ is called a *critical pair*. A critical pair $\langle \sigma(l)[\sigma(r')]_p, \sigma(r) \rangle$ with $p = \Lambda$ is called an *overlay*. A critical pair $\langle s, t \rangle$ such that $s = t$ is called *trivial*. A left-linear TRS without critical pairs is called *orthogonal*. A left-linear TRS whose critical pairs are trivial overlays is called *almost orthogonal*.

3. NATURAL REWRITING

Some rewrite strategies proposed to date are called demand driven and can be classified as call-by-need. Loosely speaking, *demandness* is understood as follows: if possible, a term t is evaluated at the top; otherwise, some arguments of the root of t are (recursively) evaluated if they might promote the application of a rewrite rule at the top, i.e. if a rule “demands” the evaluation of these arguments. In [8], Antoy and Lucas show that modern functional (logic) languages include evaluation strategies which consider elaborated definitions of ‘demandness’ which are, in fact, related [3, 6, 7, 12, 19, 20, 22].

In [1], we provided a notion of demandness which gives us a useful algorithm to calculate the rules and “demanded” positions necessary in a rewriting sequence. In order to define natural rewriting in terms of narrowing in later sections, we extend this algorithm to also consider variables as disagreeing positions.

DEFINITION 1 (DISAGREEING POSITIONS). [1] *Given terms $t, l \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, we let $\text{Pos}_{\neq}(t, l) = \text{minimal}_{\leq}(\{p \in \text{Pos}(t) \cap \text{Pos}_{\mathcal{F}}(l) \mid \text{root}(l|_p) \neq \text{root}(t|_p)\})$.*

DEFINITION 2 (DEMANDED POSITIONS). [1] *We define the set of demanded positions of term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ w.r.t. $l \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ (a *lhs* of a rule defining $\text{root}(t)$), i.e. the set of*

(positions of) maximal disagreeing subterms as:

$$DP_l(t) = \begin{cases} \text{Pos}_{\neq}(t, l) & \text{if } \text{Pos}_{\neq}(t, l) \cap \text{Pos}_c(t) = \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

and the set of demanded positions of t w.r.t. TRS \mathcal{R} as $DP_{\mathcal{R}}(t) = \cup\{DP_l(t) \mid l \rightarrow r \in \mathcal{R} \wedge \text{root}(t) = \text{root}(l)\}$.

EXAMPLE 4. Consider again Example 2 and terms t_3 and t_4 . We have $DP_{\mathcal{R}}(t_3) = DP_{\mathcal{R}}(t_4) = \{1, 2, 3, 4\}$ since (identically for t_4):

$$\begin{aligned} DP_{\text{solve}(X, s(Y), s(Z), W, s(H))}(t_3) &= \{2, 3\} \\ DP_{\text{solve}(X, 0, s(Z), W, s(H))}(t_3) &= \{2, 3\} \\ DP_{\text{solve}(s(X), Y, 0, W, s(H))}(t_3) &= \{1, 3\} \\ DP_{\text{solve}(0, s(Y), 0, W, s(H))}(t_3) &= \{1, 2, 3\} \\ DP_{\text{solve}(0, 0, 0, 0, H)}(t_3) &= \{1, 2, 3, 4\} \\ DP_{\text{solve}(s(X), Y, 0, W, 0)}(t_3) &= \emptyset \\ DP_{\text{solve}(X, 0, s(Z), W, 0)}(t_3) &= \emptyset \\ DP_{\text{solve}(0, s(Y), Z, 0, 0)}(t_3) &= \emptyset \\ DP_{\text{solve}(X, s(Y), s(Z), s(W), 0)}(t_3) &= \emptyset \end{aligned}$$

Note that the restriction of disagreeing positions to positions with non-constructor symbols (defined symbols in the case of rewriting) disables the evaluation of subterms which could never help to produce a redex (see [1, 3, 22]). Outermost-needed rewriting also considers this constructor-based condition though some refinement is still possible, as shown in Example 3.

Antoy and Lucas introduced the idea that some rewriting strategies select ‘popular’ demanded positions over the available set [8]. Here, we formalize a notion of popularity of demanded positions which makes the difference by counting the number of times a position is demanded by some rule and, then we select the most (frequently) demanded positions. This information, i.e. the number of times a position is demanded, is crucial and is managed by using multisets instead of sets for representing demanded positions.

We redefine the set of demanded positions of t w.r.t. TRS \mathcal{R} , $DP_{\mathcal{R}}(t)$, as follows.

DEFINITION 3 (DEMANDED POSITIONS). We define the multiset of demanded positions of a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ w.r.t. TRS \mathcal{R} as $DP_{\mathcal{R}}(t) = \oplus\{DP_l(t) \mid l \rightarrow r \in \mathcal{R} \wedge \text{root}(t) = \text{root}(l)\}$ where $M_1 \oplus M_2$ is the union of multisets M_1 and M_2 .

EXAMPLE 5. Continuing Example 4, we have $DP_{\mathcal{R}}(t_3) = DP_{\mathcal{R}}(t_4) = \{4, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 3\}$.

Now, one can simply think in a rewriting strategy which collects all demanded positions within a term w.r.t. the TRS and selects those which are “the most often demanded”. For instance, we would select position 3 for terms t_3 and t_4 in Example 5. This idea works well when using inductively sequential CSs but does not work correctly with non-inductively sequential CSs, as the following example shows.

EXAMPLE 6. Consider the well-known parallel-or TRS \mathcal{R} [5, 21]:

$$\begin{aligned} \text{True} \vee X &= \text{True} \\ X \vee \text{True} &= \text{True} \\ \text{False} \vee \text{False} &= \text{False} \end{aligned}$$

Consider the term $t = (\text{True} \vee \text{False}) \vee (\text{True} \vee \text{False})$. Positions 1 and 2 of t would be selected as reducible by natural rewriting since $DP_{\mathcal{R}}(t) = \{1, 2, 1, 2\}$. However, if we consider the following slightly different TRS \mathcal{R}' :

$$\begin{aligned} \text{True} \vee X &= \text{True} \\ X \vee \text{True} &= \text{True} \\ \text{False} \vee X &= X \end{aligned}$$

Only position 1 would be reducible by natural rewriting since it is the most often demanded position, i.e. $DP_{\mathcal{R}'}(t) = \{1, 2, 1\}$.

The solution is to consider the set of most often demanded positions which covers all the rules involved in the term evaluation.

DEFINITION 4 (DEMANDED POSITIONS). We define the set of demanded positions of a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ w.r.t. TRS \mathcal{R} as

$$DP_{\mathcal{R}}^m(t) = \{p \in DP_{\mathcal{R}}(t) \mid \exists l \in L(\mathcal{R}). p \in DP_l(t) \text{ and } \forall q \in DP_{\mathcal{R}}(t) : p <_{DP_{\mathcal{R}}(t)} q \Rightarrow l|_q \in \mathcal{X}\}$$

where $x <_M y$ denotes that the number of occurrences of x in the multiset M is less than the number of occurrences of y .

Note that the set of most often demanded positions is always included in $DP_{\mathcal{R}}^m(t)$; in symbols $\text{maximal}_{<_{DP_{\mathcal{R}}(t)}}(DP_{\mathcal{R}}(t)) \subseteq DP_{\mathcal{R}}^m(t)$.

