

Natural Rewriting for General Term Rewriting Systems

Santiago Escobar¹, José Meseguer², and Prasanna Thati³

¹ Universidad Politécnica de Valencia, Spain. sescobar@dsic.upv.es

² University of Illinois at Urbana-Champaign, USA. meseguer@cs.uiuc.edu

³ Carnegie Mellon University, USA. thati@cs.cmu.edu

Abstract. We address the problem of an efficient rewriting strategy for general term rewriting systems. Several strategies have been proposed over the last two decades for rewriting, the most efficient of all being the natural rewriting strategy [9]. All the strategies so far, including natural rewriting, assume that the given term rewriting system is a left-linear constructor system. Although these restrictions are reasonable for some functional programming languages, they limit the expressive power of equational languages, and they preclude certain applications of rewriting to equational theorem proving and to languages combining equational and logic programming. In this paper, we propose a conservative generalization of natural rewriting that does not require the rules to be left-linear and constructor-based. We also establish the soundness and completeness of this generalization.

1 Introduction

A challenging problem in modern programming languages is the discovery of sound and complete evaluation strategies which are: (i) optimal w.r.t. some efficiency criterion (typically the number of rewrite steps), (ii) easily implementable, and (iii) applicable for a large class of programs. In this paper, we focus on (iii).

The evaluation strategies for programming languages so far can be classified into two classical families: *eager strategies* (also known as *innermost* or *call-by-value*) and *lazy strategies* (also known as *outermost* or *call-by-need*). The choice of the strategy can have a big impact on performance and semantics of programming languages: for instance, lazy rewriting typically needs more resources than eager rewriting [22], but the former improves termination of the program w.r.t. the latter. To define a lazy rewriting strategy, we have to define what a *needed computation* is and also have to provide an efficient procedure to determine whether some computation is needed. These two problems were first addressed for rewriting in a seminal paper by Huet and Levy [18], where the *strongly needed reduction* strategy was proposed. Several refinements of this strategy have been proposed over the last two decades, the most significant ones being Sekar and Ramakrishnan's *parallel needed reduction* [24], and Antoy, Echahed and Hanus' *(weakly) outermost-needed rewriting* [1,3,4]. Recently, (weakly)

outermost-needed rewriting has been improved by Escobar by means of the *natural rewriting strategy* [9,10]. Natural rewriting is based on a suitable extension of the demandedness notion associated to (weakly) outermost-needed rewriting. Moreover, the strategy enjoys good computational properties such as soundness and completeness w.r.t. head-normal forms, and it preserves optimality w.r.t. the number of reduction steps for sequential parts of the program.

A typical assumption of previous strategies [14,19,16,24,1,3,4,2,9,10] is that the rewrite rules are *left-linear* and *constructor*. These restrictions are reasonable for some functional programming languages, but they limit the expressive power of equational languages such as OBJ [15], CafeOBJ [13], ASF+SDF [8], and Maude [7], where non-linear left-hand sides are perfectly acceptable. This extra generality is also necessary for applications of rewriting to equational theorem proving, and to languages combining equational and logic programming, since in both cases assuming left-linearity is too restrictive. Furthermore, for rewrite systems whose semantics is not equational but is instead rewriting logic based, such as rewrite rules in ELAN [6], or Maude system modules, the constructor assumption is unreasonable and almost never holds.

In summary, generalizing the natural rewriting strategy to *general* rewriting systems, without left-linearity and constructor conditions, will extend the scope of applicability of the strategy to more expressive equational languages and to rewriting logic based languages, and will open up a much wider range of applications. In the following, we give the reader a first intuitive example of how that generalization will work.

Example 1. Consider the following TRS for proving equality (\approx) of arithmetic expressions built using division (\div), modulus or remainder ($\%$), and subtraction ($-$) operations on natural numbers.

$$\begin{array}{ll}
(1) \ 0 \div s(N) \rightarrow 0 & (5) \ M - 0 \rightarrow M \\
(2) \ s(M) \div s(N) \rightarrow s((M-N) \div s(N)) & (6) \ s(M) - s(N) \rightarrow M-N \\
(3) \ M \% s(N) \rightarrow (M-s(N)) \% s(N) & (7) \ X \approx X \rightarrow \text{True} \\
(4) \ (0 - s(M)) \% s(N) \rightarrow N - M
\end{array}$$

Note that this TRS is not left-linear because of rule (7) and it is not constructor because of rule (4). Therefore, it is outside the scope of all the strategies mentioned above. Furthermore, note that the TRS is neither terminating nor confluent due to rule (3).

Consider the term⁴ $t_1 = 10! \div 0$. If we only had rules (1), (2), (5) and (6), the natural rewriting strategy [9] would be applicable and no reductions on t_1 would be performed, since t_1 is a head-normal form. In contrast, the other strategies mentioned above, for example, outermost-needed rewriting, would force⁵ the

⁴ The subterm $10!$ represents factorial of $s^{10}(0)$ but we do not include the rules for $!$ because we are only interested in the fact that it has a remarkable computational cost, and therefore we would like to avoid its reduction in the examples whenever possible.

⁵ Note that this behavior is independent of the fact that two possible definitional trees, a data structure guiding the outermost-needed rewriting strategy [1], exist

evaluation of the computationally expensive subterm $10!$. Hence, we would like to generalize natural rewriting to a version that enjoys this optimality (w.r.t. the number of rewrite steps) and that can also handle non-left-linear and non-constructor rules such as (7) and (4).

Consider the term $t_2 = 10! \% (\mathbf{s}(0) - \mathbf{s}(0)) \approx 10! \% 0$. We would like the generalization of the natural rewriting strategy to perform only the optimal computation:

$$\begin{aligned} 10! \% (\mathbf{s}(0) - \mathbf{s}(0)) &\approx 10! \% 0 \\ \rightarrow 10! \% (0 - 0) &\approx 10! \% 0 \rightarrow \underline{10! \% 0} \approx 10! \% 0 \rightarrow \mathbf{True} \end{aligned}$$

that avoids unnecessary reduction of the subterm $10! \% 0$ at the final rewrite step, and also avoids reductions on the computationally expensive term $10!$ during the whole rewrite sequence.

Since natural rewriting [9] uses a more refined demandedness notion for re-dexes in comparison with other strategies such as outermost needed rewriting [1,4], it leads to a very efficient lazy evaluation strategy. In this paper, we propose a conservative generalization of this demandedness notion that drops the assumptions that the rewrite rules are left-linear and constructor, while retaining soundness and completeness w.r.t. head-normal forms.

