MAX-PLANCK-INSTITUT FÜR INFORMATIK

A Needed Narrowing Strategy

Sergio Antoy
Rachid Echahed
Michael Hanus

MPI–I–93–243 November 1993

Im Stadtwald
D 66123 Saarbrücken
Germany

Authors' Addresses

Sergio Antoy

Department of Computer Science
Portland State University
Portland, OR 97207
U.S.A.
`antoy@cs.pdx.edu`

Rachid Echahed

IMAG-LGI
CNRS
F-38041 Grenoble
France
`echahed@imag.fr`

Michael Hanus

Max-Planck-Institut für Informatik
Im Stadtwald
D-66123 Saarbrücken
Germany
`michael@mpi-sb.mpg.de`

Publication Notes

A short version of this report is to appear in the *Proceedings of the 21st ACM Symposium on Principles of Programming Languages*, pages 268–279, Portland, Oregon, January 1994.

Acknowledgements

We would like to acknowledge the support of The Oregon Center for Advanced Technology Education (OCATE) for parts of the collaborative efforts that lead to the writing of this paper.

The research of Michael Hanus was supported in part by the German Ministry for Research and Technology (BMFT) under grant ITS 9103 and by the ESPRIT Basic Research Working Group 6028 (Construction of Computational Logics).

Abstract

Narrowing is the operational principle of languages that integrate functional and logic programming. We propose a notion of a needed narrowing step that, for inductively sequential rewrite systems, extends the Huet and Lévy notion of a needed reduction step. We define a strategy, based on this notion, that computes only needed narrowing steps. Our strategy is sound and complete for a large class of rewrite systems, is optimal w.r.t. the cost measure that counts the number of distinct steps of a derivation, computes only independent unifiers, and is efficiently implemented by pattern matching.

Keywords

Functional Programming, Logic Programming, Functional Logic Programming, Narrowing, Lazy Evaluation, Needed Reduction

1 Introduction

In recent years, most proposals with a sound and complete operational semantics for the integration of functional and logic programming languages [4, 10] were based on narrowing, e.g., [5, 15, 17, 19, 37, 44]. Narrowing, originally introduced in automated theorem proving [46], *solves* equations by computing unifiers with respect to an equational theory [14]. Informally, narrowing unifies a term with the left-hand side of a rewrite rule and fires the rule on the instantiated term.

Example 1 Consider the following rewrite rules defining the operations "less than or equal to" and addition for natural numbers, which are represented by terms built with 0 and s:

$$\begin{array}{rll} 0 \leq X & \rightarrow \mathit{true} & \mathrm{R}_1 \\ s(X) \leq 0 & \rightarrow \mathit{false} & \mathrm{R}_2 \\ s(X) \leq s(Y) & \rightarrow X \leq Y & \mathrm{R}_3 \end{array} \qquad \begin{array}{rll} 0 + X & \rightarrow X & \mathrm{R}_4 \\ s(X) + Y & \rightarrow s(X+Y) & \mathrm{R}_5 \end{array}$$

The rules of "$\leq$" will be used in following examples. To narrow the equation $Z+s(0) \approx s(s(0))$, rule R_5 is applied by instantiating Z to $s(X)$. To narrow the resulting equation, $s(X+s(0)) \approx s(s(0))$, R_4 is applied by instantiating X to 0. The resulting equation, $s(s(0)) \approx s(s(0))$, is trivially true. Thus, $\{Z \mapsto s(0)\}$ is the equation's solution.

A brute-force approach to finding all the solutions of an equation would attempt to unify *each* rule with *each* non-variable subterm of the given equation. The resulting search space would be huge even for small rewrite programs. Therefore, many narrowing strategies for limiting the size of the search space have been proposed, e.g., basic [25], innermost [15], outermost [12], outer [49], lazy [9, 36, 44], or narrowing with redundancy tests [6, 31]. Each strategy demands certain conditions of the rewrite relation to ensure the completeness of narrowing (the ability to compute all the solutions of an equation.)

Our contribution is a strategy that, for *inductively sequential* systems [1], preserves the completeness of narrowing and performs only steps that are "unavoidable" for solving equations. This characterization leads to the optimality of our strategy with respect to the number of "distinct" steps of a derivation. Advantages of our strategy over existing ones include: the large class of rewrite systems to which it is applicable, both the optimality of the derivations and the independence of the unifiers it computes, and the ease of its implementation.

The notion of an unavoidable step is well-known for rewriting. *Orthogonal* systems have the property that in every term t not in normal form there exists a redex, called *needed*, that must "eventually" be reduced to compute the normal form of t [24, 30, 39]. Furthermore, repeated rewriting of needed redexes in a term suffices to compute its normal form, if it exists. Loosely speaking, only needed redexes really matter for rewriting in orthogonal systems. We extend this fact to narrowing in inductively sequential systems, a subclass of the orthogonal systems.

Restricting our discussion to this subclass is not a limitation for the use of narrowing in programming languages. Computing a needed redex in a term is an unsolvable problem. *Strongly sequential* systems are, in practice, the largest class for which the problem becomes solvable. Inductively sequential systems are a large constructor-based subclass of the strongly sequential systems.

After some preliminaries in Section 2, we present our strategy in Section 3. We formulate the soundness and completeness results in Section 4. We address our strategy's optimality in Section 5. We compare related work in Section 6. Our conclusion is in Section 7.

2 Preliminaries

We recall some key notions and notations about rewriting. We are consistent with the conventions of [11, 29]. First of all, we fix the notations for terms.

Definition 1 A many-sorted *signature* Σ is a pair (S, Ω) where S is a set of *sorts* and Ω is a family of *operation* sets of the form $\Omega = (\Omega_{w,s} | w \in S^*, s \in S)$. Let $\mathcal{X} = (\mathcal{X}_s | s \in S)$ be an S-sorted, countably infinite set of *variables*. Then the set $\mathcal{T}(\Sigma, \mathcal{X})_s$ of *terms* of sort s built from Σ and $\mathcal{X}$ is the smallest set containing $\mathcal{X}_s$ such that $f(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma, \mathcal{X})_s$ whenever $f \in \Omega_{(s_1,\ldots,s_n),s}$ and $t_i \in \mathcal{T}(\Sigma, \mathcal{X})_{s_i}$. If $f \in \Omega_{\epsilon,s}$, we write f instead of $f()$. $\mathcal{T}(\Sigma, \mathcal{X})$ denotes the set of all terms. The set of variables occurring in a term t is denoted by $\mathcal{V}ar(t)$. A term t is called *ground term* if $\mathcal{V}ar(t) = \emptyset$. A term is called *linear* if it does not contain multiple occurrences of one variable. In the following Σ stands for a many-sorted signature.

In practice, most equational logic programs are *constructor-based*, i.e., symbols, called *constructors*, that construct data terms are distinguished from those, called *defined functions* or *operations*, that operate on data terms (see, for instance, the Equational Interpreter [40] and the functional logic languages *ALF* [19], *BABEL* [37], *K-LEAF* [16], *LPG* [5], *SLOG* [15]). Hence we define:

Definition 2 A many-sorted signature Σ is *constructor-based* iff the set of operations Ω is partitioned into two disjoint sets $\mathcal{C}$ and $\mathcal{D}$. $\mathcal{C}$ is the set of *constructors* and $\mathcal{D}$ is the set of *defined operations.* The terms in $\mathcal{T}(\mathcal{C}, \mathcal{X})$ are called *constructor terms.* A term $f(t_1, \ldots, t_n)$ $(n \geq 0)$ is called *innermost* if $f \in \mathcal{D}$ and $t_1, \ldots, t_n$ are constructor terms. A *constructor-based term rewriting system* $\mathcal{R}$ is a set of *rewrite rules*, $l \to r$, such that l and r have the same sort, l is innermost, and $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$.

In the rest of this paper we assume that $\mathcal{R}$ is a *constructor-based term rewriting system.* Substitutions are essential to the notions of rewriting and narrowing.

Definition 3 A *substitution* is a mapping $\sigma\colon \mathcal{X} \to \mathcal{T}(\Sigma, \mathcal{X})$ with $\sigma(x) \in \mathcal{T}(\Sigma, \mathcal{X})_s$ for all variables $x \in X_s$ such that its *domain* $\mathcal{D}om(\sigma) = \{x \in \mathcal{X} \mid \sigma(x) \neq x\}$ is finite. We frequently identify a substitution σ with the set $\{x \mapsto \sigma(x) \mid x \in \mathcal{D}om(\sigma)\}$. Substitutions are extended to morphisms on $\mathcal{T}(\Sigma, \mathcal{X})$ by $\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$ for every term $f(t_1, \ldots, t_n)$. A substitution σ is called *constructor substitution* if $\sigma(x)$ is a constructor term for all $x \in \mathcal{D}om(\sigma)$. The *composition of two substitutions* σ and τ is defined by $(\sigma \circ \tau)(x) = \sigma(\tau(x))$ for all $x \in \mathcal{X}$. The *restriction* $\sigma_{|V}$ of a substitution σ to a set V of variables is defined by $\sigma_{|V}(x) = \sigma(x)$ if $x \in V$ and $\sigma_{|V}(x) = x$ if $x \notin V$. A substitution σ is *more general* than σ', denoted by $\sigma \leq \sigma'$, if there is a substitution τ with $\sigma' = \tau \circ \sigma$. If V is a set of variables, we write $\sigma = \sigma'[V]$ iff $\sigma_{|V} = \sigma'_{|V}$, and we write $\sigma \leq \sigma'[V]$ iff there is a substitution τ with $\sigma' = \tau \circ \sigma[V]$. Two substitutions σ and σ' are *independent* on a set of variables V iff there exists some $x \in V$ such that $\sigma(x)$ and $\sigma'(x)$ are not unifiable.

