Natural Narrowing for General Term Rewriting Systems

Santiago Escobar[1], José Meseguer[2], and Prasanna Thati[3]

[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. For narrowing to be an efficient evaluation mechanism, several *lazy* narrowing strategies have been proposed, although typically for the restricted case of left-linear constructor systems. These assumptions, while reasonable for functional programming applications, are too restrictive for a much broader range of applications to which narrowing can be fruitfully applied, including applications where rules have a non-equational meaning either as *transitions* in a concurrent system or as *inferences* in a logical system. In this paper, we propose an efficient lazy narrowing strategy called *natural narrowing* which can be applied to general term rewriting systems with no restrictions whatsoever. An important consequence of this generalization is the *wide range of applications* that can now be efficiently supported by narrowing, such as symbolic model checking and theorem proving.

1 Introduction

Rewriting is currently recognized as a very general declarative formalism to specify, program, and reason about computational systems. The more traditional applications have been in the context of equational reasoning and of equational/functional programming, where the rewriting relation is understood as *oriented equality*. But there is an increasing awareness of the usefulness of rewriting in non-equational contexts, where a rewrite is understood as a *transition* or an *inference*: for example to specify and program concurrent systems or logical inference systems. This has led, for example, to theoretical developments such as rewriting logic [28], and to the development of language implementations supporting non-equational rewriting such as ELAN [6] and Maude [7].

A similar widening of the scope is needed for *narrowing*, which generalizes rewriting by performing unification in nonvariable positions instead of the usual matching. Narrowing can in this way endow rewriting languages with new programming and reasoning capabilities in a much wider setting and for a much wider range of applications than those based on equational logic. The traditional understanding of narrowing [15,23] has been as a mechanism for equational *unification*, that is, for solving equational goals $E \vdash (\exists \overrightarrow{x})\ t = t'$; as a consequence, properties such as confluence and termination have often been assumed, and many equational reasoning applications, as well as a number of functional/logic programming languages supporting narrowing, have been developed. As proposed for example in [29], in a non-equational setting narrowing can instead be understood as a powerful mechanism for solving *reachability goals* $\mathcal{R} \vdash (\exists \overrightarrow{x})\ t \to^* t'$ in a rewrite theory $\mathcal{R}$. The traditional equational interpretation can still be kept as a special case since, as explained in Section 4.2, solving equations becomes a special case of solving more general reachability goals. However, in this wider setting traditional assumptions such as confluence and termination are in general no longer reasonable (see Section 5 and [29,34] for a discussion of completeness issues for narrowing in this setting).

A key challenge for narrowing is the danger of *combinatorial explosion* in exploring the narrowing tree. It becomes in fact essential in practice to use adequate *narrowing strategies* that are as greedy as possible, yet remain complete. One important breakthrough in this direction was the realization that one could extend the work on optimal lazy *reduction* strategies originating with Huet and Levy [22], and extended in different ways by other researchers (see, e.g., [32,1,3,11]) to obtain efficient lazy *narrowing* strategies that only instantiate those positions that are really needed. This was first achieved by Antoy, Echahed and Hanus, who extended their (weakly) outermost-needed rewriting strategy to a *(weakly) needed narrowing* strategy [4,3]. Recently, both (weakly) outermost-needed rewriting and (weakly) outermost-needed narrowing have been further improved by Escobar by means of the *natural rewriting* and *natural narrowing* strategies [11,12]. We postpone a more detailed discussion of related work in this area until Section 5. For the moment, the main point to bear in mind is that most of the work on lazy rewriting and lazy narrowing strategies [17,25,19,32,1,3,4,2,11,12] has taken place within the context of functional (logic) programming languages, and depends on assumptions such as having *left-linear* and *constructor* rules. These assumptions are reasonable for some functional (logic) programming languages, but they substantially limit the expressive power of equational languages such as OBJ [18], CafeOBJ [16], ASF+SDF [10], and the equational subset of Maude [7], where non-linear left-hand sides which need not be constructor-based are perfectly acceptable. Such assumptions become even more restrictive and unreasonable for non-equational rewriting languages such as ELAN [6] and Maude [7], where a rewrite $t \to t'$ is no longer understood as a step of equational simplification but as a transition, and where the rules need not be confluent nor terminating, need not be left-linear, and the constructor assumption is utterly unreasonable and almost never holds.

The goal of this paper is to propose an efficient lazy narrowing strategy called *generalized natural narrowing* that greatly extends the natural narrowing strategy of [11] and keeps all the good properties while overcoming all the above limitations. In fact our strategy can be applied to *completely general* rewrite systems with no restrictions whatsoever: even rewrite rules with extra variables in their righthand sides are allowed; furthermore, we allow rewritings to be *context-sensitive* [26] according to a function ϕ specifying which argument positions in each function symbol are *frozen*, so that rewriting in the subterms at those positions is forbidden. In this way, we obtain a general lazy narrowing strategy applicable in the broader setting of solving reachability goals for rewrite systems whose rules can have a non-equational semantics. Furthermore, this generalization is obtained together with *actual gains in efficiency*, in the sense that our natural narrowing strategy, besides being efficiently implementable, performs strictly better than previously proposed lazy strategies when specialized to their setting (see Section 5). A further efficiency advantage, generalizing that of [4,3,11,12], is that, as explained in Section 3, natural narrowing computes substitutions in an *incremental* way, without explicit use of a unification algorithm. Perhaps the most important consequence of the generality of our natural narrowing strategy is the *wide range of applications* that can now be supported. To give the reader a better feeling for some of these application areas, including symbolic model checking and theorem proving, we discuss those areas, and the respective benefits of using natural narrowing for each of them, in Section 4. What emerges, in summary, is a general and efficient *unified mechanism*, seamlessly integrating rewriting and narrowing, and making it available for a very wide range of programming and proving applications.

To give the reader a first intuitive feeling for how our generalized natural narrowing works, and for the difficulties that it resolves, we illustrate some of the key issues by means of a simple example.

Example 1. Consider the following rewrite system for proving equality ($\approx$) of arithmetic expressions built using modulus or remainder (%), subtraction ($-$), and minimum (`min`) operations on natural numbers.

```
(1) M % s(N) → (M−s(N)) % s(N)     (5) min(0, N) → 0
(2) (0 − s(M)) % s(N) → N − M      (6) min(s(N),0) → 0
(3) M − 0 → M                      (7) min(s(N),s(M)) → s(min(M,N))
(4) s(M) − s(N) → M−N              (8) X ≈ X → True
```

Note that this rewrite system is not left-linear because of rule (8) and it is not constructor-based because of rule (2). Furthermore, note that it is neither terminating nor confluent due to rule (1).