It is worthy to mention that an exception in the previous strategies about the left-linearity requirement is [2]. In [2], a non-left-linear rule “ $l \rightarrow r$ ” is transformed into a left-linear conditional rule “ $l' \rightarrow r$ if $\dots, X \downarrow X_1, \dots, X \downarrow X_n, \dots$ ” by renaming, in the linear term l' , extra occurrences of a variable X to X_1, \dots, X_n . However, $t \downarrow s$ succeeds only if there exists a constructor term w such that $t \rightarrow^* w$ and $s \rightarrow^* w$ and this is an unreasonable condition for the kind of rewrite systems with a non-equational semantics, like rewriting logic, considered in this paper. Anyway, the strategy of [2] is not applicable to Example 1, since imposes the constructor condition.

The reader might wonder if a suitable program transformation that converts a term rewriting system into a left-linear constructor system whose semantics is equivalent to the original is possible. However, existing techniques for linearization of term rewriting systems, such as [17], or for transformation of a term rewriting system into a constructor system, such as [23], are not applicable in general to term rewriting systems; for instance, these two techniques do not apply to Example 1, since they require the TRS to be terminating, confluent and forward-branching (see [23] for further details).

for symbol \div ; see [9, Example 21]. The idea is that for any of the two definitional trees available, there will exist terms for which the previous problem still persists. Note that this problem becomes unavoidable, since a definitional tree is fixed for a program, not for a source term. Specifically, if we consider that the set of constructor symbols is $\{0, \mathbf{s}, \mathbf{pred}\}$ (instead of simply $\{0, \mathbf{s}\}$), then the subterm $10!$ in the term $s_1 = 10! \div \mathbf{pred}(0)$ is uselessly reduced for one of the definitional trees, whereas subterm $10!$ in the term $s_2 = \mathbf{pred}(0) \div 10!$ is uselessly reduced for the other definitional tree. However, both terms are detected as head-normal forms by natural rewriting [9], like the previous term $10! \div 0$.

After some preliminaries in Section 2, we present our generalization of natural rewriting strategy in Section 3, and formally define its properties. We show soundness and completeness of the generalized rewrite strategy w.r.t. head-normal forms. In Section 4, we further refine the strategy to obtain a more optimal one, without losing the soundness and completeness properties. Finally, we conclude in Section 5. Missing proofs can be found in [11].

2 Preliminaries

We assume a finite alphabet (function symbols) $\mathcal{F} = \{\mathbf{f}, \mathbf{g}, \dots\}$, and a countable set of variables $\mathcal{X} = \{\mathbf{x}, \mathbf{y}, \dots\}$. We denote the set of terms built from \mathcal{F} and \mathcal{X} by $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and write $\mathcal{T}(\mathcal{F})$ for ground terms built only from \mathcal{F} . We write $\mathcal{V}ar(t)$ for the set of variables occurring in t . A term is said to be linear if it has no multiple occurrences of a single variable. We use finite sequences of integers to denote a position in a term. Given a set $S \subseteq \mathcal{F} \cup \mathcal{X}$, $\mathcal{P}os_S(t)$ denotes positions in t where symbols or variables in S occur. We write $\mathcal{P}os_{\mathcal{F}}(t)$ and $\mathcal{P}os(t)$ as a shorthand for $\mathcal{P}os_{\{\mathcal{F}\}}(t)$ and $\mathcal{P}os_{\mathcal{F} \cup \mathcal{X}}(t)$, respectively. We denote the *root position* by Λ . Given positions p, q , we denote its concatenation as $p.q$ and define $p/q = p'$ if $p = q.p'$. Positions are ordered by the standard prefix ordering \leq . We say p and q are *disjoint positions* and write $p \parallel q$, if $p \not\leq q$ and $q \not\leq p$. For sets of positions P, Q we define $P.Q = \{p.q \mid p \in P \wedge q \in Q\}$. We write $P.q$ as a shorthand for $P.\{q\}$ and similarly for $p.Q$. The subterm of t at position p is denoted as $t|_p$, and $t[s]_p$ is the term t with the subterm at position p replaced by s . We define $t|_P = \{t|_p \mid p \in P\}$. The symbol labeling the root of t is denoted as $root(t)$. Given a set of positions P , we call $p \in P$ an *outermost position* in P if there is no $q \in P$ such that $q < p$.

A *substitution* is a function $\sigma : \mathcal{X} \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{X})$ which maps variables to terms, and which is different from the identity only for a finite subset $\mathcal{D}om(\sigma)$ of \mathcal{X} . We denote the homomorphic extension of σ to $\mathcal{T}(\mathcal{F}, \mathcal{X})$ also by σ , and its application to a term t by $\sigma(t)$. The set of variables introduced by σ is $\mathcal{R}an(\sigma) = \cup_{x \in \mathcal{D}om(\sigma)} \mathcal{V}ar(\sigma(x))$. 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. $\sigma(s) = t$. We write $\sigma^{-1}(x) = \{y \in \mathcal{D}om(\sigma) \mid \sigma(y) = x\}$.

A rewrite rule is an ordered pair (l, r) of terms, also written $l \rightarrow r$, with $l \notin \mathcal{X}$. 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 assume that \mathcal{F} is defined 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 symbols*, where $\mathcal{D} = \{root(l) \mid l \rightarrow r \in R\}$ 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 every $l \in L(\mathcal{R})$ is a pattern.

A term t rewrites to s at position $p \in \mathcal{P}os(t)$ using the rule $l \rightarrow r$, called a *rewrite step* and written $t \rightarrow_{\langle p, l \rightarrow r \rangle} s$ ($t \xrightarrow{p} s$ or simply $t \rightarrow s$), if $t|_p = \sigma(l)$ and $s = t[\sigma(r)]_p$. The pair $\langle p, l \rightarrow r \rangle$ is called a *redex* and we also refer to the

subterm $t|_p$ in a rewrite step from t at position p as a redex. We often underline the redex in a rewrite step for readability. We denote the reflexive and transitive closure of the rewrite relation \rightarrow by \rightarrow^* . We call t *the source* and s *the target* of a rewrite sequence $t \rightarrow^* s$. A term t is a \rightarrow -*normal form* (or normal form) if it contains no redex, i.e., there is no s such that $t \rightarrow s$. A term t is a \rightarrow -*head-normal form* (or head-normal form) if it cannot be reduced to a redex, i.e., there are no s, s' such that $t \rightarrow^* s \xrightarrow{\Lambda} s'$. We denote by $\xrightarrow{\Lambda}$ a rewrite step at a position $p > \Lambda$.