A term t' is an *instance* of t if there is a substitution σ with $t' = \sigma(t)$. In this case we write $t \leq t'$. A term t' is a *variant* of t if $t \leq t'$ and $t' \leq t$.

A *unifier* of two terms s and t is a substitution σ with $\sigma(s) = \sigma(t)$. A unifier σ is called *most general* (*mgu*) if $\sigma \leq \sigma'$ for every other unifier σ'. Most general unifiers are unique up to variable renaming. By introducing a total ordering on variables we can uniquely choose *the* most general unifier of two terms. Hence we denote by $mgu(s, t)$ the most general unifier of s and t.

We use in our proofs that a unifier is an idempotent substitution and that any variable in the domain of a unifier is already contained in one of the terms being unified. Positions, too, are essential to the notions of rewriting and narrowing.

Definition 4 An *occurrence* or *position* is a sequence of positive integers identifying a subterm in a term. For every term t, the empty sequence, denoted by Λ, identifies t itself. For every term of the form $f(t_1, \ldots, t_k)$, the sequence $i \cdot p$, where i is a positive integer not greater than k and p is a position, identifies the subterm of t_i at p. The subterm of t at p is denoted by $t|_p$ and the result of *replacing* $t|_p$ with s in t is denoted by $t[s]_p$. If p and q are positions, we write $p \leq q$ if p is *above* or is a *prefix* of q, and we write $p \parallel q$ if the positions are *disjoint* (see [11] for details). The expression $p \cdot q$ denotes the position resulting from the concatenation of the positions p and q, i.e., we overload the symbol "$\cdot$".

We are now ready to define rewriting.

Definition 5 A *rewrite step* is an application of a rewrite rule to a term, i.e., $t \rightarrow_{p,R} s$ if there exist a position p, a rewrite rule $R = l \rightarrow r$ and a substitution σ with $t|_p = \sigma(l)$ and $s = t[\sigma(r)]_p$. In this case we say t is *rewritten* (at position p) *to* s and $t|_p$ is a *redex* of t. We will omit the subscripts p and R if they are clear from the context. A redex $t|_p$ of t is an *outermost redex* if there is no redex $t|_q$ of t with $q < p$. $\stackrel{*}{\rightarrow}$ denotes the transitive and reflexive closure of $\rightarrow$. $\stackrel{*}{\leftrightarrow}$ denotes the symmetric closure of $\stackrel{*}{\rightarrow}$. A term t is *reducible to* a term s if $t \stackrel{*}{\rightarrow} s$. A term t is called *irreducible* or in *normal form* if there is no term s with $t \rightarrow s$. A term s is a *normal form of* t if t is reducible to the irreducible term s.

Rewriting is computing, i.e., the *value* of a functional expression is its normal form obtained by rewriting. Functional logic programs compute with partial information, i.e., a functional expression may contain logical variables. The goal is to compute values for these variables such that the expression is evaluable to a particular normal form, e.g., a constructor term [16, 37]. This is done by narrowing.

Definition 6 A term t is *narrowable* to a term s if there exist a non-variable position p in t (i.e., $t|_p \notin \mathcal{X}$), a variant $l \rightarrow r$ of a rewrite rule in $\mathcal{R}$ with $\mathcal{V}ar(t) \cap \mathcal{V}ar(l \rightarrow r) = \emptyset$ and a unifier σ of $t|_p$ and l such that $s = \sigma(t[r]_p)$. In this case we write $t \rightsquigarrow_{p,l\rightarrow r,\sigma} s$. If σ is a most general unifier of $t|_p$ and l, the narrowing step is called *most general.* We write $t_0 \stackrel{*}{\rightsquigarrow}_\sigma t_n$ if there is a narrowing sequence $t_0 \rightsquigarrow_{p_1,R_1,\sigma_1} t_1 \rightsquigarrow_{p_2,R_2,\sigma_2} \cdots \rightsquigarrow_{p_n,R_n,\sigma_n} t_n$ with $\sigma = \sigma_n \circ \cdots \circ \sigma_2 \circ \sigma_1$.

Since the instantiation of the variables in the rule $l \rightarrow r$ by σ is not relevant for the computed result of a narrowing derivation, we will omit this part of σ in the example derivations in this paper.

Example 2 Referring to Example 1,

$$A + B \rightsquigarrow_{\Lambda, \mathrm{R}_5, \{A \mapsto s(0), B \mapsto 0\}} s(0+0)$$

and

$$A + B \rightsquigarrow_{\Lambda, \mathrm{R}_5, \{A \mapsto s(X)\}} s(X+B)$$

are narrowing steps of $A + B$, but only the latter is a most general narrowing step.

Padawitz [42] too distinguishes between narrowing and most general narrowing, but in most papers narrowing is intended as most general narrowing (e.g., [25]). Most general narrowing has the advantage that most general unifiers are uniquely computable, whereas there exist many independent unifiers. Dropping the requirement that unifiers be most general is crucial to the definition of needed narrowing step, since these steps may be impossible with most general unifiers.

Narrowing solves equations, i.e., computes values for the variables in an equation such that the equation becomes true, where an *equation* is a pair $t \approx t'$ of terms of the same sort. Since we do not require terminating term rewriting systems, normal forms may not exist. Hence, we define the validity of an equation as a strict equality on terms in the spirit of functional logic languages with a lazy operational semantics such as *K-LEAF* [16] and *BABEL* [37].

Definition 7 An *equation* is a pair $t \approx t'$ of terms of the same sort. A substitution σ is a *solution* for an equation $t \approx t'$ iff $\sigma(t)$ and $\sigma(t')$ are reducible to a same ground constructor term.

Our definition of solution is weaker than convertibility, i.e., $\sigma(t) \overset{*}{\leftrightarrow} \sigma(t')$. This is due to the fact that we are discussing constructor-based, not necessarily terminating rewrite systems.

Equations can also be interpreted as terms by defining the symbol $\approx$ as a binary operation symbol, more precisely, one operation symbol for each sort. Therefore all notions for terms, such as substitution, rewriting, narrowing etc., will also be used for equations. The semantics of $\approx$ is defined by the following rules, where $\wedge$ is assumed to be a right-associative infix symbol, and c is a constructor of arity 0 in the first rule and arity $n > 0$ in the second rule.

$$\begin{array}{rcl} c \approx c & \to & \mathit{true} \\ c(X_1, \ldots, X_n) \approx c(Y_1, \ldots, Y_n) & \to & (X_1 \approx Y_1) \wedge \cdots \wedge (X_n \approx Y_n) \\ \mathit{true} \wedge X & \to & X \end{array}$$

These are the *equality rules* of a signature. It is easy to see that the orthogonality status of a rewrite system (see below) is not changed by these rules. The same holds true for the inductive sequentiality, which will be defined shortly. With these rules a solution of an equation is computed by narrowing it to *true*—an approach also taken in *K-LEAF* [16] and *BABEL* [37]. The equivalence between the reducibility to a same ground constructor term and the reducibility to *true* using the equality rules is addressed by Proposition 1.

We also require orthogonality, which ensures the good-behavior of computations.

Definition 8 A term rewriting system $\mathcal{R}$ is *orthogonal* if for each rule $l \to r \in \mathcal{R}$ the left-hand side l is linear (*left-linearity*) and for each non-variable subterm $l|_p$ of l there exists no rule $l' \to r' \in \mathcal{R}$ such that $l|_p$ and l' unify (*non-overlapping*).

Our strategy extends to narrowing the rewriting notion of *need*. The idea, for rewriting, is to reduce in a term only certain redexes which *must* be reduced to compute the normal form of t. In orthogonal term rewriting systems, every term not in normal form has a redex that must be reduced to compute the term's normal form. The following definition [24] formalizes this idea.