The aim of natural rewriting and narrowing strategies [11] is to lazily compute head-normal forms of a given term t. Specifically, given a term that is not a head-normal form, the strategy reduces (narrows) to the extent possible, only those redexes (narroxes) that are necessary for a rule to be applied at the root. We would like to generalize natural rewriting and narrowing to a version that enjoys the good optimality properties of natural rewriting and natural narrowing (see [11]) and that can also handle non-left-linear and non-constructor rules such as (2) and (8). This is accomplished for rewriting in the *generalized natural rewriting strategy* of [14]. For example, consider the term[4] $t_1 = \texttt{10!}\,\%\,\texttt{min(X,X-0)} \approx \texttt{10!}\,\%\,\texttt{0}$ and the following two narrowing sequences we are interested in amongst all possible. First, the following sequence leading to `True`, that starts by unifying subterm $t_1|_{1.2}$ with left-hand side (lhs) l_5:

[4] The subterm `10!` represents factorial of $\texttt{s}^{10}\texttt{(0)}$ 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.

$$\texttt{10!\%}\underline{\texttt{min(X,X-0)}} \approx \texttt{10!\%0} \leadsto_{[\texttt{X}\mapsto\texttt{0}]} \underline{\texttt{10!\%0} \approx \texttt{10!\%0}} \leadsto_{id} \texttt{True}$$

Second, the following sequence not leading to `True`, that starts by reducing subterm $t_1|_{1.2.2}$ with lhs l_3 and that early instantiates variable `X`:

$$\texttt{10!\%min(X,}\underline{\texttt{X-0}}\texttt{)} \approx \texttt{10!\%0} \leadsto_{[\texttt{X}\mapsto\texttt{s(X')}]} \texttt{10!\%}\underline{\texttt{min(s(X'),s(X'))}} \approx \texttt{10!\%0}$$
$$\leadsto_{id} \quad \texttt{10!\%s(min(X',X'))} \approx \texttt{10!\%0}$$

Note that although it is possible to further narrow the last term, we are not interested in doing so, since such term is already a head-normal form. In the following, we informally introduce the key points of our strategy:

1. (*Demanded positions*). This notion is relative to a lhs l and determines which positions in a term t should be narrowed in order to be able to apply lhs l at root position. For the term t_1 = `10!%min(X,X-0)` ≈ `10!%0` and lhs l_8 = `X` ≈ `X`, only subterm `min(X,X-0)` is demanded, since it is the only disagreeing part in $t_1|_1$ w.r.t. $t_1|_2$.
2. (*Failing term*). This notion is relative to a lhs l and stops further wasteful narrowing steps. Specifically, the last term `10!%s(min(X',X'))` ≈ `10!%0` of the second former sequence fails w.r.t. l_8, since the subterm `s(min(X',X'))` is demanded by l_8 but there is no possible narrowing step above it that would convert it into term `0`.
3. (*Most frequently demanded positions*). This notion determines those demanded positions w.r.t. non-failing lhs's that are demanded by the maximum number of rules and that cover all such non-failing lhs's. It provides the optimality properties of our natural rewriting and narrowing strategies, since it substantially reduces the set of positions to be considered. If we look closely at lhs's l_5, l_6, and l_7 defining `min`, we can see that position 1 in the term `min(X,X-0)` is more demanded than position 2, i.e., position 1 is disagreeing w.r.t. l_5, l_6, and l_7, whereas position 2 is disagreeing only w.r.t. l_6 and l_7. Thus, position 1 is the most frequently demanded position for all rules defining `min` that also covers such rules. Note that position 1 is rooted by a variable and this motivates the following point.
4. (*Lazy instantiation*). This notion relates to an incremental construction of unifiers without the explicit use of a unification algorithm. This is necessary in the previous example, since subterm `min(X,X-0)` does not unify with lhs l_6 and l_7. However, we can deduce that narrowing at subterm `X-0` is only necessary when substitution [`X` ↦ `s(X')`], inferred from l_6 and l_7, has been applied. Thus, we early construct the appropriate substitutions [`X` ↦ `0`] and [`X` ↦ `s(X')`] in order to reduce the search space.

In Section 2, we present the preliminary background. In Section 3 we define our generalized natural narrowing strategy. In Section 4, we motivate our work by illustrating various applications of the generalized narrowing strategy. In Section 5, we compare our work with related approaches, and we conclude in Section 6. Proofs of all results can be found in [13].

2 Preliminaries

We assume some familiarity with term rewriting and narrowing (see [33] for missing definitions). We assume a finite alphabet (function symbols) Σ and a countable set of variables X. We denote the set of terms built from Σ and X by $T_\Sigma(X)$ and write T_Σ for ground terms. 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 $S \subseteq \Sigma \cup X$, $\mathcal{P}os_S(t)$ denotes positions in t where symbols or variables in S occur. We write $\mathcal{P}os_f(t)$ and $\mathcal{P}os(t)$ as a shorthand for $\mathcal{P}os_{\{f\}}(t)$ and $\mathcal{P}os_{\Sigma\cup X}(t)$, respectively. We denote the *root position* by Λ. Given positions p, q, we denote its concatenation as $p.q$. 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 $root(t)$.

A *substitution* is a function $\sigma : X \to T_\Sigma(X)$ which maps variables to terms, and which is different from the identity only for a finite subset $\mathcal{D}om(\sigma)$ of X. We homomorphically extend substitutions to terms. We denote by id the identity substitution: $id(x) = x$ for all $x \in X$. 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 $l \to r$, with $l \notin X$. A *rewrite system*[5] is a triple $\mathcal{R} = (\Sigma, \phi, R)$ with Σ a signature, R a set of rewrite rules, and $\phi : \Sigma \to \mathcal{P}(\mathbb{N})$ specifies the *frozen* arguments $\phi(f) \subseteq \{1, \ldots, k\}$ for the arity k of f. We say position p in t is *frozen* if $\exists q < p$ such that $p = q.i.q'$ and $i \in \phi(root(t|_q))$. $L(\mathcal{R})$ denotes the set of lhs's of $\mathcal{R}$. A rewrite system $\mathcal{R}$ is left-linear if for all $l \in L(\mathcal{R})$, l is a linear term. Given $\mathcal{R} = (\Sigma, \phi, R)$, we assume that Σ is defined as the disjoint union $\Sigma = \mathcal{C} \uplus \mathcal{D}$ of symbols $c \in \mathcal{C}$, called *constructors*, and symbols $f \in \mathcal{D}$, called *defined symbols*, where $\mathcal{D} = \{root(l) \mid l \to r \in R\}$ and $\mathcal{C} = \Sigma - \mathcal{D}$. A pattern is a term $f(l_1, \ldots, l_k)$ where $f \in \mathcal{D}$ and $l_i \in T_\mathcal{C}(X)$, for $1 \leq i \leq k$. A rewrite system $\mathcal{R} = (\mathcal{C} \uplus \mathcal{D}, \phi, R)$ is a constructor system (CS) if every $l \in L(\mathcal{R})$ is a pattern.