A (sequential) rewrite strategy \mathcal{S} for a TRS \mathcal{R} is a subrelation $\xrightarrow{\mathcal{S}} \subseteq \rightarrow$ [21]. A rewrite strategy is *head normalizing* [21] if it provides a head-normal form for every source term, if such head-normal form exists. In this paper, we are only interested in *head-normalizing* rewrite strategies, since they are the basis for lazy rewriting strategies, and the following correctness and completeness criteria:

1. (Correctness) If a term t is a $\xrightarrow{\mathcal{S}}_{\mathcal{R}}$ -normal form, then t is a head-normal form.
2. (Completeness) If $t \rightarrow^* s$, then $\exists s'$ s.t. $t \xrightarrow{\mathcal{S}}^* s'$, $root(s') = root(s)$ and $s' \xrightarrow{\Lambda}^* s$.

3 Generalizing Natural Rewriting

As mentioned earlier, we are interested in a lazy strategy that, to the extent possible, performs only those reductions that are essential for reaching head-normal forms. Now, if a term t is not a head-normal form; then we know that after a (possibly empty) sequence of rewrites at positions other than the root, a rule $l \rightarrow r$ can be applied at the root. Accordingly, we adopt the approach of computing a *demanded* set of redexes in t such that *at least* one of the redexes in the demanded set has to be reduced before *any* rule can be applied at the root position in t . This idea of demandedness for reductions at the root is common in lazy evaluation strategies for programming languages, such as outermost needed rewriting [4]; see [5] for a survey on demandedness in programming languages.

Definition 1. For a term s and a set of terms $T = \{t_1, \dots, t_n\}$ we say that s is a context of the terms in T if $s \leq t_i$ for all $1 \leq i \leq n$. There is always a least general context s of T , i.e., one such that for any other context s' we have $s' \leq s$; furthermore s is unique up to renaming of variables. For $1 \leq i \leq n$, let the substitution σ_i be such that $\sigma_i(s) = t_i$ and $Dom(\sigma_i) \subseteq Var(s)$. We define the set $\mathcal{Pos}_{\neq}(T)$ of disagreeing positions between the terms in T as those $p \in \mathcal{Pos}_{\mathcal{X}}(s)$ such that there is an i with $\sigma_i(s|_p) \neq s|_p$.

Example 2. Consider the set of terms $T = \{10! \% (s(0) - s(0)), 10! \% 0\}$ borrowed from Example 1. The least general context of T is the term $s = 10! \% Z$ and the set of disagreeing positions between terms in T is $\mathcal{Pos}_{\neq}(T) = \{2\}$.

Definition 2 (Demanded positions). For terms l and t , let s be the least general context of l and t , and let σ be the substitution such that $\sigma(s) = l$. We define the set of demanded positions in t w.r.t. l as

$$DP_l(t) = \bigcup_{x \in \mathcal{V}ar(s)} \begin{array}{l} \text{if } \sigma(x) \notin \mathcal{X} \text{ then } \mathcal{P}os_x(s) \text{ else } Q.\mathcal{P}os_{\neq}(t|_Q) \\ \text{where } Q = \mathcal{P}os_{\sigma^{-1}(\sigma(x))}(s) \end{array}$$

Let us unpack the definition above. Intuitively, the set $DP_l(t)$ returns a set of positions in t at which t necessarily has to be “changed” before applying the rule $l \rightarrow r$ at the root position, i.e., for l to be able to match the term under consideration. Suppose, s is the least general context of l and t , and σ is such that $\sigma(s) = l$. Note that for every non-variable position p in s , it is the case that t and l have the same symbol at p . Now, if σ maps a variable $x \in \mathcal{V}ar(s)$ to a non-variable term, then t and l disagree (have a different symbol) at every position $p \in \mathcal{P}os_x(s)$; this is a consequence of the fact that s is the least general context of l and t . The other case is where a variable $x \in \mathcal{V}ar(s)$ is mapped to a possibly non-linear variable of l . In this case, consider the positions of all the variables in s that are mapped to the same variable as x , namely $Q = \mathcal{P}os_{\sigma^{-1}(\sigma(x))}(s)$. Now, l matches t only if all the subterms of t at positions in Q are identical. Thus, we compute the disagreeing positions $\mathcal{P}os_{\neq}(t|_Q)$, and add $Q.\mathcal{P}os_{\neq}(t|_Q)$ to the set $DP_l(t)$. Finally, note that when $l \leq t$, it is the case that l is the least general context of l and t , and $DP_l(t) = \emptyset$.

Example 3. Consider the left-hand side $l_7 = \mathbf{X} \approx \mathbf{X}$ and the term $t_2 = 10! \% (\mathbf{s}(0) - \mathbf{s}(0)) \approx 10! \% 0$ of Example 1. The least general context of l_7 and t_2 is $s = \mathbf{W} \approx \mathbf{Y}$. Now, for $\sigma = \{\mathbf{W} \mapsto \mathbf{X}, \mathbf{Y} \mapsto \mathbf{X}\}$, we have $\sigma(s) = l_7$. While computing $DP_{l_7}(t_2)$, we obtain the set of disagreeing positions between the subterms in t_2 corresponding to the non-linear variable \mathbf{X} in l_7 , i.e., the set $\mathcal{P}os_{\neq}(10! \% (\mathbf{s}(0) - \mathbf{s}(0)), 10! \% 0) = \{2\}$. Thus, $DP_{l_7}(t_2) = \{1, 2\}.\{2\} = \{1.2, 2.2\}$.

Note that the symbol at a position $p \in DP_l(t)$ in t can be changed by not only a rewrite at p , but also by a rewrite at a position $q < p$. Thus, besides considering the positions in $DP_l(t)$ as candidates for rewrites, we also need to consider the positions q in t that are above some position in $DP_l(t)$. Thus, for a position q in a term t , we define $D_t^\uparrow(q) = \{p \mid p \leq q \wedge p \in \mathcal{P}os_{\mathcal{D}}(t)\}$. We lift this to sets of positions as $D_t^\uparrow(Q) = \cup_{q \in Q} D_t^\uparrow(q)$.

Example 4. Consider the TRS in Example 1, the left-hand side $l_7 = \mathbf{X} \approx \mathbf{X}$, and the new term $t = 0 \div \mathbf{s}(10!) \approx 0 \div \mathbf{s}(\mathbf{s}(10!))$. We have $DP_{l_7}(t) = \{1.2.1, 2.2.1\}$, and the subterms at positions 1.2.1 and 2.2.1 should be identical for the above rule to be applied at the root position. Now, reductions in only the subterm $10!$ at position 1.2.1 would never result in $\mathbf{s}(10!)$, which is the subterm at position 2.2.1, and vice versa. The right reduction sequence leading to constant **True** is the one reducing the symbols \div above the demanded positions 1.2.1 and 2.2.1:

$$\begin{aligned} \underline{0 \div s(10!)} &\approx 0 \div s(s(10!)) \\ &\rightarrow 0 \approx \underline{0 \div s(s(10!))} \rightarrow \underline{0 \approx 0} \rightarrow \text{True} \end{aligned}$$

We are now ready to compute the demanded set of redexes of a given term t .