EXAMPLE 7. Consider Example 6 again. Now we have $DP_{\mathcal{R}'}^m(t) = \{1, 2\}$ since position 1 alone does not cover the rule $X \vee \text{True} = \text{True}$.

The following definition establishes the strategy used in natural rewriting for selecting demanded positions.

DEFINITION 5 (NATURAL REWRITING). Given a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and a TRS \mathcal{R} , $\mathfrak{m}(t)$ is defined as the smallest set satisfying

$$\mathfrak{m}(t) \ni \begin{cases} (\Lambda, l \rightarrow r) & \text{if } l \rightarrow r \in \mathcal{R} \text{ and } l \leq t \\ (p.q, l \rightarrow r) & \text{if } p \in DP_{\mathcal{R}}^m(t) \text{ and } (q, l \rightarrow r) \in \mathfrak{m}(t|_p) \end{cases}$$

We consider \mathfrak{m} as simply a function returning positions if the rules selected for reduction are not relevant or directly deducible from the context. Roughly speaking, given a term, natural rewriting always reduces, in a non-deterministic way, the term itself or the most often demanded positions within the term covering all eventually applicable rules. We say term t reduces by natural rewriting to term s , denoted by $t \xrightarrow{\mathfrak{m}}_{\{p, l \rightarrow r\}} s$ (or simply $t \xrightarrow{\mathfrak{m}} s$) if $(p, l \rightarrow r) \in \mathfrak{m}(t)$.

EXAMPLE 8. Continuing Example 5. It is easy to see that $\mathfrak{m}(t_3) = \{3\}$ since positions 1, 2, and 4 are also demanded, but the redex at position 3 is the most often demanded (which also covers all eventually applicable rules). Then, the only possible natural rewriting step is:

$$\begin{aligned} \text{solve}(10!, 0+s(0), 0+s(0), W, s(0)) \\ \xrightarrow{\mathfrak{m}} \text{solve}(10!, 0+s(0), s(0), W, s(0)) \end{aligned}$$

EXAMPLE 9. Again, consider the TRS \mathcal{R}' and the term t in Example 6. Example 7 showed that $\mathfrak{m}(t) = \{1, 2\}$ and, thus, there exist two possible natural rewriting steps:

$$\begin{aligned} (\underline{\text{True} \vee \text{False}}) \vee (\text{True} \vee \text{False}) \\ \xrightarrow{\mathfrak{m}} \text{True} \vee (\text{True} \vee \text{False}) \\ (\text{True} \vee \text{False}) \vee (\underline{\text{True} \vee \text{False}}) \\ \xrightarrow{\mathfrak{m}} (\text{True} \vee \text{False}) \vee \text{True} \end{aligned}$$

From now on, we investigate the properties of natural rewriting.

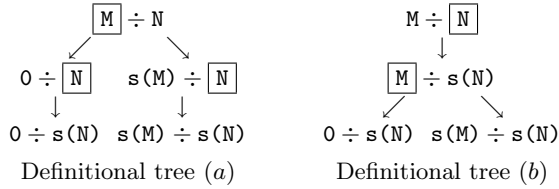


Figure 1: The two possible definitional trees for the symbol \div .

3.1 Neededness

First, we recall the definition of a definitional tree and an inductively sequential CS. A definitional tree of a defined symbol f is a finite, non-empty set \mathcal{T} of linear patterns partially ordered by \leq and having the following properties (up to renaming of variables) [4]:

leaf property The maximal elements of \mathcal{T} , referred to as the *leaves*, are variants of the left-hand sides of the rules defining f . Non-maximal elements are referred to as *branches*.

root property The minimum element of \mathcal{T} is referred to as the *root*⁴.

parent property If π is a pattern of \mathcal{T} different from the root, there exists in \mathcal{T} a unique pattern π' strictly preceding π such that there exists no other pattern strictly between π and π' . π' is referred to as the *parent* of π and π as a *child* of π' .

induction property All the children of a same parent differ from each other only at the position of a variable of their parent, referred to as *inductive position*.

A defined symbol f is called *inductively sequential* if there exists a definitional tree \mathcal{T} with pattern $f(x_1, \dots, x_k)$ (where x_1, \dots, x_k are different variables) whose leaves contain all and only the rules defining f . In this case, we say that \mathcal{T} is a definitional tree for f , denoted as \mathcal{T}_f . A left-linear CS⁵ \mathcal{R} is *inductively sequential* if all its defined function symbols are inductively sequential. An inductively sequential CS can be viewed as a set of definitional trees, each defining a function symbol. It is often convenient and simplifies understanding to provide a graphical representation of definitional trees as a tree of patterns where the inductive position in branch nodes is surrounded by a box [4].

EXAMPLE 10. *The symbol \div in Example 3 is inductively sequential since there exists a definitional tree for pattern $M \div N$. Figure 1 shows the two possible definitional trees.*

When a left-linear CS is not inductively sequential, a rule partition which obtains inductively sequential subsets is applied.

⁴This definition of root property differs from that of [4]; in [4]: “the minimum element, referred to as the root, of \mathcal{T} is $f(x_1, \dots, x_k)$, where x_1, \dots, x_k , are fresh, distinct variables.” This provides us with a more flexible definition which fits our interests.

⁵Left-linear CSs is the largest class where (weakly) outermost-needed rewriting and narrowing can be defined [5].

EXAMPLE 11. *The symbol `solve` in Example 2 is non-inductively sequential, since a rule partition which splits its rules into subsets which are inductively sequential is applied. Figure 2 shows the definitional tree of each part of the partition.*

Now, we extend the notion of an inductively sequential symbol to terms. This is the key idea of the paper for speaking about (optimal) sequential evaluation. We denote the maximal set of rules which can eventually be matched to a term t (possibly after some evaluation steps) by $match_t(\mathcal{R}) = \{l \in L(\mathcal{R}) \mid DP_l(t) \neq \emptyset \text{ or } l \leq t\}$.

DEFINITION 6 (INDUCTIVELY SEQUENTIAL TERM). *Let $\mathcal{R} = (\mathcal{F}, R)$ be a left-linear CS and $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. We say t is inductively sequential if for all $p \in \mathcal{Pos}_{\mathcal{D}}(t)$, there exists a definitional tree \mathcal{T}_p with root pattern π such that $\pi \leq t|_p$ and leaves of \mathcal{T}_p are all and only instances of $match_{t|_p}(\mathcal{R})$.*

Note that inductive sequentiality of a term is preserved by instantiation since, given a term t with variables and a substitution σ , $match_{\sigma(t)}(\mathcal{R}) \subseteq match_t(\mathcal{R})$.

EXAMPLE 12. *Consider Example 2. Terms t_3 and t_4 are inductively sequential since*

$$\begin{aligned}
 match_{t_3}(\mathcal{R}) = match_{t_4}(\mathcal{R}) = & \{solve(X, s(Y), s(Z), W, s(H)), \\
 & solve(X, 0, s(Z), W, s(H)), solve(s(X), Y, 0, W, s(H)), \\
 & solve(0, s(Y), 0, W, s(H)), solve(0, 0, 0, 0, H)\}
 \end{aligned}$$

and the definitional tree associated to both terms is depicted in Figure 3.