A term t rewrites to s at a non-frozen position $p \in \mathcal{P}os_\mathcal{D}(t)$ using the rule $l \to r \in R$, called a *rewrite step* and written $t \to_{\langle p, l \to r\rangle} s$ ($t \xrightarrow{p} s$ or simply $t \to s$), if $t|_p = \sigma(l)$ and $s = t[\sigma(r)]_p$. The pair $\langle p, l \to r\rangle$ is called a redex. A term t is a *normal form* if it contains no redex. A term t is a *head-normal form* if it cannot be reduced to a redex. We denote by $\xrightarrow{>\Lambda}$ a rewrite step at a position $p > \Lambda$. A substitution σ is called *normalized* if $\sigma(x)$ is a normal form for all variables x. Similarly, a *narrowing step* is defined as $t \rightsquigarrow_{\langle p, l \to r, \sigma\rangle} s$ (or simply $t \rightsquigarrow_\sigma s$) if $\sigma(t) \to_{\langle p, l \to r\rangle} s$. Note that we do not require $p \in \mathcal{P}os_\mathcal{D}(t)$ for a narrowing step, which is a usual condition for left-linear constructor systems. Instead, we require substitutions computed by narrowing to be normalized.

[5] What we call here a rewrite system is a special case of a *rewrite theory* [28], which is a 4-tuple (Σ, ϕ, E, R) with E a set of equations.

3 Generalizing Natural Narrowing

In [14], we have generalized the natural rewriting strategy of [11] to the larger class of rewrite systems that need not be left-linear and constructor-based. In this paper, we define, again for this larger class of systems, a generalized natural narrowing strategy that, to the extent possible, performs only those narrowing steps that are essential for applying some rule at root position. That is, if a term t is not a head-normal form (or some substitution making the term not a head-normal form exists), then we know that after a (possibly empty) sequence of narrowing steps at positions other than the root, a narrowing step at the root position with a rule $l \to r$ is possible. We use the notions of demanded positions, failing terms, and most frequently demanded positions, as in [14]. First, we recall the notion of demanded positions.

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

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*

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

Intuitively, the set $DP_l(t)$ returns a set of positions in t at which t necessarily has to be "changed" (either by a narrowing step if it is a non-variable position, or by an instantiation if it is a variable position) before the rule $l \to r$ can be applied at the root position, i.e., before l can match the term under consideration.

Example 2. Consider the left-hand side l_8 = `X` $\approx$ `X` and the term t_1 = `10!` $\%$ `min(X,X-0)` $\approx$ `10!` $\%$ `0` of Example 1. The least general context of l_8 and t_1 is s = `W` $\approx$ `Z`. Now, for $\sigma = \{$`W` $\mapsto$ `X`, `Z` $\mapsto$ `X`$\}$, we have $\sigma(s) = l_8$. Then, while computing $DP_{l_8}(t_1)$, we compute the set of disagreeing positions between the subterms in t_1 corresponding to the non-linear variable `X` in l_8, i.e., the set $\mathcal{P}os_{\neq}(T)$ for $T = t_1|_{\{1,2\}} = \{$`10!` $\%$ `min(X,X-0)`, `10!` $\%$ `0`$\}$. According to Definition 1, the least general context of T is the term `10!` $\%$ `Y` and the set of disagreeing positions between terms in T is then $\mathcal{P}os_{\neq}(T) = \{2\}$. Thus, we obtain that $DP_{l_8}(t_1) = \{1,2\}.\{2\} = \{1.2, 2.2\}$.

Now, four points have to be addressed. First, note that the symbol at a position $p \in DP_l(t)$ in t can be changed not only by a narrowing step or by instantiation at p, but also by a narrowing step or instantiation at a position

$q < p$. Thus, besides considering the positions in $DP_l(t)$ as candidates for evaluation, 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 $D_t^{\uparrow}(q) = \{p \mid p \leq q \ \wedge \ p \in \mathcal{P}os_{\mathcal{D}}(t) \wedge p \text{ is not frozen}\}$. We lift this to sets of positions as $D_t^{\uparrow}(Q) = \cup_{q \in Q} D_t^{\uparrow}(q)$. This gives us the following useful result that shows how demandedness captures the neededness of positions in rewrite sequences for the application of a rule at root position.

Lemma 1. [14] *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, either $l_n \leq t$ or there is a k, $1 \leq k < n$, such that $p_k \in D_t^{\uparrow}(DP_{l_n}(t))$.*

Example 3. Continuing Example 2, we have $D_{t_1}^{\uparrow}(DP_{l_8}(t_1)) = D_{t_1}^{\uparrow}(\{1.2, 2.2\}) = \{\Lambda, 1, 1.2, 2\}$, since position 2.2 is rooted by a constructor symbol, and thus removed.

Second, we only compute $DP_l(t)$ for those left-hand sides l such that t does not *fail* w.r.t. l. Roughly, we say t fails w.r.t. l if no sequence of narrowing steps or instantiations in t will help to produce a term to which the rule $l \rightarrow r$ can be applied at the root; this notion is undecidable but we provide a safe and computable approximation.

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 $root(t|_p) \notin X$ and $D_t^{\uparrow}(p) \setminus \{\Lambda\} = \varnothing$.