Definition 3 (Demanded redexes). We define the sufficient set of positions of a term t as

$$SP(t) = \bigcup_{l \in L(\mathcal{R}) \wedge l \leq t} D_l^\uparrow(\text{Pos}_{\mathcal{X}}(l)) \cup D_t^\uparrow(FP(t))$$

where $FP(t) = \bigcup_{l \in L(\mathcal{R})} DP_l(t)$. Then, we define the demanded set of redexes of a term t as

$$DR(t) = \{\langle A, l \rightarrow r \rangle \mid l \in L(\mathcal{R}) \wedge l \leq t\} \cup \bigcup_{q \in SP(t) \setminus \{A\}} q.DR(t|_q)$$

where for a set of redexes S , we define $q.S = \{\langle q.p, l \rightarrow r \rangle \mid \langle p, l \rightarrow r \rangle \in S\}$.

The set $DR(t)$ is recursively computed as follows. Whenever $l \leq t$ for a rule $l \rightarrow r$, the redex $\langle A, l \rightarrow r \rangle$ is included in $DR(t)$. When $l \not\leq t$, we recursively compute $DR(t|_q)$ for each position $q \in D_t^\uparrow(DP_l(t))$. The case $l \leq t$ has an additional subtlety; specifically, in this case, we also have to recursively compute $DR(t|_q)$ for the positions q in t that have a defined symbol and that are above a variable position in l . This is necessary for the strategy to be complete, as illustrated by the following example.

Example 5. Consider the TRS

$$(i) \text{ first}(\text{pair}(X, Y)) \rightarrow X \quad (ii) \text{ pair}(X, Y) \rightarrow \text{pair}(Y, X)$$

and the term $t = \text{first}(\text{pair}(a, b))$. If we simply define $SP(t) = D_t^\uparrow(FP(t))$, then $DR(t) = \{\langle A, (i) \rangle\}$, and the only rewrite sequence starting from t and beginning with a redex in $DR(t)$ would be

$$\underline{\text{first}(\text{pair}(a, b))} \rightarrow a$$

But the term t can also be reduced to the head-normal form b as follows

$$\underline{\text{first}(\text{pair}(a, b))} \rightarrow \underline{\text{first}(\text{pair}(b, a))} \rightarrow b \quad (*)$$

Hence, although the left-hand side of rule (i) matches t , for the strategy to be complete, we also have to consider the subterm $\text{pair}(a, b)$ of t at position 1 (which is above variable positions 1.1 and 1.2 in the left-hand side of rule (i)), and recursively compute $DR(\text{pair}(a, b))$. Then we will have $DR(t) = \{\langle A, (i) \rangle, \langle 1, (ii) \rangle\}$, which enables us to account for the rewrite sequence (*) above.

From now on, while displaying the sets $DR(t)$ in examples, we will omit the rule $l \rightarrow r$ in a redex $\langle p, l \rightarrow r \rangle$ and simply write $\langle p \rangle$, whenever there is no scope for ambiguity about the rule.

Example 6. Consider again the term $t_2 = 10! \% (s(0) - s(0)) \approx 10! \% 0$ from Example 1, and the computation of $DR(t_2)$. Since t_2 is not a redex, we have that

$$DR(t_2) = \bigcup_{q \in SP(t_2) \setminus \{A\}} q.DR(t_2|_q).$$

But, since t_2 is not a redex, we have $SP(t_2) = D_{t_2}^\dagger(FP(t_2)) = D_{t_2}^\dagger(\cup_{l \in L(\mathcal{R})} DP_l(t_2))$. Now, from Example 3, we have that $DP_{l_7}(t_2) = \{1.2, 2.2\}$ for the rule $l_7 = \mathbf{X} \approx \mathbf{X}$ and $DP_l(t_2) = \{A\}$ for any other rule l in \mathcal{R} . So then, $D_{t_2}^\dagger(\cup_{l \in L(\mathcal{R})} DP_l(t_2)) = \{A, 1, 2, 1.2\}$, where the position 2.2 has been removed since it is not rooted by a defined symbol. Hence, we have

$$DR(t_2) = 1.DR(t_2|_1) \cup 2.DR(t_2|_2) \cup 1.2.DR(t_2|_{1.2}).$$

Now, we consider $DR(t_2|_{1.2})$. Subterm $\mathbf{s}(0) - \mathbf{s}(0)$ at position 1.2 is a redex and thus $\langle A \rangle \in DR(t_2|_{1.2})$. Furthermore, $SP(t_2|_{1.2}) \setminus \{A\} = \emptyset$, because all symbols under root position in $\mathbf{s}(0) - \mathbf{s}(0)$ are constructor symbols. Thus, we have $DR(t_2|_{1.2}) = \{ \langle A \rangle \}$.

Next, we consider $DR(t_2|_1)$. The subterm $10! \% (\mathbf{s}(0) - \mathbf{s}(0))$ is not a redex, and thus

$$DR(t_2|_1) = \cup_{q \in SP(t_2|_1) \setminus \{A\}} q.DR(t_2|_{1.q}).$$

Now consider $SP(t_2|_1)$. Since $10! \% (\mathbf{s}(0) - \mathbf{s}(0))$ is not a redex, we have $SP(t_2|_1) = D_{t_2|_1}^\dagger(FP(t_2|_1)) = D_{t_2|_1}^\dagger(\cup_{l \in L(\mathcal{R})} DP_l(t_2|_1))$. Consider $DP_{l_3}(t_2|_1)$ and $DP_{l_4}(t_2|_1)$ for left-hand sides $l_3 = \mathbf{M} \% \mathbf{s}(\mathbf{N})$ and $l_4 = (\mathbf{0} - \mathbf{s}(\mathbf{M})) \% \mathbf{s}(\mathbf{N})$; note that $DP_l(t_2|_1) = \{A\}$ for any other rule l in \mathcal{R} . Then, we have $DP_{l_3}(t_2|_1) = \{2\}$ and $DP_{l_4}(t_2|_1) = \{1, 2\}$ and therefore we have

$$DR(t_2|_1) = 1.DR(t_2|_{1.1}) \cup 2.DR(t_2|_{1.2}).$$

Now, this implies that we have to compute recursively $DR(t_2|_{1.1})$ and $DR(t_2|_{1.2})$. Now, $DR(t_2|_{1.2})$ was already computed before, and the reader can check that $DR(t_2|_{1.1}) = \{ \langle A \rangle \}$. So, we can conclude $DR(t_2|_1) = \{ \langle 1 \rangle, \langle 2 \rangle \}$.