EXAMPLE 13. *On the other hand, consider Example 2 again but term $t' = solve(10!, 0+1, 0+1, 0+1, 0+1)$. This term is not inductively sequential since all lhs's for the symbol `solve` could be (potentially) matched and a rule partition is necessary, as shown in Figure 2. Then, optimal evaluation is not ensured. For instance, natural rewriting returns positions 3 and 5 as the most popular candidates for reduction, i.e. $m(t') = \{3, 5\}$ where*

$$\begin{aligned}
 DP_{solve(X, s(Y), s(Z), W, s(H))}(t') &= \{2, 3, 5\} \\
 DP_{solve(X, 0, s(Z), W, s(H))}(t') &= \{2, 3, 5\} \\
 DP_{solve(s(X), Y, 0, W, s(H))}(t') &= \{1, 3, 5\} \\
 DP_{solve(0, s(Y), 0, W, s(H))}(t') &= \{1, 2, 3, 5\} \\
 DP_{solve(0, 0, 0, 0, H)}(t') &= \{1, 2, 3, 4\} \\
 DP_{solve(s(X), Y, 0, W, 0)}(t') &= \{1, 3, 5\} \\
 DP_{solve(X, 0, s(Z), W, 0)}(t') &= \{2, 3, 5\} \\
 DP_{solve(0, s(Y), Z, 0, 0)}(t') &= \{1, 2, 4, 5\} \\
 DP_{solve(X, s(Y), s(Z), s(W), 0)}(t') &= \{2, 3, 4, 5\}
 \end{aligned}$$

The (computable) notion of an inductively sequential term is based on the idea of a definitional tree whose root pattern is not necessarily⁶ of the form $f(x_1, \dots, x_k)$ where x_1, \dots, x_k are variables. The following optimality results are consequences of Definition 6 above. We first introduce the notion of neededness of a rewriting step.

In [21], *root-neededness* of a redex is introduced as the basis to develop *normalizing* and *infinitary normalizing* rewrite strategies for lazy (or call-by-need) rewriting. In [16], it is proved that (almost) orthogonal strongly sequential TRSs

⁶This is why we made the definition of a definitional tree more flexible.

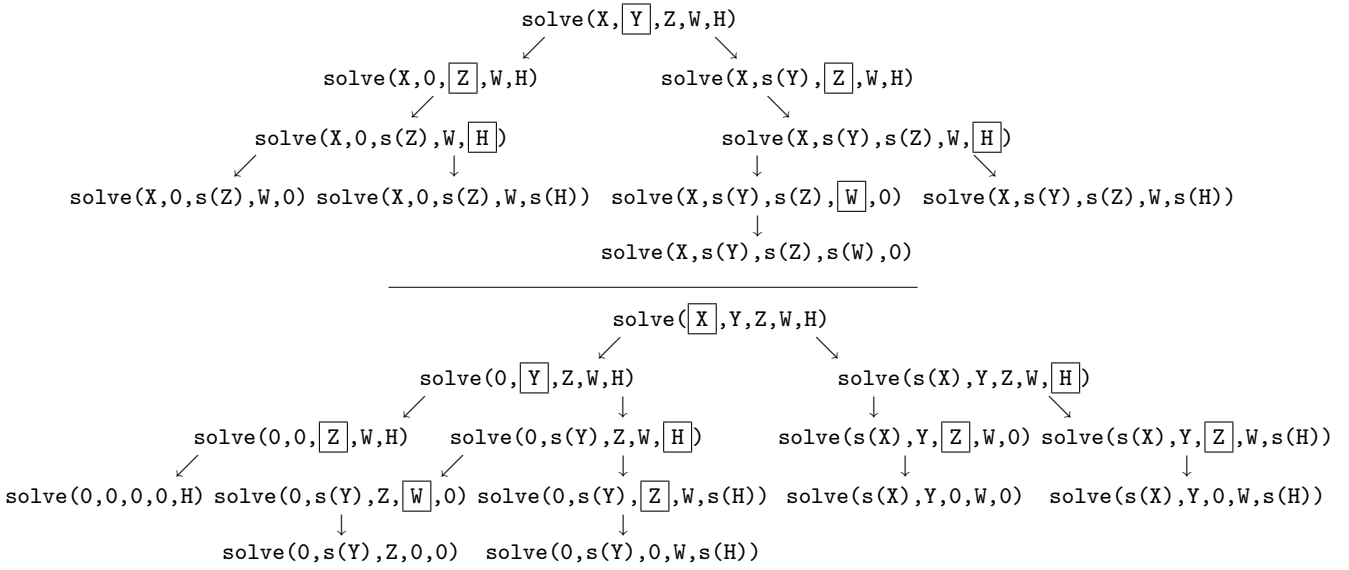


Figure 2: A partition of definitional trees for the symbol solve.

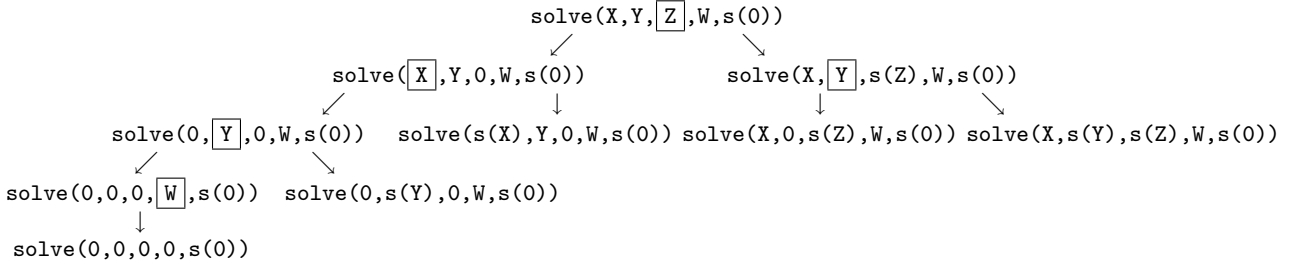


Figure 3: Definitional tree for the terms $\text{solve}(10!, 0+1, 0+1, W, 1)$ and $\text{solve}(X, Y, 0+2, W, 1)$.

amount to reducing root-needed redexes. Here, we investigate a related problem, namely whether natural rewriting (induced by left-linear CSSs) performs only ‘root-needed’ rewriting steps when considering inductively sequential terms. First, some preliminary notions about root-neededness are given.

DEFINITION 7 (DESCENDANTS OF A POSITION). [17] Let $A : t \xrightarrow{p} l \rightarrow r$ s be a rewriting step and $q \in \text{Pos}(t)$. The set $q \setminus A$ of descendants of q in s is defined as follows:

$$q \setminus A = \begin{cases} \{q\} & \text{if } q < p \text{ or } q \parallel p \\ \{p.p_3.p_2 \mid r|_{p_3} = l|_{p_1}\} & \text{if } q = p.p_1.p_2, p_1 \in \text{Pos}_X(l) \\ \emptyset & \text{otherwise} \end{cases}$$

If $Q \subseteq \text{Pos}(t)$ then $Q \setminus A$ denotes the set $\bigcup_{q \in Q} q \setminus A$. The notion of descendant extends to rewrite sequences in the obvious way. If Q is a set of pairwise disjoint positions in t and $A : t \rightarrow^* s$ then the positions in $Q \setminus A$ are pairwise disjoint.