Definition 3 (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 $l \not\blacktriangleleft t$ that t is not failing w.r.t. l.*

Example 4. Consider the subterm $t_1|_2 =$ `10!` % `0` and the lhs's $l_1 =$ `M` % `s(N)` and $l_2 =$ `(0 - s(M))` % `s(N)` in Example 1. We have that $l_1 \blacktriangleleft t_1|_2$ and $l_2 \blacktriangleleft t_1|_2$, because position 2 in $t_1|_2$ belongs to $DP_{l_1}(t_1|_2)$ and $DP_{l_2}(t_1|_2)$, is stable, and $R_{l_1}(2) = R_{l_2}(2) = \{2\}$. Similarly, the term $t' =$ `10!` % `s(min(X',X'))` $\approx$ `10!` % `0` fails w.r.t. l_8 since $1.2, 2.2 \in DP_{l_8}(t')$, 1.2, 2.2 are stable, and $R_{l_8}(1.2) = R_{l_8}(2.2) = \{1.2, 2.2\}$.

Third, we do not consider all positions in each $DP_l(t)$ but a subset called the *most frequently demanded positions.* The idea behind this is that narrowing or instantiating at those positions is enough for being able to reduce at root position. In this way, we can substantially reduce (or optimize) the set of positions to be considered.

Definition 4 (Set cover). *For a set of positions P, a sequence of lhs's $l_1, \ldots, l_n$, and a sequence of sets of positions $Q_1, \ldots, Q_n$, we say that P* covers *$l_1, \ldots, l_n$ and $Q_1, \ldots, Q_n$ 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 5 (Most frequently Demanded positions). *We define the filtered set of demanded positions of a term t by the set $FP(t)$ returning one of the minimal sets of positions that cover $l_1, \ldots, l_n$ and $DP_{l_1}(t), \ldots, DP_{l_n}(t)$, where $\{l_1, \ldots, l_n\} = \{l \in L(\mathcal{R}) \mid l \not\blacktriangleleft t\}$.*

Example 5. Consider the subterm $t_1|_1 =$ `10! % min(X,X-0)` and the lhs's l_1 and l_2 in Example 1. The reader can check that $DP_{l_1}(t_1|_1) = \{2\}$, $DP_{l_2}(t_1|_1) = \{1, 2\}$, and $DP_l(t_1|_1) = \{\Lambda\}$ for any other lhs l. Then, the set $P = \{2, \Lambda\}$ covers all lhs's and we obtain that $FP(t_1|_1) = \{\Lambda, 2\}$. On the other hand, consider the term t_1 and the lhs l_8 in Example 1. From Example 2, we have $DP_{l_8}(t_1) = \{1.2, 2.2\}$ and $DP_l(t_1) = \{\Lambda\}$ for any other lhs l. Thus, $FP(t_1) = \{\Lambda, 1.2, 2.2\}$, since this set is closed by reflection.

Fourth, whenever t matches l for a rule $l \to r$, in addition to reducing t with $l \to r$ we have to consider as candidates for evaluation those positions q in t that have a defined symbol and that are above a variable position in l; see [14] for further explanations. Now, we are able to define the set of *sufficient demanded positions* that collects the four previous ideas.

Definition 6 (Sufficient demanded positions). *We define the* sufficient set of demanded positions *of a term t as $SP(t) = \cup_{l \in L(\mathcal{R}) \wedge l \leq t} D_l^{\uparrow}(\mathcal{P}os_{\mathcal{X}}(l)) \quad \cup$ $D_t^{\uparrow}(FP(t))$.*

Example 6. Consider term t_1 in Example 1. Since t_1 is not a redex, we have $SP(t_1) = D_{t_1}^{\uparrow}(FP(t_1))$. By Example 5, we have $SP(t_1) = D_{t_1}^{\uparrow}(\{\Lambda, 1.2, 2.2\})$. And by Example 3, we have $SP(t_1) = \{\Lambda, 1, 1.2, 2\}$.

Before defining generalized natural narrowing, we have to define a set of demanded substitutions for narrowing, to address the question of what substitutions the variables at demanded positions are to be instantiated with.

Definition 7 (Demanded substitutions). *We define the set $DSub(t)$ of* demanded substitutions *for narrowing t at the root position as follows. For each pair of $p \in FP(t) \cap \mathcal{P}os_{\mathcal{X}}(t)$ and $l \in L(\mathcal{R})$ such that $p \in DP_l(t)$ and $l \not\blacktriangleleft t$, we construct the substitution σ as explained below, and stipulate that $\sigma \in DSub(t)$.*

If $p \in \mathcal{P}os_{\Sigma}(l)$, then $\sigma = [t|_p \mapsto root(l|_p)(\overrightarrow{w})]$ for distinct fresh variables $\overrightarrow{w}$. On the other hand, if $p \notin \mathcal{P}os_{\Sigma}(l)$, then we know that p is under a non-linear variable position in l, i.e., $|R_l(p) \cap DP_l(t)| > 1$, and let $Q = R_l(p) \cap DP_l(t)$. There are two cases: (i) if all the terms in $t|_Q$ are variables, then we define σ to be such that for every $q \in Q$ we have $\sigma(t|_q) = w$ for a fresh variable w, (ii) if every non-variable term in $t|_Q$ is rooted by the same symbol f, then we define $\sigma = [t|_p \mapsto f(\overrightarrow{w})]$.

Note that, in the previous definition, if there are two non-variable positions $p_1, p_2 \in Q$ such that $t|_{p_1}$ and $t|_{p_2}$ are rooted with different symbols, then no substitution can resolve the conflict between the disagreeing positions p_1 and p_2 and they are demanded for evaluation.

We can deduce the following useful result that ensures that appropriate substitutions are inferred from left-hand sides for demanded variable positions.

Lemma 2. *Let l, t, σ, and $p \in \mathit{FP}(t) \cap \mathcal{P}os_{\mathcal{X}}(t)$. If $l \not\blacktriangleleft t$, $p \in \mathit{DP}_l(t)$, and $p \notin \mathit{DP}_l(\sigma(t))$, then there exists $\theta \in \mathit{DSub}(t)$ s.t. $\theta(t|_p) \neq t|_p$ and $\theta|_{\mathcal{V}ar(t)} \leq \sigma|_{\mathcal{V}ar(t)}$.*

Example 7. Consider the subterm $t_1|_{1.2}$ = `min(X,X−0)` and the lhs's l_5 = `min(0,N)`, l_6 = `min(s(N),0)`, and l_7 = `min(s(N),s(M))` of Example 1. The reader can check that $\mathit{DP}_{l_5}(t_1|_{1.2}) = \{1\}$, $\mathit{DP}_{l_6}(t_1|_{1.2}) = \{1,2\}$, $\mathit{DP}_{l_7}(t_1|_{1.2}) = \{1,2\}$, and $\mathit{DP}_l(t_1|_{1.2}) = \{\Lambda\}$ for any other lhs l. Thus $\mathit{FP}(t_1|_{1.2}) = \{\Lambda, 1\}$ according to Definition 5. Then, position 1 is a variable position and we have that the demanded substitutions for its variable `X` are $\mathit{DSub}(t_1|_{1.2}) = \{[\mathtt{X} \mapsto \mathtt{0},\ \mathtt{X} \mapsto \mathtt{s(X')}]\}$, since $1 \in \mathcal{P}os_{\Sigma}(l_5)$, $1 \in \mathcal{P}os_{\Sigma}(l_6)$, $\mathit{root}(l_5|_1) = \mathtt{0}$, and $\mathit{root}(l_6|_1) = \mathtt{s}$.