Finally, consider $DR(t_2|_2)$. The subterm $10! \% 0$ is not a redex, thus

$$DR(t_2|_2) = \cup_{q \in SP(t_2|_2) \setminus \{A\}} q.DR(t_2|_{2.q}).$$

But using a similar reasoning that in the previous term $t_2|_1$, we can conclude $DR(t_2|_2) = \{ \langle 1 \rangle \}$. Finally, we have that $DR(t_2) = \{ \langle 1.1 \rangle, \langle 1.2 \rangle, \langle 2.1 \rangle \}$.

We are now ready to formally define the natural rewriting strategy.

Definition 4 (Natural rewriting). *We say that term t reduces by natural rewriting to term s , denoted by $t \xrightarrow{m}_{\langle p, l \rightarrow r \rangle} s$ (or simply $t \xrightarrow{m} s$) if $t \rightarrow_{\langle p, l \rightarrow r \rangle} s$ and $\langle p, l \rightarrow r \rangle \in DR(t)$.*

Example 7. Continuing Example 6, we have three possible natural rewriting steps from the term t_2 : (i) a rewriting step reducing the subterm $\mathbf{s}(0) - \mathbf{s}(0)$ at position 1.2, (ii) a rewriting step reducing the subterm $10!$ at position 1.1, and (iii) a rewriting step reducing the subterm $10!$ at position 2.1. The last two rewriting steps are undesirable and unnecessary for obtaining the normal form **True**, as shown in Example 1. Using the further refinements to the natural rewriting strategy presented in the next section, we will be able to avoid reducing these unnecessary redexes.

It is worthy to note that although some refinements are still necessary to obtain the efficient rewrite strategy we desire, we are already able to avoid some unnecessary rewrite steps while computing head-normal forms, as shown in the following example.

Example 8. Consider Example 1 and the term $t = 0 \div \mathbf{s}(10!)$. The term is a redex, so we have $DR(t) = \{ \langle \Lambda \rangle \} \cup \cup_{q \in SP(t) \setminus \{ \Lambda \}} q.DR(t|_q)$. Now we have $SP(t) = \{ \Lambda \} \cup D_t^\dagger(\cup_{l \in L(\mathcal{R})} DP_l(t))$. Now, $DP_{l_1}(t) = \emptyset$ for $l_1 = 0 \div \mathbf{s}(\mathbf{M})$, $DP_{l_2}(t) = \{ 1 \}$ for $l_2 = \mathbf{s}(\mathbf{M}) \div \mathbf{s}(\mathbf{N})$, and $DP_l(t) = \{ \Lambda \}$ for any other rule l in \mathcal{R} . Then, $SP(t) = \{ \Lambda \}$, since position 1 corresponds to a constructor, and therefore $DR(t) = \{ \langle \Lambda \rangle \}$. So, our natural rewriting strategy performs only the sequence: $0 \div \mathbf{s}(10!) \rightarrow 0$, and avoids any reduction on the computational expensive term $10!$.

In the remaining part of this section, we show that the natural rewriting strategy defined above satisfies the correctness and completeness criteria w.r.t. head-normal forms, that are described in Section 2.

The following is a property of $DR(t)$ that is easy to check, and that will be useful.

Remark 1. If $\langle q, l \rightarrow r \rangle \in DR(t)$ and $p < q.q'$ for $q' \in \mathcal{Pos}_{\mathcal{X}}(l)$, then $p.DR(t|_p) \subseteq DR(t)$.

In order to prove completeness of the generalized natural rewriting strategy, we introduce some auxiliary notation. Given two rewrite sequences $\pi = t \rightarrow^* s$ and $\pi' = s \rightarrow^* w$, we write $\pi; \pi'$ for the sequence $t \rightarrow^* s \rightarrow^* w$. Given a rewrite sequence $\pi = t_0 \xrightarrow{p_1} t_1; \pi'$ with $\pi' = t_1 \xrightarrow{p_2} t_2 \cdots \xrightarrow{p_n} t_n$ and given an outermost position p_k amongst p_1, \dots, p_n , we define the projection $\pi|_{p_k}$ as follows:

$$\pi|_{p_k} = \begin{cases} \pi & \text{if } n = 0 \\ \pi'|_{p_k} & \text{if } p_1 \parallel p_k \\ t_0|_{p_k} \xrightarrow{p_1/p_k} t_1|_{p_k}; \pi'|_{p_k} & \text{otherwise} \end{cases}$$

We now establish a key property of the set $DR(t)$ that will be useful in proving the correctness and completeness results.

Lemma 1. *Consider a rewrite sequence $t \rightarrow_{\langle p_1, l_1 \rightarrow r_1 \rangle} t_1 \cdots \rightarrow_{\langle p_n, l_n \rightarrow r_n \rangle} t_n$ such that $p_n = \Lambda$. Then, there is a k s.t. $1 \leq k \leq n$ and $\langle p_k, l_k \rightarrow r_k \rangle \in DR(t)$.*

Now, we prove correctness of our generalized rewrite strategy w.r.t. head-normal forms.

Theorem 1 (Correctness). *If a term t is a $\xrightarrow{\mathbf{m}}$ -normal form, then t is a head-normal form.*

Proof. We prove the contrapositive. Specifically, if t is not a head-normal form, then, by Lemma 1, we have $t \xrightarrow{\mathbf{m}}_{\langle p, l \rightarrow r \rangle} s$ for some s and t is not a $\xrightarrow{\mathbf{m}}$ -normal form. \square

In the following, we give some useful definitions and results that will be useful in proving completeness.

Lemma 2. *Consider a rewrite sequence $t \rightarrow_{\langle p_1, l_1 \rightarrow r_1 \rangle} t_1 \cdots \rightarrow_{\langle p_n, l_n \rightarrow r_n \rangle} t_n$ such that $\langle p_i, l_i \rightarrow r_i \rangle \notin DR(t)$ for all i s.t. $1 \leq i \leq n$. Then, there is no i and $\langle q, l \rightarrow r \rangle \in DR(t)$ such that $p_i < q.q'$ for $q' \in Pos_{\mathcal{X}}(l)$.*

To prove completeness, we will show that whenever there is a rewrite sequence $\pi = t \rightarrow^* t'$, then there is also a rewrite sequence $t \rightarrow_{\langle q, l \rightarrow r \rangle} s \rightarrow^* t'$ that begins with a redex $\langle q, l \rightarrow r \rangle \in DR(t)$. Furthermore, the rewrite sequence $\pi' = s \rightarrow^* t'$ is “smaller” in an appropriate sense in comparison to π . Specifically, we will define a well-founded metric on rewrite sequences, and show that the metric of π' is strictly smaller than that of π . The completeness result will then follow by noetherian induction on this metric.