DEFINITION 8 (ROOT-NEEDEDNESS). [21] A position p of a term t is called root-needed if in every reduction sequence from t to a head-normal form, either $t|_p$ or a descendant of $t|_p$ is reduced. A rewriting step $t \xrightarrow{p} s$ is root-needed if the reduced position p is root-needed.

It is worth noting that the notion of neededness of a rewriting step is only meaningful for orthogonal TRSs, since all

redexes of a term correspond to non-overlapping rules and their descendants cannot be destroyed. However, this notion is well-defined when we consider inductively sequential terms because all redexes of an inductively sequential term correspond to non-overlapping rules.

PROPOSITION 1. Let \mathcal{R} be a left-linear CS. If $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is an inductively sequential term, then for all $p \in \text{Pos}_{\mathcal{D}}(t)$, left-hand sides in $\text{match}_{t|_p}(\mathcal{R})$ are non-overlapping.

The following theorem shows that natural rewriting always performs root-needed rewriting steps when inductively sequential terms are considered.

THEOREM 1 (OPTIMALITY). Let $\mathcal{R} = (\mathcal{F}, R)$ be a left-linear CS and $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ be an inductively sequential term which is not root-stable. Then each $t \xrightarrow{m}_{\{p, l \rightarrow r\}} s$ is root-needed.

Note that since we do not restrict ourselves to orthogonal TRSs, later reduction steps in any derivation starting from the natural rewriting step may not hold the property of root-neededness. However, the idea behind this result is that even if a later step is not root-needed, the natural rewriting step (from the inductively sequential term) is still unavoidable to obtain a root-stable form, as shown by the following example.

EXAMPLE 14. Consider the following non-orthogonal TRS \mathcal{R} :

$$\begin{array}{ll} \mathbf{f} \rightarrow \mathbf{g}(\mathbf{h}, \mathbf{h}) & \mathbf{g}(\mathbf{x}, 0) \rightarrow \mathbf{g}(0, 0) \\ \mathbf{h} \rightarrow 0 & \mathbf{g}(0, \mathbf{x}) \rightarrow 0 \end{array}$$

Term \mathbf{f} is inductively sequential since there is only one applicable rule. Natural rewriting performs the step $\mathbf{f} \xrightarrow{\mathbf{m}} \mathbf{g}(\mathbf{h}, \mathbf{h})$. This natural rewriting step is root-needed since it is the unique reduction on term \mathbf{f} which leads to the root-stable form 0. Here, the idea behind Theorem 1 shows up since positions 1 and 2 of $\mathbf{g}(\mathbf{h}, \mathbf{h})$ are not root-needed because each one of them can be avoided depending on the chosen rule for \mathbf{g} . Indeed, the term $\mathbf{g}(\mathbf{h}, \mathbf{h})$ is not inductively sequential because it cannot be given a definitional tree without a partition into inductively sequential subsets of rules.

The condition \mathcal{R} must be a constructor system is necessary, as shown by the following example.

EXAMPLE 15. Let \mathcal{R} be the following non-CS:

$$\mathbf{f}(\mathbf{g}) \rightarrow \mathbf{f}(\mathbf{g}) \quad \mathbf{g} \rightarrow 0$$

Let $t = \mathbf{f}(\mathbf{g})$ be an inductively sequential term. We have that $\mathbf{m}(t) = \{\Lambda\}$, but position Λ is not root-needed since $\mathbf{f}(\mathbf{g}) \rightarrow \mathbf{f}(0)$ is the unique root-normalizing sequence.

On the other hand, left-linearity is also necessary.

EXAMPLE 16. Consider the CS:

$$\mathbf{f}(\mathbf{x}, \mathbf{x}) \rightarrow \mathbf{f}(\mathbf{x}, \mathbf{x}) \quad \mathbf{g} \rightarrow \mathbf{h} \quad \mathbf{h} \rightarrow \mathbf{a} \quad \mathbf{h} \rightarrow \mathbf{b}$$

The term $t = \mathbf{f}(\mathbf{g}, \mathbf{g})$ is inductively sequential and natural rewriting strategy yields $\mathbf{m}(t) = \{\Lambda\}$. However, position Λ is not root-needed since the following sequence, which does not reduce Λ , is root-normalizing:

$$\mathbf{f}(\mathbf{g}, \mathbf{g}) \rightarrow \mathbf{f}(\mathbf{h}, \mathbf{g}) \rightarrow \mathbf{f}(\mathbf{a}, \mathbf{g}) \rightarrow \mathbf{f}(\mathbf{a}, \mathbf{b})$$

Note that term $\mathbf{f}(\mathbf{g}, \mathbf{g})$ is inductively sequential while term $\mathbf{f}(\mathbf{h}, \mathbf{g})$ is not since there exist overlapping rules for the symbol \mathbf{h} .

We are also concerned with correctness and completeness of natural rewriting. We are able to prove that normal forms of $\xrightarrow{\mathbf{m}}$ are root-stable forms.

THEOREM 2 (CORRECTNESS). Let $\mathcal{R} = (\mathcal{F}, R)$ be a left-linear TRS. If $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is a $\xrightarrow{\mathbf{m}}$ -normal form, then s is root-stable.

The following example shows that left-linearity condition is necessary.

EXAMPLE 17. Consider the following TRS:

$$\mathbf{f}(\mathbf{x}, \mathbf{x}) \rightarrow 0 \quad \mathbf{g} \rightarrow 0$$

For the term $t = \mathbf{f}(\mathbf{g}, 0)$, we obtain that $\mathbf{m}(t) = \emptyset$ since t is not a redex and there is no demanded position. However, t is not root-stable since there exists the following rewrite sequence: $\mathbf{f}(\mathbf{g}, 0) \rightarrow \mathbf{f}(0, 0) \rightarrow 0$.

And we can prove completeness of natural rewriting w.r.t. root-stable forms.

THEOREM 3 (COMPLETENESS). Let $\mathcal{R} = (\mathcal{F}, R)$ be a left-linear CS and $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. If $t \rightarrow^* s$ and s is root-stable, then $\exists s' \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ s.t. $t \xrightarrow{\mathbf{m}}^* s'$, $\text{root}(s') = \text{root}(s)$ and $s' \xrightarrow{\Delta}^* s$.

Here, Examples 15 and 17 can be used to show the necessity of CS and left-linearity conditions, respectively.

Besides the properties of natural rewriting w.r.t. inductively sequential terms, in the following section, we formalize a class of TRSs where natural rewriting preserves optimality for sequential parts of the program.

3.2 Preserving Sequentiality

We provide a class of TRSs called *inductively sequential preserving* with some interesting normalization properties. This class of programs is based on Definition 6 above.

DEFINITION 9 (INDUCTIVELY SEQUENTIAL PRESERVING). We say a left-linear CS \mathcal{R} is inductively sequential preserving if for all rule $l \rightarrow r \in \mathcal{R}$, right-hand side r is an inductively sequential term.