Definition 9 Let $A = t \to_{u,l\to r} t'$ be a rewrite step of some term t into t' at position u with rule $l \to r$. The set of *descendants* (or *residuals*) of a position v by A, denoted $v \setminus A$, is

$$v \setminus A = \begin{cases} \emptyset & \text{if } u = v, \\ \{v\} & \text{if } u \not\leq v, \\ \{up'q \ \text{ such that } \ r|_{p'} = x\} & \text{if } v = upq \text{ and } l|_p = x, \text{ where } x \text{ is a variable.} \end{cases}$$

The set of *descendants* of a position v by a rewrite derivation B is defined by induction as follows

$$v \setminus B = \begin{cases} \{v\} & \text{if } B \text{ is the null derivation,} \\ \displaystyle\bigcup_{w \in v \setminus B'} w \setminus B'' & \text{if } B = B'B'', \text{ where } B' \text{ is the initial step of } B. \end{cases}$$

A position u of a term t is called *needed* iff in every rewrite derivation of t to a normal form a descendant of $t|_u$ is rewritten at its root.

A position uniquely identifies a subterm of a term. The notion of *descendant* for terms stems directly from the corresponding notion for positions.

A more intuitive definition of descendant of a position or term is proposed in [30]. Let $t \xrightarrow{*} t'$ be a reduction sequence and s a subterm of t. The descendants of s in t' are computed as follows: Underline the root of s and perform the reduction sequence $t \xrightarrow{*} t'$. Then, every subterm of t' with an underlined root is a *descendant* of s.

Example 3 Consider the operation that doubles its argument by means of an addition. The rules of addition are in Example 1.

$$double(X) \to X + X \qquad \mathrm{R}_6$$

In the following reduction of $double(0+0)$ we show, by means of underlining, the descendants of $0+0$.

$$double(0 \underline{+} 0) \to_{\Lambda, \mathrm{R}_6} (0 \underline{+} 0) + (0 \underline{+} 0)$$

The set of descendants of position 1 by the above reduction is $\{1, 2\}$.

3 Outermost-needed narrowing

An efficient narrowing strategy must limit the search space. No suitable rule can be ignored, but some positions in a term may be neglected without losing completeness. For instance, Hullot [25] has introduced *basic narrowing*, where narrowing is not applied at positions introduced by substitutions, Fribourg [15] has proposed *innermost narrowing*, where narrowing is applied only at an innermost position, and Hölldobler [22] has combined innermost and basic narrowing. Narrowing only at *outermost* positions is complete only if the rewrite system satisfies strong restrictions such as non-unifiability of subterms of the left-hand sides of rewrite rules [12]. *Lazy narrowing* [9, 36, 44], akin to lazy evaluation in functional languages, attempts to avoid unnecessary evaluations of expressions. A lazy narrowing step is applied at outermost positions with the exception that inner arguments of a function are evaluated, by narrowing them to their head normal forms, if their values are required for an outermost narrowing step. Unfortunately, the property “required” depends on the rules tried in following steps, and looking-ahead is not a viable option.

We want to perform only narrowing steps that are necessary for computing solutions. Naively, one could say that a narrowing step $t \leadsto_{p, l \to r, \sigma} t'$ is *needed* iff p is a position of t, σ is the most general unifier of $t|_p$ and l, and $\sigma(t|_p)$ is a needed redex. Unfortunately, a substantial complication arises from this simple approach. If t' is a normal form, the step is trivially needed. However, some instantiation performed later in the derivation could “undo” this need.

Example 4 Referring to Example 1, consider the term $t = X \leq Y + Z$. According to the naive approach, the following narrowing step of t at position 2

$$X \leq Y + Z \leadsto_{2, \mathrm{R}_4, \{Y \mapsto 0\}} X \leq Z$$

would be needed, since $X \leq Z$ is a normal form. This step is indeed necessary to solve the inequality if $s(x)$, for some term x, is eventually substituted for X, although this claim may not be obvious without the results presented in this note. However, the same step becomes unnecessary if 0 is substituted for X, as shown by the following derivation, which computes a more general solution of the inequation without ever narrowing any descendant of t at 2.

$$X \leq Y + Z \leadsto_{\Lambda, \mathrm{R}_1, \{X \mapsto 0\}} true$$

Thus, in our definition, we impose a condition strong enough to ensure the necessity of a narrowing step, no matter which unifiers might be used later in the derivation.

Definition 10 A narrowing step $t \rightsquigarrow_{p,R,\sigma} t'$ is called *needed* or *outermost-needed* iff, for every $\eta \geq \sigma$, p is the position of a needed or outermost-needed redex of $\eta(t)$, respectively. A narrowing derivation is called *needed* or *outermost-needed* iff every step of the derivation is needed or outermost-needed, respectively.

Our definition adds, with respect to rewriting, a new dimension to the difficulty of computing needed narrowing steps. We must take into account any instantiation of a term in addition to any derivation to normal form. Luckily, as for rewriting, the problem has an efficient solution in inductively sequential systems. We forgo the requirement that the unifier of a narrowing step be most general. The instantiation that we demand in addition to that for the most general unification ensures the need of the position irrespective of future unifiers. It turns out that this extra instantiation would eventually be performed later in the derivation. Thus we are only "anticipating" it, and the completeness of narrowing is preserved. This approach, however, complicates the notion of narrowing strategy.

According to [12, 42], a narrowing strategy is a function from terms into non-variable positions in these terms so that exactly one position is selected for the next narrowing step. Unfortunately, this notion of narrowing strategy is inadequate for narrowing with arbitrary unifiers, which, as Example 4 shows, are essential to capture the need of a narrowing step.

Definition 11 A *narrowing strategy* is a function from terms into sets of triples. If $\mathcal{S}$ is a narrowing strategy, t is a term, and $(p, l \to r, \sigma) \in \mathcal{S}(t)$, then p is a position of t, $l \to r$ is a rewrite rule, and σ a substitution such that $t \rightsquigarrow_{p,l\to r,\sigma} \sigma(t[r]_p)$ is a narrowing step.

We now define a class of rewrite systems for which there exists an efficiently computable needed narrowing strategy. Systems in this class have the property that the rules defining any operation can be organized in a hierarchical structure called definitional tree [1], which is used to implement needed rewriting. This note generalizes that result to narrowing.

The symbols *branch*, *rule*, and *exempt*, used in the next definition, are uninterpreted functions used to classify the nodes of the tree. A *pattern* is an innermost term contained in each node.

Definition 12 $\mathcal{T}$ is a *partial definitional tree*, or *pdt*, with pattern π w.r.t. a constructor-based rewrite system $\mathcal{R}$ iff one of the following cases holds:

$\mathcal{T} = \mathit{branch}(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k)$, where π is a pattern, o is the occurrence of a variable of π, the sort of $\pi|_o$ has constructors $c_1, \ldots, c_k$, for some $k > 0$, and for all i in $\{1, \ldots, k\}$, $\mathcal{T}_i$ is a *pdt* with pattern $\pi[c_i(X_1, \ldots, X_n)]_o$, where n is the arity of c_i and $X_1, \ldots, X_n$ are new variables.

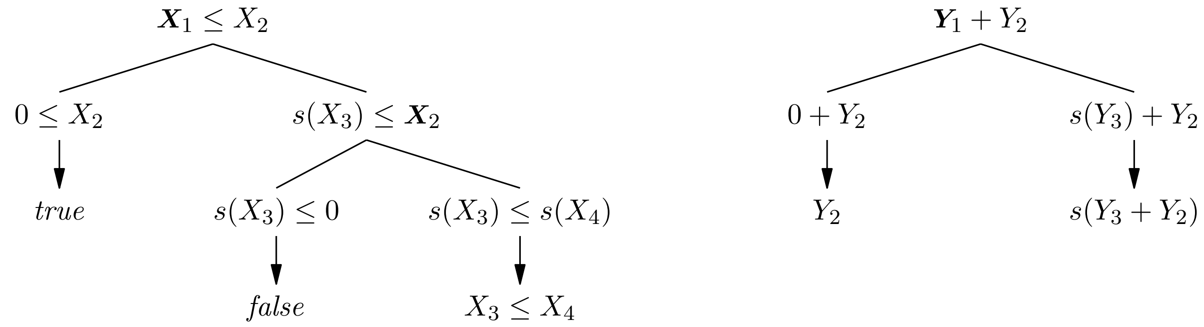
$\mathcal{T} = \mathit{rule}(\pi, l \to r)$, where π is a pattern and $l \to r$ is a rewrite rule in $\mathcal{R}$ such that $l = \pi$.

$\mathcal{T} = \mathit{exempt}(\pi)$, where π is a pattern and $l \not\leq \pi$ for every rule $l \to r$ in $\mathcal{R}$.

$\mathcal{T}$ is a *definitional tree* of an operation f iff $\mathcal{T}$ is a *pdt* with $f(X_1, \ldots, X_n)$ as the pattern argument, where n is the arity of f and $X_1, \ldots, X_n$ are new variables.