Definition 5. *Given a rewrite sequence $\pi = t_0 \xrightarrow{p_1} t_1 \cdots \xrightarrow{p_n} t_n$, we define a metric $\mu(\pi)$ returning a sequence of natural numbers as follows.*

- Let k be the smallest integer such that $p_k = \Lambda$, if any. Then,

$$\mu(\pi) = \mu(\pi_1).1.\mu(\pi_2)$$

where $\pi_1 = t_0 \rightarrow^* t_{k-1}$ and $\pi_2 = t_k \rightarrow^* t_n$.

- If $p_i \neq \Lambda$ for all i , then let q_1, \dots, q_k be the outermost positions in p_1, \dots, p_n . We define

$$\mu(\pi) = \sum_{i=1}^k \mu(\pi|_{q_i})$$

where $+$ is inductively defined as: $n_1.v_1 + n_2.v_2 = (n_1 + n_2).(v_1 + v_2)$, $\epsilon + v = v$, and $v + \epsilon = v$.

We define the ordering $<$ on metrics as follows $v_1 < v_2$ if (i) $|v_1| < |v_2|$, or (ii) $|v_1| = |v_2|$, $v_1 = v'_1.n_1.v$, $v_2 = v'_2.n_2.v$, and $n_1 < n_2$; where $|\cdot|$ denotes the length of a sequence of natural numbers. Note that $<$ is a well-ordering.

The metric $\mu(\pi)$ essentially represents the parallelism that is implicit in the rewrite sequence π . Specifically, consider the rewrite sequence in Definition 5. If $p_k = \Lambda$, then the first $k - 1$ rewrites in π have to be performed before the k^{th} rewrite, and similarly the k^{th} rewrite has to be performed before any of the remaining $n - k$ rewrites. On the other hand, if p_i and p_j are two different outermost positions in p_1, \dots, p_n then all the rewrites in $\pi|_{p_i}$ and $\pi|_{p_j}$ can be performed parallelly. Thus, $|\mu(\pi)|$ is the number of sequential steps that would remain when π is parallelized to the extent possible, and further, if the i^{th} number in the sequence $\mu(\pi)$ is n_i then the i^{th} step in the parallelized version of π would contain n_i parallel reductions.

Example 9. Consider the TRS of Example 1 and the following sequence π :

$$s(\underline{0-0}) - 0) - s(0-0)$$

$$\rightarrow \underline{s(0-0)} - \underline{s(0-0)} \rightarrow \underline{s(0)} - \underline{s(0-0)} \rightarrow \underline{s(0)-s(0)} \rightarrow \underline{0-0} \rightarrow 0$$

The metric for this sequence is $\mu(\pi) = \mu(\pi').1.\mu(\pi'')$, where π' is the sequence containing the first three steps of π and π'' is the sequence containing the last step of π . Further, $\mu(\pi'') = 1$, since there is only one step at root position. Now, the outermost positions of π' are namely 1.1 and 2.1, and hence we have $\mu(\pi') = \mu(\pi'|_{1.1}) + \mu(\pi'|_{2.1})$. Further, $\pi'|_{1.1} = \underline{(0-0)}-0 \rightarrow \underline{0-0} \rightarrow 0$, and $\pi'|_{2.1} = \underline{0-0} \rightarrow 0$. Now, $\mu(\pi'|_{2.1}) = 1$, and the reader can check that $\mu(\pi'|_{1.1}) = 1.1$. So finally, $\mu(\pi) = \mu(\pi').1.\mu(\pi'') = \mu(\pi').1.1 = (\mu(\pi'|_{1.1}) + \mu(\pi'|_{2.1})).1.1 = (1.1 + 1).1.1 = 2.1.1.1$, that indicates that there are two steps at the beginning that can be performed parallelly, followed by three other steps that cannot be performed parallelly.

The following are some useful properties of the metric.

Lemma 3. $|\mu(\pi_1 ; \pi_2)| \leq |\mu(\pi_1)| + |\mu(\pi_2)|$.

Lemma 4. Let $\pi = t_1 \rightarrow^* t_2 \xrightarrow{q} t_3 \rightarrow^* t_4$ and $\rho = t'_1 \rightarrow^* t_3 \rightarrow^* t_4$ be rewrite sequences such that $|\mu(t'_1 \rightarrow^* t_3)| \leq |\mu(t_1 \rightarrow^* t_2)|$, and all the reductions in both $t_1 \rightarrow^* t_2$ and $t'_1 \rightarrow^* t_3$ happen under the position q . Then $\mu(\rho) < \mu(\pi)$.

Lemma 5. Let $\pi = t_1 \rightarrow^* t_2 \xrightarrow{\langle q, l \rightarrow r \rangle} t_3 \rightarrow^* t_4$ where $\langle q, l \rightarrow r \rangle \in DR(t_1)$ and none of the redexes in $t_1 \rightarrow^* t_2$ is in $DR(t_1)$. Then, there is a rewrite sequence $t_1 \xrightarrow{\langle q, l \rightarrow r \rangle} s \rightarrow^* t_4$ such that $\mu(s \rightarrow^* t_4) < \mu(\pi)$.

Theorem 2 (Completeness). If $t \rightarrow^* s$, then there is an s' such that $t \xrightarrow{m} s'$, $root(s') = root(s)$ and $s' \xrightarrow{\geq A} s$.

Proof. Let $\pi = t \xrightarrow{\langle p_1, l_1 \rightarrow r_1 \rangle} t_1 \cdots \xrightarrow{\langle p_n, l_n \rightarrow r_n \rangle} s$. We prove the theorem by noetherian induction on $\mu(\pi)$. The base case $\mu(\pi) = \epsilon$ is obvious, since $|\pi| = 0$. For the induction step there are two cases:

- Suppose there is no i such that $1 \leq i \leq n$ and $\langle p_i, l_i \rightarrow r_i \rangle \in DR(t)$. Then, by Lemma 1, $p_i > A$ for all i , and thus the statement holds by taking $s' = t$.
- Now, we consider the least k such that $\langle p_k, l_k \rightarrow r_k \rangle \in DR(t)$. Then, by Lemma 5, we have $\pi' = t \xrightarrow{\langle p_k, l_k \rightarrow r_k \rangle} t'_1 ; \delta$ for some t'_1 and a rewrite sequence δ with target s . Furthermore, $\mu(\delta) < \mu(\pi)$. By induction hypothesis, we have $t'_1 \xrightarrow{m} s'$ for some s' such that $root(s') = root(s)$ and $s' \xrightarrow{\geq A} s$. Then, the statement follows from the observation that $t \xrightarrow{m} s'$. \square

4 Refinements of the Strategy

In this section, we further refine the natural rewriting strategy, using the notions of *failing terms* and *most frequently demanded positions*, both of which were originally introduced in [9], although not in an explicit way and for left-linear constructor systems.