EXAMPLE 18. The program of Example 1 is trivially inductively sequential preserving. Consider now the program of Example 2. This CS is also inductively sequential preserving since rhs $\text{solve}(\mathbf{X}, 0, \mathbf{Z}, \mathbf{W}, \mathbf{H})$ induces sequential evaluation in position 3, rhs $\text{solve}(\mathbf{X}, \mathbf{Y}, 0, \mathbf{W}, \mathbf{H})$ induces sequential evaluation in position 1, and rhs $\text{solve}(\mathbf{X}, \mathbf{Y}, \mathbf{s}(\mathbf{Z}), \mathbf{W}, \mathbf{H})$ induces sequential evaluation in position 2 or 5. On the other hand, consider the CS of Example 16. This CS is clearly non-inductively sequential preserving since rhs \mathbf{h} is a non-inductively sequential term because there exist overlapping rules for defined the symbol \mathbf{h} .

This class of TRSs is larger than the class of inductively sequential CSs.

COROLLARY 1. Inductively sequential left-linear CSs are trivially inductively sequential preserving left-linear CSs.

The following result asserts that natural rewriting steps in an inductively sequential preserving CS preserve inductive sequentiality of terms.

THEOREM 4. Let $\mathcal{R} = (\mathcal{F}, R)$ be an inductively sequential preserving left-linear CS and $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ be an inductively sequential term. If $t \xrightarrow{\mathbf{m}}_{\{p, l \rightarrow r\}} s$, then s is inductively sequential.

Specifically, this last theorem provides the following interesting property about natural rewriting.

THEOREM 5 (ROOT-NORMALIZATION). Reduction of inductively sequential terms within the class of inductively sequential preserving left-linear CSs is root-normalizing.

EXAMPLE 19. Consider Example 2 again. Since Example 18 shows CS of Example 2 is inductively sequential preserving and Example 12 proves term t_3 is inductively sequential, rewriting term t_3 through natural rewriting is optimal and provides appropriate root-stable forms.

Now, we clarify the precise relation between outermost-needed rewriting and natural rewriting. Roughly speaking, natural rewriting is conservative w.r.t. outermost-needed rewriting for inductively sequential left-linear CSs.

3.3 (Weakly) Outermost-needed rewriting

The formal definition of outermost-needed rewriting for an inductively sequential TRS (i.e. through a definitional tree) can be found in [4] whereas the definition for non-inductively sequential TRSs can be found in [6]. The reader should note that definitional trees (as well as the necessary partition) are commonly associated to the TRS at a first stage and remain fixed during execution time.

In strongly sequential TRSs, the order of evaluation is determined by matching automata [17]. In inductively sequential left-linear CSs, the order of evaluation is determined by

a mapping φ which implements the strategy by traversing definitional trees as a finite state automata in order to compute the demanded positions to be reduced [4, 5, 6, 7].

DEFINITION 10 (OUTERMOST-NEEDED REWRITING). [4] *Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ with $\text{root}(t) = f$ and its definitional tree \mathcal{T}_f , $\varphi(t, \mathcal{T}_f)$ be defined as follows:*

$$\varphi(t, \mathcal{T}_f) = \begin{cases} (\Lambda, l \rightarrow r) & \text{if } \pi \in \mathcal{T}_f \text{ is a leaf, } l \rightarrow r \in \mathcal{R}, \\ & \pi \text{ is a variant of } l, \text{ and } \pi \leq t. \\ (p.q, l \rightarrow r) & \text{if } \pi \in \mathcal{T}_f \text{ is a branch with inductive} \\ & \text{position } p, \text{ root}(t|_p) = f' \in \mathcal{D}, \\ & \pi \leq t, \text{ and } \varphi(t|_p, \mathcal{T}_{f'}) = (q, l \rightarrow r) \end{cases}$$

We say term t reduces by *outermost-needed rewriting* to term s , denoted by $t \xrightarrow{\text{m}}_{\{p, l \rightarrow r\}} s$ if $\varphi(t, \mathcal{T}_{\text{root}(t)}) = (p, l \rightarrow r)$.

For non-inductively sequential left-linear CSs, the rules defining a non-inductively defined the symbol f must be partitioned into inductively sequential subsets, i.e. subsets for which there exists a definitional tree (see [4]). In such case, φ is applied to all inductively sequential subsets and the disjoint outermost positions from all computed positions are selected.

EXAMPLE 20. *Again, consider Example 2 and the definitional trees for the partition of the symbol `solve` in Figure 2. It is clear that $\varphi(t_3, \mathcal{T}_{\text{solve}}) = \{1, 2\}$ and that subterm `10!` is rewritten. This is because `solve` has two definitional trees, positions 1 and 2 are the inductive positions of the root branch pattern of each definitional tree, and term t_3 has defined symbols at these positions.*

The following theorem establishes that natural rewriting and outermost-needed rewriting coincide for inductively sequential CSs.

THEOREM 6. *Let \mathcal{R} be an inductively sequential left-linear CS and $t, s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that t is not root-stable. Then, $t \xrightarrow{\text{m}}_{\{p, l \rightarrow r\}} s$ if and only if $t \xrightarrow{\text{m}}_{\{p, l \rightarrow r\}} s$.*

Note that the condition of non root-stability of t is necessary since natural rewriting performs a more exhaustive matching test.

EXAMPLE 21. *Continuing Example 3, according to Definition 4, there is no demanded position for the term $t = 10! \div 0$, i.e. $DP_{\mathcal{R}}^{\text{m}}(t) = \emptyset$. However, if definitional tree (a) of Figure 1 is used by outermost-needed rewriting, it cannot detect that t is a head-normal form and forces the reduction of the first argument since it blindly follows the selected definitional tree.*

It is worth noting that if definitional tree (b) of Figure 1 is selected for use, this concrete behavior does not happen. However, the reader should note that the problem persists for other terms, i.e. it cannot be solved by simply selecting an alternative definitional tree. For instance, if we re-consider the set of constructor symbols in the program as $\mathcal{C} = \{0, \text{s}, \text{pred}\}$ (instead of simply $\{0, \text{s}\}$), subterm `10!` in the term $t_1 = 10! \div \text{pred}(0)$ is (uselessly) reduced when using definitional tree (a), whereas subterm `10!` in the term $t_2 = \text{pred}(0) \div 10!$ is (uselessly) reduced when using definitional tree (b). Note that both terms t_1 and t_2 are detected as head-normal forms by natural rewriting.

In the following, we consider the complementary problem of whether natural rewriting is also optimal for non-inductively sequential terms.

3.4 Non-sequentiality

We are able to provide some results regarding reduction on non-inductively sequential terms. First, Definition 8 of neededness of a rewriting step is not applicable when non-inductively sequential CSs are considered (see [24, 21]) and the notion of a necessary set of redexes is used instead.

DEFINITION 11 (ROOT-NECESSARY SET OF REDEXES). [21] *A set of (pairwise disjoint) redexes in a term t is called root-necessary if in every rewrite sequence from t to a head-normal form a descendant of at least one of the redexes in the set is contracted.*

Natural rewriting always computes a root-necessary set of redexes.