We call *inductively sequential* an operation f of a rewrite system $\mathcal{R}$ iff there exists a definitional tree $\mathcal{T}$ of f such that the rules contained in $\mathcal{T}$ are all and only the rules defining f in $\mathcal{R}$. We call *inductively sequential* a rewrite system $\mathcal{R}$ iff any operation of $\mathcal{R}$ is inductively sequential.

Example 5 We show a pictorial representations of definitional trees of the operations defined in Example 1. A branch node of the picture shows the pattern of a corresponding node of the definitional tree. A leaf node of the picture shows the right sides of a rule contained in a rule node of the tree. The occurrence argument of a branch node is shown by emboldening the corresponding subterm in the pattern argument.



Inductively sequential systems are constructor-based and strongly sequential [1]. We conjecture that these two classes are the same. Inductively sequential systems model the first-order functional component of programming languages, such as *ML* and *Haskell*, that establish priorities among rules by textual precedence or specificity [28]. We now give an informal account of our strategy.

The patterns of a definitional tree are a finite set partially ordered by the subsumption preordering and complete in the sense of [23]. Let $t = f(t_1, \ldots, t_k)$ be a term to narrow. We unify t with some maximal element of the set of patterns of a definitional tree of f. Let π denote such a pattern, τ the most general unifier of t and π, and $\mathcal{T}$ the *pdt* in which π is contained. If $\mathcal{T}$ is a *rule pdt*, then we narrow $\tau(t)$ at the root with the rule contained in $\mathcal{T}$. If $\mathcal{T}$ is an *exempt pdt*, then $\tau(t)$ cannot be narrowed to a constructor-rooted term. If $\mathcal{T}$ is a *branch pdt*, then we recur on $\tau(t|_o)$, where o is the occurrence contained in $\mathcal{T}$ and τ is the *anticipated* substitution. The result of the recursive invocation is suitably composed with τ and o. The details of this composition are in the formal definition presented below.

We derive our outermost-needed strategy from a mapping, λ, that implements the above computation. λ takes two arguments, an operation-rooted term t and a definitional tree $\mathcal{T}$ of the root of t, and non-deterministically returns a triple, (p, R, σ), where p is a position of t, R is either a rule $l \to r$ of $\mathcal{R}$ or the distinguished symbol "?", and σ is a substitution. If $R = l \to r$, then our strategy performs the narrowing step $t \rightsquigarrow_{p, l \to r, \sigma} \sigma(t[r]_p)$. If $R = ?$, then our strategy gives up, since it is impossible to narrow t to a constructor-rooted term.

In the following definition, $\mathit{pattern}(\mathcal{T})$ denotes the pattern argument of $\mathcal{T}$.

Definition 13 The function λ takes two arguments, an operation-rooted term t and a *pdt* $\mathcal{T}$ such that $\mathit{pattern}(\mathcal{T})$ and t unify. The function λ yields a set of triples of the form (p, R, σ), where p is a position of t, R is either a rewrite rule or the distinguished symbol "?", and σ is a unifier of $\mathit{pattern}(\mathcal{T})$ and t. Thus, let t be a term and $\mathcal{T}$ a *pdt* in the domain of λ. The function λ is defined by strong arithmetical induction on the number of occurrences of operation symbols in t with a nested structural induction on $\mathcal{T}$ as follows.

$$\lambda(t,\mathcal{T}) \ni \begin{cases} (\Lambda, R, mgu(t,\pi)) & \text{if } \mathcal{T} = rule(\pi, R); \\ (\Lambda, ?, mgu(t,\pi)) & \text{if } \mathcal{T} = exempt(\pi); \\ (p, R, \sigma) & \text{if } \mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k), \\ & \quad t \text{ and } pattern(\mathcal{T}_i) \text{ unify, for some } i, \text{ and} \\ & \quad (p, R, \sigma) \in \lambda(t, \mathcal{T}_i); \\ (o \cdot p, R, \sigma \circ \tau) & \text{if } \mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k), \\ & \quad t \text{ and } pattern(\mathcal{T}_i) \text{ do not unify, for any } i, \\ & \quad \tau = mgu(t,\pi), \\ & \quad \mathcal{T}' \text{ is a definitional tree of the root of } \tau(t|_o), \text{ and} \\ & \quad (p, R, \sigma) \in \lambda(\tau(t|_o), \mathcal{T}'). \end{cases}$$

The function λ is well-defined in the third case since, by the definition of *pdt*, there exists a proper sub*pdt* $\mathcal{T}_i$ of $\mathcal{T}$ such that $pattern(\mathcal{T}_i)$ and t unify if $t|_o$ is constructor-rooted or a variable. Similarly, λ is well-defined in the fourth case since this case can only occur if $t|_o$ is operation-rooted. In this case $\tau_{|\mathcal{V}ar(t)}$ is a constructor substitution since π is a linear innermost term. Since t is operation-rooted and $o \neq \Lambda$, $\tau(t|_o)$ has fewer occurrences of operation symbols than t. Since $t|_o$ is operation-rooted, so is $\tau(t|_o)$. By the definition of *pdt*, $pattern(\mathcal{T}') \leq \tau(t|_o)$, i.e., $pattern(\mathcal{T}')$ and $\tau(t|_o)$ unify. This implies that λ is well-defined in this case too.

As in proof procedures for logic programming, we have to apply *variants* of the rewrite rules *with fresh variables* to the current term. Therefore, we assume in the following that the definitional trees contain new variables if they are used in a narrowing step.

The computation of $\lambda(t,\mathcal{T})$ may entail a non-deterministic choice when $\mathcal{T}$ is a *branch pdt*—the integer i when $t|_o$ is constructor-rooted or a variable. The substitution τ when $t|_o$ is operation-rooted is the *anticipated* substitution guaranteeing the need of the computed position. It is pushed down in the recursive call to λ to ensure the consistency of the computation when t is non-linear. The anticipated substitution is neglected when $t|_o$ is not operation-rooted, since the pattern in $\mathcal{T}_i$ is an instance of π. Hence, σ extends the anticipated substitution.

Example 6 We trace the computation of λ for the initial step of a derivation of $X \leq Y + Z$, which was discussed in Example 4.

$$\begin{array}{l} \lambda(X \leq Y + Z, branch(X_1 \leq X_2, 1, \ldots)) \\ \quad \lambda(X \leq Y + Z, branch(s(X_3) \leq X_2, 2, \ldots)) \\ \quad\quad \lambda(Y + Z, branch(Y_1 + Y_2, 1, \ldots)) \\ \quad\quad\quad \lambda(Y + Z, rule(0 + Y_2, \mathrm{R}_4)) \\ \quad\quad\quad (\Lambda, \mathrm{R}_4, \{Y \mapsto 0, Y_2 \mapsto Z\}) \\ \quad\quad (\Lambda, \mathrm{R}_4, \{Y \mapsto 0, Y_2 \mapsto Z\}) \\ \quad (2, \mathrm{R}_4, \{X \mapsto s(X_3), X_2 \mapsto 0 + Z, Y \mapsto 0, Y_2 \mapsto Z\}) \\ (2, \mathrm{R}_4, \{X \mapsto s(X_3), X_2 \mapsto 0 + Z, Y \mapsto 0, Y_2 \mapsto Z\}) \end{array}$$

We prove two simple technical lemmas concerning the mutual relationships between the patterns of the *pdt*s of a definitional tree.

Lemma 1 *Let $\mathcal{T}$ be a* pdt*, p and q two positions of $\mathcal{T}$, and π_p and π_q the patterns of the* pdt*s at the positions p and q of $\mathcal{T}$ respectively. If $p \leq q$, then $\pi_p \leq \pi_q$.*

Proof If $p \leq q$, then there exists a position r such that $q = p \cdot r$. The proof is by induction on r. Base case: $r = \Lambda$ implies $p = q$ and consequently $\pi_p = \pi_q$. Ind. case: $r = i \cdot r'$, for some positive integer i and position r'. Let $\mathcal{T}_p$ be the *pdt* of $\mathcal{T}$ at p. $\mathcal{T}_p = \mathit{branch}(\pi_p, o, \mathcal{T}_{p_1}, \ldots, \mathcal{T}_{p_k})$, for some position o, and *pdt*s $\mathcal{T}_{p_1}, \ldots, \mathcal{T}_{p_k}$, for some $k \geq i$. Let π_{p_i} be the pattern in $\mathcal{T}_{p_i}$. Since π_{p_i} is obtained by instantiating with a constructor term the variable of π_p at o, $\pi_p < \pi_{p_i}$. By construction, π_q is equal to the pattern of the *pdt* of $\mathcal{T}_{p_i}$ at r'. By the induction hypothesis, $\pi_{p_i} \leq \pi_q$. By the transitivity of "$\leq$", $\pi_p < \pi_q$. □

Lemma 2 *The patterns in the* pdt*s of two disjoint positions of a definitional tree* $\mathcal{T}$ *do not unify.*

Proof The proof is by structural induction on the *pdt* $\mathcal{T}$.