4.1 Failing Terms

For a position p and a term t , we define the set $R_t(p)$ of *reflections* of p w.r.t. t as follows: if p is under a variable position in t , i.e., $p = q.q'$ for some q such that $t|_q = x$, then $R_t(p) = \mathcal{P}os_x(t).q'$, else $R_t(p) = \{p\}$. We say that the path to p in t is *stable* (or simply p is stable) if $D_t^\uparrow(p) \setminus \{\Lambda\} = \emptyset$.

Definition 6 (Failing term). *Given terms l, t , we say t fails w.r.t. l , denoted by $l \blacktriangleleft t$, if there is $p \in DP_l(t)$ such that p is stable, and one of the following holds: (i) $R_l(p) \cap DP_l(t) = \{p\}$; or (ii) there is $q \in R_l(p) \cap DP_l(t)$ with $root(t|_p) \neq root(t|_q)$, and q is also stable. We denote by $l \blacktriangleleft^* t$ that t is not failing w.r.t. l .*

The idea behind the definition above is that if $l \blacktriangleleft t$, then no sequence of reductions in t will help produce a term to which the rule $l \rightarrow r$ can be applied at the root. We can thus safely ignore the positions demanded by l while computing the set $DR(t)$.

Example 10. Consider the terms $t = 10! \% 0$ and $l_3 = M \% s(N)$ from Example 1. We have that $l_3 \blacktriangleleft t$, because position $2 \in DP_{l_3}(t)$ is stable and $R_{l_3}(2) = \{2\}$. Now, consider the terms $t' = s(Z) \approx 0$ and $l_7 = X \approx X$, again from Example 1. We have $l_7 \blacktriangleleft t'$, since position $1 \in DP_{l_7}(t')$ is stable, $R_{l_7}(1) = \{1, 2\}$, $root(t'|_1) = s \neq 0 = root(t'|_2)$, and position 2 is also stable.

Definition 7 (Demanded redexes). *We improve the set $DR(t)$ in Definition 3 and replace the former set $FP(t)$ by $FP(t) = \bigcup_{l \in L(\mathcal{R}) \wedge l \blacktriangleleft^* t} DP_l(t)$.*

With trivial modifications to the proofs, the correctness and completeness properties of natural rewriting hold with this refined Definition 7 instead of Definition 3.

Example 11. Consider again the term $t_2 = 10! \% (s(0) - s(0)) \approx 10! \% 0$ from Example 7. With the refined Definition 7 we have $DR(t_2) = \{\langle 1.1 \rangle, \langle 1.2 \rangle\}$ and the redex at position 2.1 is not considered anymore. The reason is that from Example 10, it follows that the subterm $10! \% 0$ is failing w.r.t. rules l_3 and l_4 , and hence $SP(t_2|_2) \setminus \{\Lambda\} = \emptyset$. Therefore, we have only two possible rewriting steps from the term t_2 : (i) a rewriting step reducing the subterm $s(0) - s(0)$ at position 1.2, and (ii) a rewriting step reducing the subterm $10!$ at position 1.1. The second rewrite step is still undesirable and its removal motivates the next refinement.

4.2 Most Frequently Demanded Positions

Suppose $l \preceq t$, $l \blacktriangleleft^* t$, and that we have a rewrite sequence $t \xrightarrow{p_1} \dots t_{n-1} \xrightarrow{\langle p_n, l \rightarrow r \rangle} t_n$ where $p_n = \Lambda$. Then, observe that for every $q \in DP_l(t)$, there is a reduction above a position in $R_l(q)$, i.e., there is $q' \in R_l(q)$ and k such that $p_k \in D_t^\uparrow(q')$; clearly, only then it is possible that $l \leq t_{n-1}$. Now, recall that we are only interested in computing a demanded set of redexes in t such that before any rule can

be applied at the root position in t , *at least* one of the redexes in the demanded set has to be reduced. Therefore, while computing the set $SP(t)$ in Definition 7, for each $l \in L(\mathcal{R})$ such that $l \blacktriangleleft t$, instead of considering (the defined symbols above) every position in $DP_l(t)$, it is sufficient to consider only the positions $R_l(q)$ for at least one $q \in DP_l(t)$. This motivates the following refinement of Definition 7.

Definition 8 (Set cover). *For a set of positions P , a sequence of lhs's l_1, \dots, l_n , and a sequence of sets of positions Q_1, \dots, Q_n , we say that P covers l_1, \dots, l_n and Q_1, \dots, Q_n if for all $1 \leq i \leq n$, there is a position $p \in P \cap Q_i$ such that $R_{l_i}(p) \subseteq P$.*

Definition 9 (Demanded redexes). *We improve the set $DR(t)$ in Definition 7 and replace the former set $FP(t)$ by the set $FP(t)$ returning one of the minimal sets of positions that cover l_1, \dots, l_n and $DP_{l_1}(t), \dots, DP_{l_n}(t)$, where $\{l_1, \dots, l_n\} = \{l \in L(\mathcal{R}) \mid l \blacktriangleleft t\}$.*

The set $FP(t)$ above is a minimal set cover that is closed under reflection. Roughly, minimality amounts to giving priority to those positions in t that are demanded by the maximum number of rules, i.e., what we call the *most frequently demanded positions*. This idea of giving priority to 'popular' demanded positions is familiar from other lazy evaluation strategies such as outermost needed rewriting [4], and was first formalized in a similar fashion as above in [9]. With trivial modifications to the proofs, the correctness and completeness properties of natural rewriting hold also with the above refinement.