THEOREM 7. *Let \mathcal{R} be a left-linear CS and $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. Then, $\text{m}(t)$ is a root-necessary set of redexes.*

Note that for almost orthogonal TRSs, repeated contraction of root-necessary sets of redexes results in a root-stable form, whenever the latter exists (see [21]). On the other hand, natural rewriting is more conservative than weakly outermost-needed rewriting, as shown by the following theorem.

THEOREM 8. *Let \mathcal{R} be a left-linear CS and $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. Then, $\text{m}(t) \subseteq \varphi(t, \mathcal{T}_{\text{root}(t)})$.*

From now on, we extend natural rewriting to narrowing with similar achievements.

4. NATURAL NARROWING

Narrowing is an extension of rewriting for term rewriting systems where pattern matching is substituted by unification (as in *logic programming*) [18, 25]. Narrowing is the basic computational model of (multiparadigm) functional logic languages [13, 14, 15, 23]. The narrowing mechanism is complete in the sense of logic programming (computation of *answers*) as well as in the sense of functional programming (computation of *values*). Functional logic programming inherits advantages of the different functional and logic paradigms in a unique and seamless way: functional programming provides nested expressions, efficient evaluation by deterministic (often lazy) evaluation and higher-order functions; whereas logic programming provides existentially quantified variables, partial data structures and built-in search. See [13] for a survey.

The extension of natural rewriting to narrowing is not difficult and stems from the idea that narrowing differs from rewriting only in the instantiation step prior to rewriting.

DEFINITION 12 (NATURAL NARROWING). *Given a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and a TRS \mathcal{R} , $\text{mn}(t)$ is defined as the smallest set satisfying*

$$\text{mn}(t) \ni \begin{cases} (\Lambda, id, l \rightarrow r) & \text{if } l \rightarrow r \in \mathcal{R} \text{ and } l \leq t \\ (p.q, \theta, l \rightarrow r) & \text{if } DP_{\mathcal{R}}^{\text{m}} \cap \text{Pos}_{\mathcal{X}}(t) = \emptyset, p \in DP_{\mathcal{R}}^{\text{m}}, \\ & \text{and } (q, \theta, l \rightarrow r) \in \text{mn}(t|_p) \\ (p, \theta \circ \sigma, l \rightarrow r) & \text{if } p \in DP_{\mathcal{R}}^{\text{m}}, t|_p = x \in \mathcal{X}, c \in \mathcal{C}, \\ & \sigma(x) = c(\bar{w}), \bar{w} \text{ are fresh variables,} \\ & \text{and } (p, \theta, l \rightarrow r) \in \text{mn}(\sigma(t)) \end{cases}$$

Intuitively, instantiations are performed only at those variables which are part of the most demanded positions, whereas

reductions are only performed when no variable at the most demanded positions is available. We say term t narrows by natural narrowing to term s at position p using substitution σ , denoted by $t \xrightarrow{m}_{\{p, l \rightarrow r, \sigma\}} s$ (or simply $t \xrightarrow{m}_{\sigma} s$), if $(p, \sigma, l \rightarrow r) \in \text{mn}(t)$.

EXAMPLE 22. Consider TRS \mathcal{R} and term t_4 of Example 2. Natural narrowing performs only the following narrowing sequences:

$$\begin{aligned} & \text{solve}(X, Y, 0 + s(s(0)), W, s(0)) \\ & \xrightarrow{m}_{id} \underline{\text{solve}(X, Y, s(s(0)), W, s(0))} \\ & \xrightarrow{m}_{\{Y \mapsto 0\}} \underline{\text{solve}(X, 0, s(0), W, 0)} \\ & \quad | \xrightarrow{m}_{\{Y \mapsto s(Y')\}} \underline{\text{solve}(X, Y', s(s(0)), W, 0)} \\ & \xrightarrow{m}_{id} \text{False} \quad | \xrightarrow{m}_{\{Y' \mapsto 0\}} \text{False} \\ & \quad | \xrightarrow{m}_{\{Y' \mapsto s(Y''), W \mapsto 0, X \mapsto 0\}} \text{False} \\ & \quad | \xrightarrow{m}_{\{Y' \mapsto s(Y''), W \mapsto s(W')\}} \text{False} \end{aligned}$$

Note that these narrowing sequences coincide with the ‘optimal’ narrowing sequences associated to t_4 in Example 2.

4.1 Neededness and sequentiality

In [7], it is shown that neededness of a narrowing step does not stem in a simple way from its counterpart of rewriting since some unifiers may not be strictly unavoidable to achieve a normal form. Intuitively, a narrowing step is needed no matter which unifiers might be used later in the derivation.

DEFINITION 13 (OUTERMOST-NEEDEDNESS). [7] A narrowing step $t \xrightarrow{p}_{\{l \rightarrow r, \sigma\}} s$ is called *outermost-needed* (or *needed*) if for every substitution $\eta \geq \sigma$, p is the position of a root-needed or outermost-needed redex of $\eta(t)$, respectively.

In the following, we prove that optimality w.r.t. inductively sequential terms, similar to Theorem 1, can be established for natural narrowing.

THEOREM 9 (OPTIMALITY). Let $\mathcal{R} = (\mathcal{F}, R)$ be a left-linear CS and $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ be an inductively sequential term which is not root-stable. Then each natural narrowing step $t \xrightarrow{m}_{\{p, \sigma, l \rightarrow r\}} s$ is outermost-needed.

Following the explanations given for Theorem 1, the notion of neededness of a narrowing step is only meaningful for orthogonal TRSs though it is well-defined when we consider inductively sequential terms (see Proposition 1). Moreover, the idea behind the previous theorem is again similar to that of natural rewriting (see Example 14).

EXAMPLE 23. Consider Example 22. All the steps in the narrowing sequences from t_4 are outermost-needed since they are unavoidable when attempting to achieve the normal form **False**. Specifically, the initial rewrite step is necessary to achieve the normal form and the instantiation on Y is unavoidable if a normal form is desired.

Moreover, root-stability of Theorem 2 is extensible to natural narrowing.

THEOREM 10 (CORRECTNESS). Let $\mathcal{R} = (\mathcal{F}, R)$ be a left-linear CS, If $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is a \xrightarrow{m} -normal form, then s is root-stable.

Note that the CS condition is necessary here (in contrast to Theorem 2), as shown by the following example.

EXAMPLE 24. Consider the following non-constructor system where $\mathcal{C} = \{0\}$ and $\mathcal{D} = \{\mathbf{f}, \mathbf{g}\}$:

$$\mathbf{f}(\mathbf{g}) \rightarrow 0$$

Natural narrowing cannot narrow the term $\mathbf{f}(\mathbf{x})$ to the root-stable form 0 since it is not able to build the substitution $\{\mathbf{x} \mapsto \mathbf{g}\}$.

We can also prove completeness of natural narrowing w.r.t. root-stable forms.

THEOREM 11 (COMPLETENESS). Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a left-linear CS and $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. If $t \xrightarrow{*} s$ and s is root-stable, then $\exists s' \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ s.t. $t \xrightarrow{m,*} s'$, $\text{root}(s') = \text{root}(s)$ and $s' \xrightarrow{\Lambda,*} s$.