$\mathcal{T} = \mathit{rule}(\pi, R)$, for some pattern π and rule R, or $\mathcal{T} = \mathit{exempt}(\pi)$, for some pattern π.

There are no disjoint positions in $\mathcal{T}$ and the claim vacuously holds.

$\mathcal{T} = \mathit{branch}(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k)$, for some pattern π, position o, and *pdt*s $\mathcal{T}_1, \ldots, \mathcal{T}_k$, for some $k \geq 0$.

Let p and q be two disjoint positions in $\mathcal{T}$. Both p and q differ from Λ, hence there exist integers i and j in $\{1, \ldots k\}$, and positions p' and q' such that $p = i \cdot p'$ and $q = j \cdot q'$. If $i = j$, then p' and q' are disjoint positions of $\mathcal{T}_i$. The patterns of $\mathcal{T}$ at p and q are equal to the patterns of $\mathcal{T}_i$ at p' and q' respectively. The latter do not unify by the induction hypothesis. If $i \neq j$, then, by Lemma 1, the patterns of $\mathcal{T}$ at p and q are instances of the root patterns of $\mathcal{T}_i$ and $\mathcal{T}_j$ respectively. The latter do not unify, since have a different symbol at position o, thus the former do not unify either. □

We are interested only in narrowing derivations that end in a constructor term. Our key result is that if λ, on input of a term t, computes a position p and a substitution σ, and η extends σ, then $\eta(t)$ must "eventually" be narrowed at p to obtain a constructor term. "Eventually" is formalized by the notion of *descendant*, which, initially proposed for rewriting [24], is extended to narrowing simply by replacing $\rightarrow_{u, l \rightarrow r}$ with $\rightsquigarrow_{u, l \rightarrow r, \sigma}$ in Definition 9.

Theorem 1 *Let* $\mathcal{R}$ *be an inductively sequential rewrite system,* t *an operation-rooted term, and* $\mathcal{T}$ *a definitional tree of the root of* t*. Let* $(p, R, \sigma) \in \lambda(t, \mathcal{T})$ *and* η *extend* σ*, i.e.,* $\eta \geq \sigma$*.*

1. *In any narrowing derivation of* $\eta(t)$ *to a constructor-rooted term a descendant of* $\eta(t|_p)$ *is narrowed to a constructor-rooted term.*

2. *If* $R = l \rightarrow r$*, then* $t \rightsquigarrow_{p, R, \sigma} \sigma(t[r]_p)$ *is an outermost-needed narrowing step.*

3. *If* $R = ?$*, then* $\eta(t)$ *cannot be narrowed to a constructor-rooted term.*

Proof The proof is by arithmetical induction on the number of occurrences of operation symbols in t with a nested structural induction on the *pdt* $\mathcal{T}$. We consider the cases of the definition of λ.

$\mathcal{T} = \mathit{rule}(\pi, R')$, for some pattern π and rule R'.

In this case $(p, R, \sigma) = (\Lambda, R', \mathit{mgu}(t, \pi))$. Since $\eta(t)$ is operation-rooted and is a descendant of itself, claim number 1 trivially holds. Let $R' = l \rightarrow r$, for some terms l and r. By the definition of a definitional tree, $\pi = l$, which implies that $\sigma(l) = \sigma(t|_p)$. Thus, $t \rightsquigarrow_{p, R, \sigma} \sigma(t[r]_p)$ is a narrowing step. Since $\mathcal{R}$ is orthogonal, its redex schemes do not overlap, consequently, R keeps matching any descendant of $\eta(t)$ obtained by reductions strictly below Λ. Thus $\eta(t)$ is a needed redex of itself and it is obviously outermost. Claim number 3 vacuously holds.

$\mathcal{T} = exempt(\pi)$, for some pattern π.

In this case $(p, R, \sigma) = (\Lambda, ?, mgu(t, \pi))$. Since $\eta \geq \sigma$, η unifies π and t too. We could extend $\mathcal{R}$ by changing the *exempt* node into a *rule* node in which the left-hand side of the rule is obviously π and the right-hand side is arbitrary. Thus, similar to the previous case, π would keep unifying with any descendant of $\eta(t)$ obtained by reductions strictly below the root. Thus, by Lemma 2 there exists no rule in $\mathcal{R}$ that would unify with $\eta(t)$. Thus, $\eta(t)$ cannot be narrowed to a constructor-rooted term which implies claim number 3 and, trivially, claim number 1. Claim number 2 vacuously holds.

$\mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k)$, for some pattern π, position o, and *pdt*s $\mathcal{T}_1, \ldots, \mathcal{T}_k$, for some $k \geq 0$. We consider the two subcases of the definition of λ for *branch* nodes.

$t|_o$ is either constructor-rooted or is a variable.

By the definition of *pdt*, there exists some i in $\{1, \ldots k\}$ such that $pattern(\mathcal{T}_i)$ and t unify. By the definition of λ, $\lambda(t, \mathcal{T}) = \lambda(t, \mathcal{T}_i)$. By the induction hypothesis, all the claims hold already for $\lambda(t, \mathcal{T}_i)$ and they are independent of $\mathcal{T}_i$.

$t|_o$ is operation-rooted.

By the definition of λ, π and t unify. Let $\tau = mgu(t, \pi)$. Since $t|_o$ is operation-rooted, so is $\tau(t|_o)$. Let $\mathcal{T}'$ be a definitional tree of the root of $\tau(t|_o)$. Let $(p', R', \sigma') \in \lambda(\tau(t|_o), \mathcal{T}')$ such that $(p, R, \sigma) = (o \cdot p', R', \sigma' \circ \tau)$, where p' is a position of $t|_o$, R' is either a rule or "?", and σ' is a substitution. In this case $(t|_o)|_{p'} = t|_{o \cdot p'} = t|_p$. By Lemma 2, any rule whose left-hand side might unify with t is contained in a leaf of $\mathcal{T}_i$. If $l \to r$ is a rule contained in a leaf of $\mathcal{T}_i$, then, by Lemma 1, $pattern(\mathcal{T}_i) \leq l$. Thus, by the definition of definitional tree, l has a constructor symbol at position o. However, the case being considered assumes that t, has an operation symbol at position o. Hence in any narrowing derivation of $\eta(t)$ that includes a step at the root a descendant of $\eta(t|_o)$ must be narrowed to a constructor-rooted term. Since t is operation-rooted, also $\eta(t)$ is operation-rooted, and in any narrowing derivation of $\eta(t)$ to a constructor-rooted term a descendant of $\eta(t)$ is narrowed at the root, and, consequently, a descendant of $\eta(t|_o)$ is narrowed to a constructor-rooted term. By the definition of λ, $\tau(t|_o)$ has fewer occurrences of operation symbols than t. Thus, by the induction hypothesis on t, for any $\eta' \geq \sigma'$, in any narrowing derivation of $\eta'(\tau(t|_o))$ to a constructor-rooted term a descendant of $\eta'(\tau(t|_p))$ is narrowed to a constructor-rooted term. Since $\eta \geq \sigma$, $\eta = \phi \circ \sigma$ for some substitution ϕ. Let $\eta' = \phi \circ \sigma' \geq \sigma'$ which implies $\eta'(\tau(t)) = \eta(t)$ since $\sigma = \sigma' \circ \tau$. Hence in any narrowing derivation of $\eta(t|_o)$ to a constructor-rooted term a descendant of $\eta(t|_p)$ is narrowed to a constructor-rooted term. Thus, claim number 1 holds by transitivity.

We consider the two cases for R'.

R' is a rule.

By the induction hypothesis on t, $\tau(t|_o) \leadsto_{p', R, \sigma'} \sigma'(\tau(t|_o)[r]_{p'})$ is an outermost-needed narrowing step, hence, $t \leadsto_{p, R, \sigma} \sigma(t[r]_p)$ is a narrowing step. The need of $\eta(t|_p)$ with respect to $\eta(t)$ is an immediate consequence of claim number 1. By the definition of λ, π and t unify, and by the definition of definitional tree, π is an innermost term and o is a position of π. These conditions imply that there is only one operation symbol in $\sigma(t)$ above o, the root of $\sigma(t)$. In constructor-based systems, redexes occur only at positions of operation symbols. We have proved above that $\eta(t)$ is not a redex. Thus, there are no redexes in $\eta(t)$ above o and, by the induction hypothesis on t, the redex $\eta(t|_p)$ is outermost in $\eta(t)$ too. Claim number 3 vacuously holds.

$R' = ?$.