Example 12. Consider the subterm $t = 10! \% (\mathbf{s}(0) - \mathbf{s}(0))$ of the term t_2 in Example 1. Consider also the left-hand sides of the rules (3) and (4): $l_3 = \mathbf{M} \% \mathbf{s}(\mathbf{N})$ and $l_4 = (\mathbf{O} - \mathbf{s}(\mathbf{M})) \% \mathbf{s}(\mathbf{N})$. We have that $DP_{l_3}(t) = \{2\}$, $DP_{l_4}(t) = \{1, 2\}$, and $DP_{l'}(t) = \{A\}$ for any other lhs l' . Then, the set $P = \{2, A\}$ covers all lhs's. Now, let us continue with term $t_2 = 10! \% (\mathbf{s}(0) - \mathbf{s}(0)) \approx 10! \% 0$ from Example 11. With Definition 9, the redexes computed by the natural rewriting strategy become even more refined. Specifically, we have $DR(t_2) = \{\langle 1.2 \rangle\}$ and the redex at position 1.1 is not considered anymore. The reason is that since position 2 in $10! \% (\mathbf{s}(0) - \mathbf{s}(0))$ is enough to obtain a set cover of all the positions demanded by rules (3) and (4), position 1 in the subterm $t_2|_1$ is not considered as demanded, i.e., $SP(t_2|_1) = FP(t_2|_1) = \{2, A\}$. Finally, we have only the optimal rewriting step for position 1.2 from the term t_2 and the optimal rewrite sequence:

$$\begin{aligned} & 10! \% (\mathbf{s}(0) - \mathbf{s}(0)) \approx 10! \% 0 \\ & \rightarrow 10! \% (\mathbf{O} - \mathbf{O}) \approx 10! \% 0 \rightarrow 10! \% 0 \approx 10! \% 0 \rightarrow \mathbf{True} \end{aligned}$$

Note that we have $DR(t_3) = \{\langle 1.2 \rangle\}$ and $DR(t_4) = \{\langle A \rangle\}$ for the terms $t_3 = 10! \% (\mathbf{O} - \mathbf{O}) \approx 10! \% 0$ and $t_4 = 10! \% 0 \approx 10! \% 0$ above.

5 Conclusion

We have extended natural rewriting to general rewriting systems while preserving correctness and completeness w.r.t. head-normal forms. A noteworthy feature

of this generalization is that it is conservative, i.e., the generalized strategy coincides with the original one for the class of left-linear constructor systems. This makes the strategy available for both expressive equational languages and for rewriting logic languages. Since our generalization is conservative, we inherit all the optimality results presented in [9] for left-linear constructor systems. An important problem for future research is to identify optimality results for this new generalized natural rewriting strategy. We believe that the notion of *inductively sequential terms* introduced in [9] can provide significant insights for the more general optimality results, since this notion identifies specific terms, rather than classes of rewrite systems, that can be optimally evaluated. Another observation is that our generalized natural rewriting is easily implementable, since the demanded set of redexes is computed using simple recursive procedures. Indeed in [10], we have proposed a technique for the efficient implementation of the natural rewriting strategy of [9] for left-linear constructor systems, which moves the computation of the demanded set of redexes to compilation phase instead of execution phase. Extending this implementation technique to the generalized rewriting strategy would be considered as future work. However, a complexity analysis of the generalized natural rewriting strategy is also planned.

This work provides a basis for a subsequent generalization of natural rewriting first to narrowing, already achieved in [12], and second to even more expressive rewrite theories suited for concurrent system specifications [20] and supporting: (i) sorts and subsorts; (ii) rewriting *modulo* axioms such as associativity, commutativity, identity, and so on; and (iii) conditional rewriting.

Acknowledgements S. Escobar has been partially supported by projects MEC TIN 2004-07943-C04-02, EU ALA/95/23/2003/077-054, GV Grupos03/025 and grant 2667 of *Universidad Politécnica de Valencia* during a stay at Urbana-Champaign, USA.

References

1. 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.
2. S. Antoy. Constructor-based conditional narrowing. In *Proc. of 3rd International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, PPDP'01*, pages 199–206, Florence, Italy, Sept. 2001. ACM.
3. 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.
4. S. Antoy, R. Echahed, and M. Hanus. A needed narrowing strategy. In *Journal of the ACM*, volume 47(4), pages 776–822, 2000.
5. 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.
6. P. Borovanský, C. Kirchner, H. Kirchner, and P.-E. Moreau. ELAN from a rewriting logic point of view. *Theoretical Computer Science*, 285:155–185, 2002.

7. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285:187–243, 2002.
8. A. Deursen, J. Heering, and P. Klint. *Language Prototyping: An Algebraic Specification Approach*. World Scientific, 1996.
9. S. Escobar. Refining weakly outermost-needed rewriting and narrowing. In D. Miller, editor, *Proc. of 5th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, PPDP'03*, pages 113–123. ACM Press, New York, 2003.
10. S. Escobar. Implementing natural rewriting and narrowing efficiently. In Y. Kameyama and P. J. Stuckey, editors, *7th International Symposium on Functional and Logic Programming (FLOPS 2004)*, volume 2998 of *Lecture Notes in Computer Science*, pages 147–162. Springer-Verlag, Berlin, 2004.
11. S. Escobar, J. Meseguer, and P. Thati. Natural narrowing as a general unified mechanism for programming and proving. Technical Report DSIC-II/16/04, DSIC, Universidad Politécnic de Valencia, 2004. Available at <http://www.dsic.upv.es/users/elp/papers.html>.
12. S. Escobar, J. Meseguer, and P. Thati. Natural narrowing for general term rewriting systems. In J. Giesl, editor, *Proc. of 16th International Conference on Rewriting Techniques and Applications, RTA'05*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, 2005.
13. K. Futatsugi and R. Diaconescu. *CafeOBJ Report*. World Scientific, AMAST Series, 1998.
14. E. Giovannetti, G. Levi, C. Moiso, and C. Palamidessi. Kernel Leaf: A Logic plus Functional Language. *Journal of Computer and System Sciences*, 42(2):139–185, 1991.
15. J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. Introducing OBJ. In *Software Engineering with OBJ: Algebraic Specification in Action*, pages 3–167. Kluwer, 2000.
16. J. C. González-Moreno, M. T. Hortalá-González, F. J. López-Fraguas, and M. Rodríguez-Artalejo. An approach to declarative programming based on a rewriting logic. *Journal of Logic Programming*, 40(1):47–87, 1999.
17. D. Hofbauer and M. Huber. Linearizing term rewriting systems using test sets. *Journal of Symbolic Computation*, 17:91–129, 1994.
18. 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.
19. R. Loogen, F. López-Fraguas, and M. Rodríguez-Artalejo. A Demand Driven Computation Strategy for Lazy Narrowing. In *Proc. of PLILP'93*, volume 714 of *Lecture Notes in Computer Science*, pages 184–200. Springer-Verlag, Berlin, 1993.
20. J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.
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. S. Peyton-Jones. *The Implementation of Functional Programming Languages*. Prentice Hall International, London, 1987.
23. B. Salinier and R. Strandh. Efficient simulation of forward-branching systems with constructor systems. *Journal of Symbolic Computation*, 22:381–399, 1996.
24. R. Sekar and I. Ramakrishnan. Programming in equational logic: Beyond strong sequentiality. *Information and Computation*, 104(1):78–109, 1993.