Similarly, we provide the extension of root-normalization within the class of inductively sequential preserving CSs of Theorem 4 and 5.

THEOREM 12. Let \mathcal{R} be an inductively sequential preserving left-linear CS and $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ be an inductively sequential term. If $t \xrightarrow{m}_{\{p, \sigma, l \rightarrow r\}} s$, then s is inductively sequential.

THEOREM 13 (ROOT-NORMALIZATION). Narrowing of inductively sequential terms within the class of inductively sequential preserving left-linear CSs is root-normalizing.

Now, we clarify the precise relation between outermost-needed narrowing and natural narrowing.

4.2 (Weakly) Outermost-needed narrowing

The formal definition of outermost-needed narrowing for an inductively sequential CS (i.e. through a definitional tree) can be found in [7] whereas the definition for non-inductively sequential TRSs can be found in [6]. Mapping φ in Section 3.3 is extended to mapping λ to be able to manage substitutions.

DEFINITION 14 (OUTERMOST-NEEDED NARROWING). [7] Given a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ with $\text{root}(t) = f$ and its definitional tree \mathcal{T}_f , $\lambda(t, \mathcal{T}_f)$ is defined as the smallest set satisfying:

$$\lambda(t, \mathcal{T}_f) \ni \left\{ \begin{array}{l} (\Lambda, \sigma, l \rightarrow r) \text{ if } \pi \in \mathcal{T}_f \text{ is a leaf, } l \rightarrow r \in \mathcal{R}, \\ \quad \pi \text{ is a variant of } l, \text{ and} \\ \quad \exists \sigma : \sigma = \text{mgu}(t, l) \\ (p.q, \theta \circ \sigma, l \rightarrow r) \text{ if } \pi \in \mathcal{T}_f \text{ is a branch with} \\ \quad \text{inductive position } p, \\ \quad \text{root}(t|_p) = f' \in \mathcal{D}, \\ \quad \exists \sigma : \sigma = \text{mgu}(t, \pi), \\ \quad \text{and } (q, \theta, l \rightarrow r) \in \lambda(\sigma(t)|_p, \mathcal{T}_{f'}) \end{array} \right.$$

We say term t narrows by *outermost-needed narrowing* to term s (at position p and using substitution σ), denoted by $t \xrightarrow{m}_{\{p, l \rightarrow r, \sigma\}} s$ if $(p, \sigma, l \rightarrow r) \in \lambda(t, \mathcal{T}_{\text{root}(t)})$.

For non-inductively sequential CSs, λ is applied to all inductively sequential subsets and the union of all computed triples is selected.

EXAMPLE 25. Consider the TRS and term t_4 of Example 2. Also consider the definitional trees for the partition of the symbol **solve** in Figure 2. Outermost-needed narrowing yields:

$$\lambda(\text{solve}(X, Y, 0 + 2, W, 1), \mathcal{T}_{\text{solve}}) = \left\{ \begin{array}{l} (3, \{Y \mapsto 0\}), \\ (3, \{Y \mapsto s(Y')\}), \\ (3, \{X \mapsto 0, Y \mapsto 0\}), \\ (3, \{X \mapsto 0, Y \mapsto s(Y')\}), \\ (3, \{X \mapsto s(X')\}) \end{array} \right\}$$

since `solve` has two definitional trees, positions 1 and 2 are the inductive positions of the root branch pattern of each definitional tree, and subsequent matching or instantiation is made until a branch whose inductive position is 3 is reached.

The following theorem establishes that natural narrowing is conservative w.r.t. outermost-needed narrowing for inductively sequential CSs.

THEOREM 14. *Let \mathcal{R} be an inductively sequential left-linear CS and $t, s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that t is not root-stable. Then, $t \xrightarrow{n}_{\{p, \sigma, l \rightarrow r\}} s$ if and only if $t \xrightarrow{m}_{\{p, \sigma, l \rightarrow r\}} s$.*

Again, condition of non root-stability is necessary (as in Theorem 6) since natural narrowing performs a more exhaustive unification process (see Examples 3 and 21).

4.3 Non-sequentiality

And finally, we are able to provide some results regarding narrowing of non-inductively sequential terms. Here, in contrast to Section 3.4, there is no a common notion of need-edges of a narrowing step when non-inductively sequential CSs are considered. Thus, we only prove that natural narrowing is more conservative than weakly outermost-needed narrowing.

THEOREM 15. *Let \mathcal{R} be a left-linear CS and $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. Then, $\text{mn}(t) \subseteq \lambda(t, \mathcal{T}_{\text{root}(t)})$.*

5. CONCLUSIONS

We have provided a suitable refinement of the demandness notion associated to *outermost-needed rewriting* which integrates as well as improves the demandness notion of the on-demand evaluation presented in [1]. On behalf of this new demandness notion, we have introduced an improvement of *outermost-needed rewriting* called *natural rewriting*, and we have extended it to narrowing.

We have shown that natural rewriting is a conservative extension of outermost-needed rewriting (see Theorem 6). Similarly, we have shown that natural narrowing is a conservative extension of outermost-needed narrowing (see Theorem 14). Moreover, natural rewriting behaves even better than outermost-needed rewriting for the class of inductively sequential left-linear CSs in the avoidance of failing computations (see Theorem 6 and Examples 3 and 21). Also in the case of narrowing (see Theorem 14 and Examples 3 and 21). On the other hand, correctness and completeness of natural rewriting and narrowing are guaranteed (see Theorems 2, 3, 10, and 11).

Moreover, we defined a new class of TRSs called *inductively sequential preserving* where natural rewriting and narrowing preserve optimality for sequential parts of a program. This new class of TRSs is based on the extension of the notion of inductive sequentiality from defined function symbols to terms (see Theorems 1 and 9). Moreover, an interesting result is obtained w.r.t. this new class of programs: reduction (narrowing) of inductively sequential terms within the class of inductively sequential preserving left-linear CSs is root-normalizing (see Theorems 5 and 13). Note that the formalization extends in general to left-linear CSs (see Theorems 8 and 15).

Finally, we have developed a prototype implementation of natural rewriting and natural narrowing (called `Natur`) which is publicly available at

<http://www.dsic.upv.es/users/elp/soft.html>

This prototype has been used to test the examples presented in this paper. The interpreter is implemented in Haskell (using `ghc 5.04.1`) and accepts OBJ programs, which have a syntax that is similar to the syntax of modern functional (logic) programs, e.g. the functional syntax used in Haskell or Curry. Note that the prototype does not build any definitional tree and directly codifies Definitions 5 and 12.

As future work, we plan to develop better implementation techniques, which would be similar to the incremental finite state automata approach of [24] or the incremental outermost-needed narrowing of [2]. On the other hand, in this paper, we have not addressed complexity of natural rewriting and narrowing (or the prototype implementation). We have left this for future work.

5.1 Related work

This work is framed into the formal basis established by [17] and [24] for optimal sequential reduction. It is worth noting that the idea of most often demanded positions can be recovered in a general way from the normalization algorithm *FindNS* of [24], though the idea was hinted at [8].