We have proved above that in any narrowing derivation of $\eta(t)$ to a constructor-rooted term a descendant of $\eta(t|_o)$ is narrowed to a constructor-rooted term. By the induction hypothesis on t, for any $\eta' \geq \sigma'$, $\eta'(\tau(t|_o))$ cannot be narrowed to a constructor-rooted term. Since $\eta \geq \sigma$, $\eta = \phi \circ \sigma$ for some substitution ϕ. Let $\eta' = \phi \circ \sigma' \geq \sigma'$ which implies $\eta = \eta' \circ \tau$ since $\sigma = \sigma' \circ \tau$. Thus $\eta(t|_o) = \eta'(\tau(t|_o))$ cannot be narrowed to a constructor-rooted term. Hence $\eta(t)$ cannot be narrowed to a constructor-rooted term. Claim number 2 vacuously holds. $\square$

We say that a narrowing derivation is *computed by* λ iff for each step $t \rightsquigarrow_{p,R,\sigma} t'$ of the derivation, (p, R, σ) belongs to $\lambda(t, \mathcal{T})$. The function λ implements our narrowing strategy as discussed next. The theorem shows (claim 2) that our strategy λ computes only outermost-needed narrowing steps. The theorem, however, does not show that the computation succeeds, i.e., a narrowing step is computed for any operation-rooted, hence expectedly narrowable, term. This requirement may seem essential, since to narrow a term "all the way" a strategy should compute a narrowing step, when one exists. Indeed, in incomplete rewrite systems, λ may fail to compute any narrowing step even when some step could be computed.

Example 7 Consider an incompletely defined operation, f, taking and returning a natural number.

$$f(0) \to 0$$

The term $t = f(s(f(0)))$ can be narrowed (actually rewritten, since it is ground) to its normal form, $f(s(0))$. The only redex position of t is $1 \cdot 1$, but λ returns the set $\{(1, ?, \{\})\}$.

The inability of λ to compute certain outermost-needed narrowing steps is a blessing in disguise. The theorem (claim 3) justifies giving up a narrowing attempt as soon as the failure to find a rule occurs—without further attempts to narrow t at other positions with the hope that a different rule might be found after other narrowing steps or that the position might be *deleted* [7] by another narrowing step. If $(p, ?, \sigma) \in \lambda(t, \mathcal{T})$, no equation having $\sigma(t)$ as one side can be solved. Any amount of work applied toward finding a solution would be wasted. This is an opportunity for optimization. In fact $\sigma(t)$ may be narrowable at other positions different from p and an equation with $\sigma(t)$ as a side may even have an infinite search space. However, any amount of work applied toward finding a solution would be wasted.

Example 8 Consider the following term rewriting system for subtraction:

$$\begin{array}{rll} X - 0 \to X & & \mathrm{R}_1 \\ s(X) - s(Y) \to X - Y & & \mathrm{R}_2 \end{array}$$

This term rewriting system is inductively sequential and a definitional tree, $\mathcal{T}$, of the operation "$-$" has an *exempt* node for the pattern $0 - s(X)$, i.e., the system is incomplete and $(\Lambda, ?, \{\}) \in \lambda(0 - s(X), \mathcal{T})$. Therefore we can immediately stop the needed narrowing derivation of the equation $0 - s(X) \approx Y - Z$ while there would be infinitely many narrowing derivations for the right-hand side of this equation.

The definition of our outermost-needed narrowing strategy does not determine the computation space for a given inductively sequential rewrite system in a unique way. The concrete strategy depends on the definitional trees, and there is some freedom to construct these. For a discussion on how to compute definitional trees from rewrite rules and the implications of some non-deterministic choices of this computation see [1]. As we will show in Section 5, this does not affect the optimality of our strategy w.r.t. computed solutions. But in case of failing derivations a definitional tree which is "unnecessarily large" could result in unnecessary derivation steps.

E.g., a minimal definitional tree of the operation "$-$" in Example 8 has an *exempt* node for the pattern $0 - s(X)$. However, Definition 12 also allows a definitional tree with a *branch* node for the pattern $0 - s(X)$ which has *exempt* nodes for the patterns $0 - s(0)$ and $0 - s(s(X_1))$. Our strategy would perform some unnecessary steps if this definitional tree were used for narrowing the term $0 - s(t)$, where t is an operation-rooted term. These unnecessary steps can be avoided if all *branch* nodes in a definitional tree are useful, i.e., there is at least one *rule* node in each *branch* sub*pdt*.

However, the non-determinism of the trees of certain operations makes it possible that some work may be wasted when a narrowing derivation computed by λ terminates with a non-constructor term. The problem seems inevitable and is due to the inherent parallelism of certain operations, such as $\approx$; this issue is discussed in some depth in [1, Display (8)]. The problem occurs only in terms with two or more outermost-needed narrowing positions, one of which cannot be narrowed to a constructor-rooted term.

4 Soundness and completeness

Outermost-needed narrowing is a sound and complete procedure to solve equations if we add the equality rules to narrow equations to *true*. The following proposition shows the equivalence between the reducibility to a same ground constructor term and the reducibility to *true* using the equality rules.

Proposition 1 *Let $\mathcal{R}$ be a term rewriting system without rules for $\approx$ and $\wedge$. Let $\mathcal{R}'$ be the system obtained by adding the equality rules to $\mathcal{R}$. The following propositions are equivalent for all terms t and t':*

1. *t and t' are reducible in $\mathcal{R}$ to a same ground constructor term.*
2. *$t \approx t'$ is reducible in $\mathcal{R}'$ to 'true'.*

Proof To show that claim 1 implies claim 2, consider a ground constructor term u such that $t \overset{*}{\to} u$ and $t' \overset{*}{\to} u$ using rules from $\mathcal{R}$. Hence $t \approx t' \overset{*}{\to} u \approx u$ using rules from $\mathcal{R}'$. To show claim 2, it is sufficient to show $u \approx u \overset{*}{\to} \mathit{true}$ using the equality rules. This is done by induction on the structure of u. Base case: If u is a 0-ary constructor, say c, then $u \approx u$ can be directly reduced to *true* using the equality rule $c \approx c \to \mathit{true}$. Induction step: Let $u = c(t_1, \ldots, t_n)$. Then

$$u \approx u \;\to\; (t_1 \approx t_1) \wedge \cdots \wedge (t_n \approx t_n)$$

is a rewrite step using the equality rule for the n-ary constructor c. By the induction hypothesis, $t_i \approx t_i \overset{*}{\to} \mathit{true}$ using the equality rules ($i = 1, \ldots, n$). Moreover, $\mathit{true} \wedge \cdots \wedge \mathit{true}$ can be reduced to *true* using the equality rule for $\wedge$.

To show that claim 2 implies claim 1, consider a reduction sequence $t \approx t' \overset{*}{\to} \mathit{true}$ using rules from $\mathcal{R}'$. We show the existence of a ground constructor term u such that $t \overset{*}{\to} u$ and $t' \overset{*}{\to} u$ using rules from $\mathcal{R}$ by induction on the number, say k, of $\approx$-rule applications in this reduction sequence. Base case ($k = 1$): There is exactly one application of a $\approx$-rule:

$$t \approx t' \;\overset{*}{\to}\; s \approx s' \;\to\; r \;\overset{*}{\to}\; \mathit{true}$$

r cannot have the symbol $\wedge$ at the root, otherwise there must be further applications of a $\approx$-rule in the derivation $r \overset{*}{\to} true$. Hence the applied $\approx$-rule is of the form $c \approx c \to true$ which implies claim 1. Induction step ($k > 1$): Then there is a first application of a $\approx$-rule:

$$t \approx t' \ \overset{*}{\to} \ s \approx s' \ \to \ r \ \overset{*}{\to} \ true$$

$r \neq true$, otherwise there are no further applications of a $\approx$-rule in $r \overset{*}{\to} true$. Therefore $s = c(t_1, \ldots, t_n)$, $s' = c(t'_1, \ldots, t'_n)$, and $r = (t_1 \approx t'_1) \wedge \cdots \wedge (t_n \approx t'_n)$. Since $r \overset{*}{\to} true$, an $\wedge$-rule must be applied to the root in this sequence, i.e., $r \overset{*}{\to} true \wedge r' \overset{*}{\to} true$. Thus $t_1 \approx t'_1 \overset{*}{\to} true$ with at most $k-1$ $\approx$-rule applications. By the induction hypothesis, there is a ground constructor term u_1 such that $t_1 \overset{*}{\to} u_1$ and $t'_1 \overset{*}{\to} u_1$ using rules from $\mathcal{R}$. By a further induction on the arguments t_i, t'_i we can show the existence of ground constructor terms $u_1, \ldots, u_n$ such that $t_i \overset{*}{\to} u_i$ and $t'_i \overset{*}{\to} u_i$ using rules from $\mathcal{R}$. Altogether, we obtain the derivations

$$\begin{aligned} t &\overset{*}{\to} c(t_1, \ldots, t_n) \overset{*}{\to} c(u_1, \ldots, u_n) \\ t' &\overset{*}{\to} c(t'_1, \ldots, t'_n) \overset{*}{\to} c(u'_1, \ldots, u'_n) \end{aligned}$$

using rules from $\mathcal{R}$. This implies claim 1. □

The soundness of outermost-needed narrowing is easy to prove, since outermost-needed narrowing is a special case of general narrowing.