Thus, substitutions are computed in a lazy and *incremental* fashion *without resorting to an explicit unification algorithm.* Now, we formally define our generalized natural narrowing strategy.

Definition 8 (Generalized Natural Narrowing). *We define the set of* demanded narroxes *of a term t as*

$$\mathit{DN}(t) = \{\langle \Lambda, l \to r, id\rangle \mid l \in L(\mathcal{R}) \wedge l \leq t\} \ \cup \ \bigcup_{q \in SP(t) \setminus \{\Lambda\}} q.\mathit{DN}(t|_q) \ \cup \ \bigcup_{\sigma \in \mathit{DSub}(t)} \mathit{DN}(\sigma(t))@\sigma$$

where for a set of narroxes S we define $q.S = \{\langle q.p, l \to r, \theta\rangle \mid \langle p, l \to r, \theta\rangle \in S\}$ and $S@\sigma = \{\langle p, l \to r, \theta \circ \sigma\rangle \mid \langle p, l \to r, \theta\rangle \in S\}$. We say that term t reduces by natural narrowing *to term s, denoted by $t \overset{\mathtt{m}}{\rightsquigarrow}_{\langle p, l \to r, \sigma\rangle} s$ (or simply $t \overset{\mathtt{m}}{\rightsquigarrow} s$) if $t \rightsquigarrow_{\langle p, l \to r, \sigma\rangle} s$, $\langle p, l \to r, \sigma\rangle \in \mathit{DN}(t)$, and $p \in \mathcal{P}os_{\mathcal{D}}(t)$.*

In the following, we omit the rule $l \to r$ in a narrowing step $\langle p, l \to r, \sigma\rangle$, whenever there is no scope for ambiguity about the rule.

Example 8. Consider again the term t_1 = `10! % min(X,X−0)` $\approx$ `10! % 0` from Example 1 and the computation of $\mathit{DN}(t_1)$. Since t_1 is not a redex, we have that $\mathit{DN}(t_1) = \cup_{q \in SP(t_1) \setminus \{\Lambda\}} q.\mathit{DN}(t_1|_q) \ \cup \ \cup_{\sigma \in \mathit{DSub}(t_1)} \mathit{DN}(\sigma(t_1))@\sigma$. By Example 6, $SP(t_1) = \{\Lambda, 1, 1.2, 2\}$. We also have $\mathit{DSub}(t_1) = \varnothing$, since no position in $\mathit{FP}(t_1)$ is a variable. Thus, $\mathit{DN}(t_1) = 1.\mathit{DN}(t_1|_1) \cup 2.\mathit{DN}(t_1|_2) \cup 1.2.\mathit{DN}(t_1|_{1.2})$. This implies that we recursively compute $\mathit{DN}(t_1|_1)$, $\mathit{DN}(t_1|_{1.2})$ and $\mathit{DN}(t_1|_2)$.

Now consider $\mathit{DN}(t_1|_{1.2}) = \mathit{DN}(\mathtt{min(X,X-0)})$. Since it is not a redex, we have $\mathit{DN}(t_1|_{1.2}) = \cup_{q \in SP(t_1|_{1.2}) \setminus \{\Lambda\}} q.\mathit{DN}(t_1|_{1.2.q}) \ \cup \ \cup_{\sigma \in \mathit{DSub}(t_1|_{1.2})} \mathit{DN}(\sigma(t_1|_{1.2}))@\sigma$. By Example 7, we have $SP(t_1|_{1.2}) = D^{\uparrow}_{t_1|_{1.2}}(\mathit{FP}(t_1|_{1.2})) = D^{\uparrow}_{t_1|_{1.2}}(\{\Lambda, 1\}) = \{\Lambda\}$ and $\mathit{DSub}(t_1|_{1.2}) = \{\sigma, \sigma'\}$ for $\sigma = [\mathtt{X} \mapsto \mathtt{0}]$ and $\sigma' = [\mathtt{X} \mapsto \mathtt{s(X')}]$. Then, $\mathit{DN}(t_1|_{1.2}) = \mathit{DN}(\sigma(t_1|_{1.2}))@\sigma \ \cup \ \mathit{DN}(\sigma'(t_1|_{1.2}))@\sigma'$, and we recursively call to $\mathit{DN}(\mathtt{min(0,0-0)})$ and $\mathit{DN}(\mathtt{min(s(X'),s(X')-0)})$. Now, the reader can check that $\mathit{DN}(\mathtt{min(0,0-0)}) = \{\ \langle \Lambda, id\rangle\ \}$, since term `min(0,0−0)` matches lhs l_5, and $\mathit{DN}(\mathtt{min(s(X'),s(X')-0)}) = 2.\mathit{DN}(\mathtt{s(X')-0}) = \{\ \langle 2, id\rangle\ \}$, since term `s(X')−0` matches lhs l_3. Hence, we can conclude $\mathit{DN}(t_1|_{1.2}) = \{\langle \Lambda, id\rangle\}@\sigma \cup \{\langle 2, id\rangle\}@\sigma' = \{\ \langle \Lambda, [\mathtt{X} \mapsto \mathtt{0}]\rangle, \langle 2, [\mathtt{X} \mapsto \mathtt{s(X')}]\rangle\ \}$.