On the one hand, note that a computable and optimal rewrite strategy based on tree automata exists for left-linear right-ground TRSs [11]. This technique is able to manage Example 1 but cannot perform sequential evaluation in Example 2. On the other hand, there exist normalizing, sequential rewrite strategies for almost orthogonal TRSs [9]. However, efficiency (w.r.t. the number of reduction steps) is not their main objective and optimal evaluation is not ensured. For instance, when rewriting the following term $t = \text{solve}(10!, 0, 0+10, 0, 1)$ with the TRS of Example 2, the sequential strategy of [9] would select the subterm `10!` for reduction instead of subterm `0+10` since both subterms have the same maximal depth and leftmost redexes are preferred.

Finally, a simple but informal clarifying idea arises as a conclusion of this work: natural rewriting and narrowing dynamically select the rules involved in the evaluation of a term in order to ensure optimality. Indeed, natural rewriting and narrowing refine outermost-needed rewriting and narrowing, respectively, as strategies which simulate the dynamical computation of the definitional tree associated to each term found in an evaluation sequence. However, the reader should remember that no definitional tree is used in the formal definition of natural rewriting and narrowing, and this behavior is obtained using only the simple but natural and powerful idea of “most often demanded positions”.

Acknowledgements. I would like to thank María Alpuente and Salvador Lucas for many helpful remarks on preliminary versions of this work and for carefully reading this paper. I am also grateful to the anonymous referees for their detailed comments to improve the final version of this paper.

6. REFERENCES

- [1] M. Alpuente, S. Escobar, B. Gramlich, and S. Lucas. Improving on-demand strategy annotations. In M. Baaz and A. Voronkov, editors, *Proc. 9th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'02)*, volume 2514 of *Lecture Notes in Computer Science*, pages 1–18, Tbilisi, Georgia, 2002. Springer-Verlag, Berlin.

- [2] M. Alpuente, S. Escobar, and S. Lucas. UPV-Curry: an Incremental Curry Interpreter. In J. Pavelka, G. Tel, and M. Bartosek, editors, *Proc. of 26th Seminar on Current Trends in Theory and Practice of Informatics, SOFSEM'99*, volume 1725 of *Lecture Notes in Computer Science*, pages 327–335. Springer-Verlag, Berlin, 1999.
- [3] M. Alpuente, M. Falaschi, P. Julián, and G. Vidal. Specialization of Lazy Functional Logic Programs. In *Proc. of the ACM SIGPLAN Conf. on Partial Evaluation and Semantics-Based Program Manipulation, PEPM'97*, volume 32, number 12 of *ACM Sigplan Notices*, pages 151–162. ACM Press, New York, 1997.
- [4] S. Antoy. Definitional trees. In *Proc. of the 3rd International Conference on Algebraic and Logic Programming ALP'92*, volume 632 of *Lecture Notes in Computer Science*, pages 143–157. Springer-Verlag, Berlin, 1992.
- [5] S. Antoy. Constructor-based conditional narrowing. In *Proc. of the 3rd International Conference on Principles and Practice of Declarative Programming (PPDP'01)*, pages 199–206, Florence, Italy, Sept. 2001. ACM.
- [6] S. Antoy, R. Echahed, and M. Hanus. Parallel evaluation strategies for functional logic languages. In *Proc. of the Fourteenth International Conference on Logic Programming (ICLP'97)*, pages 138–152. MIT Press, 1997.
- [7] S. Antoy, R. Echahed, and M. Hanus. A needed narrowing strategy. In *Journal of the ACM*, volume 47(4), pages 776–822, 2000.
- [8] S. Antoy and S. Lucas. Demandness in rewriting and narrowing. In M. Comini and M. Falaschi, editors, *Proc. of the 11th Int'l Workshop on Functional and (Constraint) Logic Programming WFLP'02*, volume 76 of *Electronic Notes in Theoretical Computer Science*. Elsevier Sciences Publisher, 2002.
- [9] S. Antoy and A. Middeldorp. A Sequential Reduction Strategy. *Theoretical Computer Science*, 165:75–95, 1996.
- [10] T. Arts and J. Giesl. A collection of examples for termination of term rewriting using dependency pairs. Technical report, AIB-2001-09, RWTH Aachen, Germany, 2001.
- [11] I. Durand and A. Middeldorp. Decidable Call by Need Computations in Term Rewriting. In W. McCune, editor, *Proc. of CADE'97*, volume 1249 of *Lecture Notes in Artificial Intelligence*, pages 4–18. Springer-Verlag, Berlin, 1997.
- [12] W. Fokkink, J. Kamperman, and P. Walters. Lazy rewriting on eager machinery. *ACM Transactions on Programming Languages and Systems*, 22(1):45–86, 2000.
- [13] M. Hanus. The integration of functions into logic programming: From theory to practice. *Journal of Logic Programming*, 19&20:583–628, 1994.
- [14] M. Hanus, S. Antoy, H. Kuchen, F. López-Fraguas, W. Lux, J. Moreno Navarro, and F. Steiner. Curry: An Integrated Functional Logic Language (version 0.8). Available at: <http://www.informatik.uni-kiel.de/~curry>, 2003.
- [15] M. Hanus, H. Kuchen, and J. Moreno-Navarro. Curry: A Truly Functional Logic Language. In *Proc. ILPS'95 Workshop on Visions for the Future of Logic Programming*, pages 95–107, 1995.
- [16] M. Hanus, S. Lucas, and A. Middeldorp. Strongly sequential and inductively sequential term rewriting systems. *Information Processing Letters*, 67(1):1–8, 1998.
- [17] G. Huet and J.-J. Lévy. Computations in Orthogonal Term Rewriting Systems, Part I + II. In *Computational logic: Essays in honour of J. Alan Robinson*, pages 395–414 and 415–443. The MIT Press, Cambridge, MA, 1992.
- [18] J. Hullot. Canonical Forms and Unification. In *Proc of 5th Int'l Conf. on Automated Deduction*, volume 87 of *Lecture Notes in Computer Science*, pages 318–334. Springer-Verlag, Berlin, 1980.
- [19] S. Lucas. Termination of on-demand rewriting and termination of obj programs. In *Proc. of 3rd International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, PPDP'01*, pages 82–93. ACM Press, New York, 2001.
- [20] S. Lucas. Context-sensitive rewriting strategies. *Information and Computation*, 178(1):294–343, 2002.
- [21] A. Middeldorp. Call by need computations to root-stable form. In *Proceedings of the 24th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 94–105. ACM Press, New York, 1997.
- [22] J. Moreno-Navarro and M. Rodríguez-Artalejo. Logic Programming with Functions and Predicates: The language Babel. *Journal of Logic Programming*, 12(3):191–224, 1992.
- [23] U. Reddy. Narrowing as the Operational Semantics of Functional Languages. In *Proc. of Second IEEE Int'l Symp. on Logic Programming*, pages 138–151. IEEE, New York, 1985.
- [24] R. Sekar and I. Ramakrishnan. Programming in equational logic: Beyond strong sequentiality. *Information and Computation*, 104(1):78–109, 1993.
- [25] J. Slagle. Automated theorem-proving for theories with simplifiers, commutativity, and associativity. *Journal of the ACM*, 21(4):622–642, 1974.
- [26] TeReSe, editor. *Term Rewriting Systems*. Cambridge University Press, Cambridge, 2003.