Theorem 2 (Soundness of outermost-needed narrowing) *Let $\mathcal{R}$ be an inductively sequential rewrite system extended by the equality rules. If $t \approx t' \overset{*}{\rightsquigarrow}_\sigma true$ is an outermost-needed narrowing derivation, then σ is a solution for $t \approx t'$.*

Proof If $t \approx t' \overset{*}{\rightsquigarrow}_\sigma true$, there exists a derivation

$$t \approx t' \rightsquigarrow_{p_1, R_1, \sigma_1} t_1 \rightsquigarrow_{p_2, R_2, \sigma_2} \cdots \rightsquigarrow_{p_n, R_n, \sigma_n} t_n$$

such that $t_n = true$ and $\sigma = \sigma_n \circ \cdots \circ \sigma_1$. By induction on the number n of narrowing steps it is easy to prove that $\sigma(t \approx t') \overset{*}{\to} true$. By Proposition 1, this implies that $\sigma(t)$ and $\sigma(t')$ are reducible to a same ground constructor term without using the equality rules. By Definition 7, σ is a solution for $t \approx t'$. □

In order to prove completeness of the outermost-needed narrowing strategy we lift the completeness result for the corresponding rewrite strategy [1] to narrowing derivations. For this purpose we recall the definition of the outermost-needed rewrite strategy for inductively sequential systems. Similarly to λ, this rewrite strategy is implemented by a function, φ, that takes two arguments, an operation-rooted term, t, and a definitional tree, $\mathcal{T}$, of the root of t (the definition of φ is a slightly modified version of the definition given in [1] extended to non-ground terms). Throughout an interleaved descent down both t and $\mathcal{T}$, φ computes a position p and, whenever possible, a rule R such that the rewriting of t at p by means of R is outermost-needed. We recall that computing needed reductions is unsolvable in the general case of sequential systems, and might be expensive in the case of strongly sequential systems. In practice, the computation of our function is efficiently performed by pattern matching.

Definition 14 The function φ takes two arguments, an operation-rooted term t and a *pdt* $\mathcal{T}$ such that $pattern(\mathcal{T}) \leq t$. The function φ yields a pair, (p, R), where p is a position of t and R is either a rewrite rule or the distinguished symbol "?". Thus, let t be a term and $\mathcal{T}$ be a *pdt* in the domain of φ. The function φ is defined by structural induction on t with a nested structural induction on $\mathcal{T}$ as follows.

$$\varphi(t,\mathcal{T}) = \begin{cases} (\Lambda, R) & \text{if } \mathcal{T} = \mathit{rule}(\pi, R); \\ (\Lambda, ?) & \text{if } \mathcal{T} = \mathit{exempt}(\pi); \\ \varphi(t, \mathcal{T}_i) & \text{if } \mathcal{T} = \mathit{branch}(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k) \text{ and } \mathit{pattern}(\mathcal{T}_i) \leq t, \text{ for some } i; \\ (o \cdot p, R) & \text{if } \mathcal{T} = \mathit{branch}(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k), \\ & \quad t|_o \text{ is operation-rooted,} \\ & \quad \mathcal{T}' \text{ is a definitional tree of the root of } t|_o, \text{ and} \\ & \quad \varphi(t|_o, \mathcal{T}') = (p, R). \\ (o, ?) & \text{if } \mathcal{T} = \mathit{branch}(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k) \text{ and } t|_o \text{ is a variable} \end{cases}$$

The function φ is well-defined in the third case since, by the definition of *pdt*, a proper sub*pdt* $\mathcal{T}_i$ of $\mathcal{T}$ with $\mathit{pattern}(\mathcal{T}_i) \leq t$ must uniquely exist iff $t|_o$ is constructor-rooted. Similarly, φ is well-defined in the fourth case since $t|_o$ is a proper subterm of t and $\mathit{pattern}(\mathcal{T}') \leq t|_o$ by the definition of *pdt*.

The following theorem shows that the function φ computes outermost-needed redexes. The proof parallels that of Theorem 1.

Theorem 3 *Let $\mathcal{R}$ be an inductively sequential rewrite system, t an operation-rooted term, $\mathcal{T}$ a definitional tree of the root of t, and $\varphi(t,\mathcal{T}) = (p, R)$.*

1. *In any reduction sequence of t to a constructor-rooted term a descendant of $t|_p$ is reduced to a constructor-rooted term.*
2. *If R is a rule of $\mathcal{R}$, then $t|_p$ is an outermost-needed redex of t matched by R.*
3. *If $R = ?$, then t cannot be reduced to a constructor-rooted term.*

Proof The proof is by arithmetical induction on the number of occurrences of operation symbols in t with a nested structural induction on the *pdt* $\mathcal{T}$. We consider the cases of the definition of φ.

$\mathcal{T} = \mathit{rule}(\pi, R')$, for some pattern π and rule R'.

In this case $(p, R) = (\Lambda, R')$. Since t is operation-rooted and is a descendant of itself, claim number 1 trivially holds. By the definition of φ, $\pi \leq t$. By the definition of definitional tree, R is a rule whose left-hand side is equal to π, Thus, t is a redex, it is matched by R, and it is obviously outermost. Since $\mathcal{R}$ is orthogonal, its redex schemes do not overlap, consequently, R keeps matching any descendant of t obtained by reductions strictly below Λ. Thus t is a needed redex. Claim number 3 vacuously holds.

$\mathcal{T} = \mathit{exempt}(\pi)$, for some pattern π.

In this case $(p, R) = (\Lambda, ?)$. By the definition of φ, $\pi \leq t$. We could extend $\mathcal{R}$ by changing the *exempt* node into a *rule* one in which the left-hand side of the rule is obviously π and the right-hand side is arbitrary. Thus, similar to the previous case, π keeps matching any descendant of t obtained by reductions strictly below the root. Thus, by Lemma 2 there exists no rule in $\mathcal{R}$ for a reduction of t at the root. Thus, t cannot be reduced to a constructor-rooted term which implies claim number 3 and, trivially, claim number 1. Claim number 2 vacuously holds.

$\mathcal{T} = \mathit{branch}(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k)$, for some pattern π, position o, and *pdt*s $\mathcal{T}_1, \ldots, \mathcal{T}_k$, for some $k \geq 0$.

We consider the three subcases of the definition of φ for *branch* nodes.

$t|_o$ is constructor-rooted.

By the definition of *pdt*, there exists some i in $\{1, \dots k\}$ such that $pattern(\mathcal{T}_i) \leq t$. By the definition of φ, $\varphi(t, \mathcal{T}) = \varphi(t, \mathcal{T}_i)$. By the induction hypothesis, all the claims hold already for $\varphi(t, \mathcal{T}_i)$ and they are independent of $\mathcal{T}_i$.

$t|_o$ is operation-rooted.

Let $\mathcal{T}'$ be a definitional tree of the root of $t|_o$. Let $\varphi(t|_o, \mathcal{T}') = (p', R')$, where p' is a position of $t|_o$ and R' is either a rule or "?". In this case $(p, R) = (o \cdot p', R')$ and $(t|_o)|_{p'} = t|_{o \cdot p'} = t|_p$. By the definition of φ, $\pi \leq t$. By Lemma 2 any rule that might reduce t at the root is contained in a leaf of $\mathcal{T}_i$. If $l \to r$ is a rule contained in a leaf of $\mathcal{T}_i$, then, by Lemma 1, $pattern(\mathcal{T}_i) \leq l$. Thus, by the definition of definitional tree, l has a constructor symbol at position o. However, the case being considered assumes that t does not have a constructor symbol at position o. Hence, in any reduction sequence of t that includes a reduction at the root a descendant of $t|_o$ must be reduced to a constructor-rooted term. Since t is operation-rooted, in any reduction sequence of t to a constructor-rooted term a descendant of t is reduced at the root, and, consequently, a descendant of $t|_o$ is reduced to a constructor-rooted term. By the induction hypothesis on t, in any reduction sequence of $t|_o$ to a constructor-rooted term a descendant of $t|_p$ is reduced to a constructor-rooted term. Thus, claim number 1 holds by transitivity.

We consider the two cases for R'.

R' is a rule.

By the induction hypothesis on t, $t|_p$ is an outermost-needed redex of $t|_o$ matched by R', hence, $t|_p$ is a redex of t matched by R'. The need of $t|_p$ with respect to t is an immediate consequence of claim number 1. By the definition of φ, $\pi \leq t$, and by the definition of definitional tree, π is an innermost term and o is a position of π. These conditions imply that there is only one operation symbol in t above o, the root of t. In constructor-based systems, redexes occur only at positions of operation symbols. We have just proved that t is not a redex. Thus, there are no redexes in t above o and, by the induction hypothesis on t, the redex $t|_p$ is outermost in t too.