Now consider $\mathit{DN}(t_1|_1) = \mathit{DN}(\mathtt{10!\ \%\ min(X,X-0)})$. Since it is not a redex, we have $\mathit{DN}(t_1|_1) = \cup_{q \in SP(t_1|_1) \setminus \{\Lambda\}} q.\mathit{DN}(t_1|_{1.q}) \ \cup \ \cup_{\sigma \in \mathit{DSub}(t_1|_1)} \mathit{DN}(\sigma(t_1|_1))@\sigma$

and $SP(t_1|_1) = D^{\uparrow}_{t_1|_1}(FP(t_1|_1))$. By Example 5, $FP(t_1|_1) = \{\Lambda, 2\}$, and then $SP(t_1|_1) = D^{\uparrow}_{t_1|_1}(\{\Lambda, 2\}) = \{\Lambda, 2\}$. Moreover, we have $DSub(t_1|_1) = \varnothing$, since no position in $FP(t_1|_1)$ is rooted by a variable. Thus, we have $DN(t_1|_1) = 2.DN(t_1|_{1.2})$. However, $DN(t_1|_{1.2})$ was already computed before, and we obtain $DN(t_1|_1) = \{\ \langle 2, [\mathtt{X} \mapsto \mathtt{0}]\rangle, \langle 2.2, [\mathtt{X} \mapsto \mathtt{s(X')}]\rangle\ \}$.

Finally, consider $DN(t_1|_2) = DN(\mathtt{10!}\ \%\ \mathtt{0})$. Since it is not a redex, we have $DN(t_1|_2) = \cup_{q \in SP(t_1|_2)\setminus\{\Lambda\}} q.DN(t_1|_{2.q})\ \cup\ \cup_{\sigma \in DSub(t_1|_2)} DN(\sigma(t_1|_2))@\sigma$. But by Example 4, we have $l_1 \blacktriangleleft t_1|_2$, $l_2 \blacktriangleleft t_1|_2$, and $DP_l(t_1|_2) = \{\Lambda\}$ for any other lhs l. Thus, we can conclude $DN(t_1|_2) = \varnothing$. Finally, putting everything together we have $DN(t_1) = \{\ \langle 1.2, [\mathtt{X} \mapsto \mathtt{0}]\rangle, \langle 1.2.2, [\mathtt{X} \mapsto \mathtt{s(X')}]\rangle\ \}$. Note these narroxes correspond to the optimal narrowing sequences in Example 1.

We are now ready to state the correctness and completeness properties of our generalized natural narrowing strategy. The key fact used in establishing these properties is the correspondence between generalized natural rewriting and generalized natural narrowing, as stated by the following lemma.

Lemma 3 (Completeness w.r.t. rewriting). *For a normalized substitution σ, if $\sigma(t) \overset{\mathtt{m}}{\rightarrow}_{\langle p, l\to r\rangle} s$, then there are η, θ, s' such that $t \overset{\mathtt{m}}{\rightsquigarrow}_{\langle p, l\to r, \theta\rangle} s'$, $\sigma|_{\mathcal{V}ar(t)} = (\eta \circ \theta)|_{\mathcal{V}ar(t)}$, $s = \eta(s')$, and η is normalized.*

Using Lemmas 2 and 3 we get the following correctness and completeness results for generalized natural narrowing.

Theorem 1 (Correctness). *If t is not a variable and is a $\overset{\mathtt{m}}{\rightsquigarrow}$ -normal form, then for every normalized σ we have $\sigma(t)$ is a head-normal form.*

Theorem 2 (Completeness). *If $\sigma(t) \to^* s$ and σ is a normalized substitution, then there are s', θ, θ' s.t. $t \overset{\mathtt{m}}{\rightsquigarrow}^*_{\theta} s'$, $\theta'(s') \xrightarrow{>\Lambda}{}^* s$, and $\sigma|_{\mathcal{V}ar(t)} = (\theta' \circ \theta)|_{\mathcal{V}ar(t)}$.*

The following example shows that our completeness result needs not hold for non-normalized substitutions.

Example 9. Consider the rewrite system from [29] with rules: (i) $\mathtt{f(b,c)} \to \mathtt{d}$, (ii) $\mathtt{a} \to \mathtt{b}$, and (iii) $\mathtt{a} \to \mathtt{c}$. For the term $t = \mathtt{f(X,X)}$ and substitution $\sigma = [\mathtt{X} \mapsto \mathtt{a}]$ we have $\sigma(t) \to^* \mathtt{d}$. But the generalized natural narrowing strategy cannot compute the normal form `d`. Specifically, positions 1 and 2 in t are demanded by rule (i), and the variable `X` is instantiated with substitutions $[\mathtt{X} \mapsto \mathtt{b}]$ and $[\mathtt{X} \mapsto \mathtt{c}]$, which are inferred from rule (i). However, both `f(b,b)` and `f(c,c)` are failing w.r.t. the left-hand side of rule (i). Thus $DN(t) = \varnothing$, and the strategy does not narrow the term `f(X,X)` any further.

4 Application Areas

In this section, we show how narrowing can be used as a unified mechanism for programming and proving, and explain informally how the generalized natural narrowing strategy makes the specific applications more efficient. Our purpose in this section is motivational. More applications, details and examples are given in [13].

4.1 Symbolic Model Checking

Narrowing can be used as a technique for symbolic reachability analysis of concurrent systems with an infinite state space. Specifically, a concurrent system can naturally be expressed as a rewrite system $\mathcal{R} = (\Sigma, \phi, R)$, where terms represent states, and a rewrite rule $t \to t'$ is understood as a (parametric) local transition [27,28]. We can then formalize a reachability problem for a concurrent system thus specified as solving an existential formula $(\exists \overrightarrow{x})\ t(\overrightarrow{x}) \to^* t'(\overrightarrow{x})$ where the *source* $t(\overrightarrow{x})$ is a term with variables representing a possibly infinite set of *initial* states (namely all its instances by *ground substitutions*) and the *target* $t'(\overrightarrow{x})$ represents a likewise possibly infinite set of *final* states that we want to reach by a sequence of transitions. *Solutions* to this reachability problem can then be described by substitutions σ for which indeed we have, $\sigma(t(\overrightarrow{x})) \to^* \sigma(t'(\overrightarrow{x}))$. More generally, we may consider conjunctive reachability goals of the form $G = (\exists \overrightarrow{x})\ t_1(\overrightarrow{x}) \to^* t'_1(\overrightarrow{x})\ \wedge \ldots \wedge\ t_n(\overrightarrow{x}) \to^* t'_n(\overrightarrow{x})$.

We can reduce solving a conjunctive reachability goal such as G, to the case of solving a *single* reachability goal by means of a theory transformation associating to a rewrite system $\mathcal{R} = (\Sigma, \phi, R)$, a corresponding rewrite system $\hat{\mathcal{R}} = (\hat{\Sigma}, \hat{\phi}, \hat{R})$, where $\hat{\Sigma} = \Sigma \cup \{\wedge, \twoheadrightarrow, \texttt{True}\}$, $\hat{\phi}$ extends ϕ with $\hat{\phi}(\twoheadrightarrow) = \{2\}$, and $\hat{R} = R \cup \{x \twoheadrightarrow x \to \texttt{True},\ \texttt{True} \wedge \texttt{True} \to \texttt{True}\}$. Note that the second argument of $\twoheadrightarrow$ is frozen, since only the sources of a goal are to be rewritten. Then, σ is a solution of the conjunctive goal G in the rewrite system R if and only if σ is a solution of the *single* reachability goal $(\exists \overrightarrow{x})\hat{G} \to^* \texttt{True}$ in the rewrite system $\hat{R}$, where $\hat{G}$ is the $\hat{\Sigma}$-term $\hat{G} = t_1 \twoheadrightarrow t'_1 \wedge \ldots \wedge t_n \twoheadrightarrow t'_n$.

Now, since the term True in the transformed theory $\hat{\mathcal{R}}$ is a head normal form, any head normalizing strategy $\mathcal{S}$ [30], including our efficient generalized natural narrowing strategy, gives us a semi-decision procedure to find all the normalized solutions of a goal G. Specifically, the algorithm incrementally builds the narrowing tree starting from the term $\hat{G}$, and searches, using $\mathcal{S}$, for narrowing derivations that result in True. When one such derivation is found, the composition of substitutions accumulated during the narrowing derivation in the reverse order gives us a solution of the goal G. Of course, the narrowing tree generated by $\mathcal{S}$ has to be explored in a *fair* manner, since the narrowing derivations can be infinitely long. The reader is referred to [29] for further details and an example where the above technique is applied for analysis of safety properties of security protocols.

Further, note that since the set of initial and final states specified in a goal can be infinite, and likewise there is no restriction on the number of reachable states, one can view the above narrowing procedure as a new form of "symbolic model checking" for infinite state systems.

4.2 Theorem Proving

Equational unification: Narrowing was originally introduced as a complete method for generating all solutions of an equational unification problem, i.e., for goals F of the form $(\exists \overrightarrow{x})\ t_1(\overrightarrow{x}) = t'_1(\overrightarrow{x}) \wedge \ldots \wedge t_n(\overrightarrow{x}) = t'_n(\overrightarrow{x})$ in free algebras modulo a set of equations that can be described by a set of confluent and terminating rewrite rules [15,23,24]. We note that the problem of solving reachability goals in rewrite systems generalizes the problem of equational unification. Specifically, suppose we are to solve the equational goal F above in the equational theory $\mathcal{E} = (\Sigma, E)$ where the equations E are confluent and terminating. Note that σ is a solution of F if and only if both $\sigma(t_i)$ and $\sigma(t'_i)$ can be reduced by the (oriented) equations E to a common term. We can thus consider the rewrite system $\mathcal{R}_{\mathcal{E}} = (\tilde{\Sigma}, \phi, R_E)$, where $\tilde{\Sigma} = \Sigma \cup \{\approx, \mathtt{True}\}$, $\phi(f) = \varnothing$ for all $f \in \tilde{\Sigma}$, and $R_E = E \cup \{x \approx x \to \mathtt{True}\}$. Then σ is a solution of the system of equations F in the equational theory $\mathcal{E}$ if and only if it is a solution of the reachability goal $G = (\exists \overrightarrow{x})\ t_1 \approx t'_1 \to^* \mathtt{True} \wedge \ldots \wedge t_n \approx t'_n \to^* \mathtt{True}$ in the rewrite system $\mathcal{R}_{\mathcal{E}}$.

Inductive theorem proving: The just-described reduction of existential equality goals to reachability goals, when combined with the reduction described in Section 4.1 of conjunctive reachability goals to a single goal has important applications to *inductive theorem proving*. Specifically, it is useful in proving existentially quantified inductive theorems like $E \vdash_{ind} (\exists \overrightarrow{x})\ t = t'$ in the initial model defined by the equations E. Natural narrowing can, using the two reductions just mentioned, provide a very efficient semi-decision procedure (and even a decision procedure for some restricted theories [31]) for proving such inductive goals because it will *detect failures* to unify, stopping with a counterexample instead of blindly expanding the narrowing tree. Furthermore, natural narrowing can make *inductionless induction* provers, particularly in the most recent formulations in [8,9], more effective and efficient, and can be used in such provers to prove also universal inductive theorems like $E \vdash_{ind} (\forall \overrightarrow{x})\ t = t'$ (or, more generally, clauses). The point is that natural narrowing's complete narrowing strategy can be used instead of the unrestricted narrowing carried out by *superposition* to compute a smaller set of deductions, which can increase the chances of termination of the inductionless induction procedure without loss of soundness. The extended version of this work [13] illustrates the previous point with an example where inductionless induction is able to prove an inductive theorem with natural narrowing, but not with unrestricted narrowing.

5 Related work

Lazy rewriting strategies are based on the original *strongly needed reduction* strategy of Huet and Levy [22]. This strategy is optimal (computes only needed steps), correct, and complete for *strongly sequential rewrite systems* (SS), a subclass of orthogonal rewrite systems. Note that this strategy does not apply to Example 1, since orthogonality implies left-linearity. Sekar and Ramakrishnan proposed the *parallel needed reduction* strategy [32] as an extension of Huet and Levy's strongly needed reduction to make it correct and complete for a larger rewrite system class, though still optimal for the former class. This larger rewrite system class is *constructor weakly orthogonal systems* (CB-WO), and thus it is not applicable to Example 1. Antoy proposed the *outermost-needed* *rewriting* [1] as an efficient implementation of strongly needed reduction for *inductively sequential rewrite systems* (IS), which are equivalent to SS's when instantiated to left-linear constructor systems [21]. In [1], Antoy also provides the *weakly outermost-needed rewriting* to make outermost-needed rewriting correct and complete for CB-WO's. Thus, both are not applicable to Example 1.

The continuous interest in the unification of functional and logic programming in a seamless way attracted much attention (see [20] for a survey) and Antoy, Echahed and Hanus extended Antoy's outermost-needed rewriting strategy to the *needed narrowing* strategy [4], becoming the best narrowing strategy for functional logic programming languages over other narrowing strategies such as [17,25,19] (see [4] for a detailed comparison). Antoy, Echahed, and Hanus extended their needed narrowing strategy to the *weakly needed narrowing strategy* [3] in order to cope with CB-WO's. Note that these strategies are not applicable to Example 1.

In recent work [11], we have proposed refinements of (weakly) outermost-needed rewriting and (weakly) needed narrowing, called *natural rewriting* and *natural narrowing*. These strategies compute less (or exactly the same) steps than outermost-needed rewriting and needed narrowing for IS's. However, they are not applicable to Example 1 because of left-linearity and constructor conditions.

This work is part of a broader joint effort to generalize narrowing from equational logic to rewriting logic, so as to make possible a much wider range of applications. It builds on our previous work extending natural rewriting to general term rewriting systems [14], and also on work by Meseguer and Thati on narrowing for rewrite theories [29,34]. As shown in [29,34], for general rewrite theories which need not be confluent and need not be terminating, the issue of *completeness*, that is, of narrowing being a complete semi-decision procedure for solving reachability goals, is nontrivial and does not always hold, essentially because rewrites can take place in the substitutions. The paper [29] characterizes several classes of rewrite theories for which narrowing is complete. This paper offers a narrowing strategy, which could be the basis of [29,34], to make narrowing efficient, and proves that this strategy is sound and complete in a different sense of "completeness," namely that any solution found by unrestricted narrowing (outside substitutions) will also be reachable using the strategy.

6 Conclusions and Future Work

We have generalized the narrowing strategy of [11] so that it can be applied to a much broader class of term rewrite systems. Specifically, the generalization drops the requirement that the rewrite system under consideration is left-linear and constructor, which is a typical assumption in functional (logic) programming. As a result of this generality, a much broader range of applications such as, symbolic reachability analysis of concurrent systems, and theorem proving, can be efficiently supported by our strategy. Since our generalization is conservative, we inherit all the optimality results presented in [11] for the class of left-linear constructor systems; note that the strategies in [11] are the best known for the class of left-linear constructor systems. An important problem for future research is to investigate optimality results of the generalized strategies for a larger class of rewrite systems.

Another interesting issue is to further generalize the strategies to the case where rewriting and narrowing are performed *modulo* a set of axioms (such as associativity, commutativity, and identity), as in languages such as ELAN [6] and Maude [7]. Specifically we are interested in strategies for rewrite theories of the form $\mathcal{R} = (\Sigma, \phi, E, R)$ where E is a set of axioms. A generalized narrowing strategy for such rewrite theories, that computes substitutions in an *incremental* manner, would have a very important efficiency advantage, since it will not explicitly use unification algorithms for the axioms E.

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 ALP'92*, LNCS 632:143–157. Springer, 1992.
2. S. Antoy. Constructor-based conditional narrowing. In *Proc. of PPDP'01*, pages 199–206. ACM, 2001.
3. S. Antoy, R. Echahed, and M. Hanus. Parallel evaluation strategies for functional logic languages. In *Proc. of 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*, 47(4):776–822, 2000.
5. S. Antoy and S. Lucas. Demandness in rewriting and narrowing. In *Proc. of WFLP'02*, ENTCS 76. Elsevier, 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. H. Comon. Inductionless induction. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, 1:913–962. Elsevier, 2001.
9. H. Comon and R. Nieuwenhuis. Induction = I-Axiomatization + First-Order Consistency. *Information and Computation*, 159(1/2):151–186, 2000.
10. A. Deursen, J. Heering, and P. Klint. *Language Prototyping: An Algebraic Specification Approach.* World Scientific, 1996.
11. S. Escobar. Refining weakly outermost-needed rewriting and narrowing. In *Proc. of PPDP'03*, pages 113–123. ACM, 2003.
12. S. Escobar. Implementing natural rewriting and narrowing efficiently. In *Proc. of FLOPS 2004*, LNCS 2998:147–162. Springer, 2004.
13. 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 Politécnica de Valencia, 2004. Available at `http://www.dsic.upv.es/users/elp/papers.html`.

14. S. Escobar, J. Meseguer, and P. Thati. Natural rewriting for general term rewriting systems. In *Proc. of LOPSTR'04*, 2004. To appear.
15. M. Fay. First order unification in equational theories. In *Proc. of CADE-04*, LNCS 87:161–167. Springer, 1979.
16. K. Futatsugi and R. Diaconescu. *CafeOBJ Report.* World Scientific, AMAST Series, 1998.
17. 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.
18. 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.
19. 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.
20. M. Hanus. The integration of functions into logic programming: From theory to practice. *Journal of Logic Programming*, 19&20:583–628, 1994.
21. M. Hanus, S. Lucas, and A. Middeldorp. Strongly sequential and inductively sequential term rewriting systems. *Information Processing Letters*, 67(1):1–8, 1998.
22. 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. MIT Press, 1992.
23. J. Hullot. Canonical forms and unification. In *Proc. of CADE-05*, LNCS 87:318–334. Springer, 1980.
24. J.-P. Jouannaud, C. Kirchner, and H. Kirchner. Incremental construction of unification algorithms in equational theories. In *Proc. of ICALP'83*, LNCS 154:361–373. Springer, 1983.
25. R. Loogen, F. López-Fraguas, and M. Rodríguez-Artalejo. A Demand Driven Computation Strategy for Lazy Narrowing. In *Proc. of PLILP'93*, LNCS 714:184–200. Springer, 1993.
26. S. Lucas. Context-sensitive rewriting strategies. *Information and Computation*, 178(1):294–343, 2002.
27. N. Martí-Oliet and J. Meseguer. Rewriting logic as a logical and semantic framework. In *Handbook of Philosophical Logic*, 9:1–88. Kluwer, 2001.
28. J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.
29. J. Meseguer and P. Thati. Symbolic reachability analysis using narrowing and its application to analysis of cryptographic protocols. In *Proc. of WRLA'04*, ENTCS 117:153–182. Elsevier, 2004.
30. A. Middeldorp. Call by Need Computations to Root-Stable Form. In *Proc. of POPL'97*, pages 94–105. ACM, 1997.
31. R. Nieuwenhuis. Basic paramodulation and decidable theories. In *Proc. of LICS'96*, pages 473–483. IEEE Computer Society, 1996.
32. R. Sekar and I. Ramakrishnan. Programming in equational logic: Beyond strong sequentiality. *Information and Computation*, 104(1):78–109, 1993.
33. TeReSe, editor. *Term Rewriting Systems.* Cambridge University Press, Cambridge, 2003.
34. P. Thati and J. Meseguer. Complete symbolic reachability analysis using back-and-forth narrowing. 2005. Submitted for publication.