$R' = ?$.

By claim number 1, in any reduction sequence of t to a constructor-rooted term a descendant of $t|_p$ is reduced to a constructor-rooted term. By the induction hypothesis in t, $t|_p$ cannot be reduced to a constructor-rooted term. Thus, t cannot be reduced to a constructor-rooted term.

$t|_o$ is a variable.

In this case $(p, R) = (o, ?)$. The proofs of claims number 1 and 3 are similar to those of the previous case, but do not require induction hypotheses. Claim number 2 vacuously holds. □

The following lemma shows the close ties between φ and λ, which are instrumental to lift outermost-needed reduction sequences to corresponding narrowing derivations. This will allow us to prove the completeness of the outermost-needed narrowing strategy.

Lemma 3 *Let $\mathcal{R}$ be an inductively sequential rewrite system. Let t be an operation-rooted term, $\mathcal{T}$ be a definitional tree of the root of t, and σ be a constructor substitution. If $\sigma(t) \to_{p,R} t'$ with $(p, R) = \varphi(\sigma(t), \mathcal{T})$, then there exists a substitution θ such that*

1. $(p, R, \theta) \in \lambda(t, \mathcal{T})$

2. $\theta \le \sigma[\mathcal{V}ar(t)]$

Proof The proof is by arithmetical induction on the number of occurrences of operation symbols in t with a nested structural induction on the *pdt* $\mathcal{T}$.

$\mathcal{T} = rule(\pi, R')$, for some pattern π and rule R'.

In this case $(p, R) = (\Lambda, R')$ and $\pi \le \sigma(t)$. This implies the existence of a substitution ϕ with $\phi(\pi) = \sigma(t)$. Hence π and t are unifiable (we assume that π and t are variable disjoint, otherwise take a new variant of the definitional tree) and there exists a most general unifier θ of π and t with $\theta \le \sigma[\mathcal{V}ar(t)]$. By the definition of λ, $(p, R, \theta) \in \lambda(t, \mathcal{T})$.

$\mathcal{T} = exempt(\pi)$: This case cannot occur since $R \ne ?$.

$\mathcal{T} = branch(\pi, o, \mathcal{T}_1, \dots, \mathcal{T}_k)$, for some pattern π, position o, and *pdt*s $\mathcal{T}_1, \dots, \mathcal{T}_k$, for some $k \ge 0$. We consider the three subcases of the definition of φ for *branch* nodes.

$\sigma(t)|_o$ is constructor-rooted.

By the definition of *pdt*, there exists some i in $\{1, \dots k\}$ such that $pattern(\mathcal{T}_i) \le \sigma(t)$. By the definition of φ, $\varphi(\sigma(t), \mathcal{T}) = \varphi(\sigma(t), \mathcal{T}_i)$. By the induction hypothesis, $(p, R, \theta) \in \lambda(\sigma(t), \mathcal{T}_i)$ and $\theta \le \sigma[\mathcal{V}ar(t)]$. By the definition of λ (note that $pattern(\mathcal{T}_i)$ and t unify), $(p, R, \theta) \in \lambda(t, \mathcal{T})$.

$\sigma(t)|_o$ is operation-rooted.

By the definition of φ, $\pi \le \sigma(t)$. $\sigma(t)$ and $pattern(\mathcal{T}_i)$ do not unify for each i in $\{1, \dots k\}$ since $\sigma(t)|_o$ is operation-rooted but $pattern(\mathcal{T}_i)$ has a constructor symbol at position o. Let $\mathcal{T}'$ be a definitional tree of the root of $\sigma(t)|_o$ and $\varphi(\sigma(t)|_o, \mathcal{T}') = (p', R')$. By the definition of φ, $(p, R) = (o \cdot p', R')$. Since $\pi \le \sigma(t)$, there exists the most general unifier τ of π and $\sigma(t)$ with $\tau \le \sigma[\mathcal{V}ar(t)]$ (we assume that π and t are variable disjoint, otherwise take a new variant of the definitional tree). $\tau_{|\mathcal{V}ar(t)}$ is a constructor substitution since π is a linear innermost term and t is operation-rooted. Let σ' be a constructor substitution such that $\sigma' \circ \tau = \sigma[\mathcal{V}ar(t)]$. Since σ is a constructor substitution, o is a position of t, and $t|_o$ is operation-rooted. Moreover, $\sigma(t) \to_{p,R} t'$ implies $\sigma'(\tau(t|_o)) \to_{p',R'} t'|_o$. Since o is different from the root position, t is operation-rooted, and $\tau_{|\mathcal{V}ar(t)}$ is a constructor substitution, $\tau(t|_o)$ has fewer occurrences of operation symbols than t. Hence we can apply the induction hypothesis to $\tau(t|_o)$ and σ' and we obtain a substitution θ' with $(p', R', \theta') \in \lambda(\tau(t|_o), \mathcal{T}')$ and $\theta' \le \sigma'[\mathcal{V}ar(\tau(t|_o))]$. By the definition of λ, $(o \cdot p', R', \theta' \circ \tau) \in \lambda(t, \mathcal{T})$, i.e., $(p, R, \theta' \circ \tau) \in \lambda(t, \mathcal{T})$. $\theta' \le \sigma'[\mathcal{V}ar(\tau(t|_o))]$ implies $\theta' \le \sigma'[\mathcal{V}ar(\tau(t))]$ since θ' instantiates only variables from $\tau(t|_o)$ and new variables of the definitional tree. Hence $\theta' \circ \tau \le \sigma' \circ \tau[\mathcal{V}ar(t)]$ which is equivalent to $\theta' \circ \tau \le \sigma[\mathcal{V}ar(t)]$.

$\sigma(t)|_o$ is a variable: This case cannot occur since $R \ne ?$. □

The following lemma shows how to lift an outermost-needed rewrite step to an outermost-needed narrowing step.

Lemma 4 *Let $\mathcal{R}$ be an inductively sequential rewrite system. Let σ be a constructor substitution, V be a finite set of variables, t be an operation-rooted term with $\mathcal{V}ar(t) \subseteq V$, and $\mathcal{T}$ be a definitional tree of the root of t. If $\sigma(t) \to_{p,R} s$ with $(p, R) = \varphi(\sigma(t), \mathcal{T})$, then there exist an outermost-needed narrowing step $t \leadsto_{p,R,\theta} t'$ and a constructor substitution σ' such that $(p, R, \theta) \in \lambda(t, \mathcal{T})$, $\sigma'(t') = s$ and $\sigma' \circ \theta = \sigma[V]$.*

Proof Let R be $l \to r$. Since $\sigma(t) \to_{p,R} s$, there exists a substitution ρ such that $\rho(l) = \sigma(t)|_p = \sigma(t|_p)$. Let $\phi = \rho \circ \sigma$ (we assume $\mathcal{D}om(\rho) \subseteq \mathcal{V}ar(R)$ and R is a rule with new variables not occurring in V and the image of σ, otherwise take an appropriate variant of R). By Lemma 3, there is a triple $(p, R, \theta) \in \lambda(t, \mathcal{T})$ with $\theta \leq \phi[\mathcal{V}ar(t)]$. Then there exists σ' such that $\sigma' \circ \theta = \phi[V]$ (w.l.o.g. we assume that $\theta(x) = x$ for all $x \in V - \mathcal{V}ar(t)$). This implies $\sigma' \circ \theta = \sigma[V]$ by definition of ϕ. By claim 2 of Theorem 1, $t \rightsquigarrow_{p,R,\theta} t'$ is an outermost-needed narrowing step. σ' is a constructor substitution since $\phi_{|V}$ is. Finally, $\sigma'(t') = \sigma'(\theta(t[r]_p)) = \phi(t[r]_p) = \sigma(t)[\rho(r)]_p = s$. □

Outermost-needed narrowing instantiates variables to constructor terms. Thus, we only show that outermost-needed narrowing is complete for constructor substitutions as solutions of equations. This is not a limitation in practice, since more general solutions would contain unevaluated or undefined expressions. This is not a limitation with respect to related work, since most general narrowing is known to be complete only for irreducible solutions [42], and lazy narrowing is complete only for constructor substitutions [16, 37]. The following theorem shows the completeness of our strategy, λ, and consequently of outermost-needed narrowing.

Theorem 4 (Completeness of outermost-needed narrowing) *Let $\mathcal{R}$ be an inductively sequential rewrite system extended by the equality rules. Let σ be a constructor substitution that is a solution of an equation $t \approx t'$ and V be a finite set of variables containing $\mathcal{V}ar(t) \cup \mathcal{V}ar(t')$. Then there exists a derivation $t \approx t' \overset{*}{\rightsquigarrow}_{\sigma'}$ true computed by λ such that $\sigma' \leq \sigma[V]$.*

Proof By Definition 7, there exists a ground constructor term, say u, such that $\sigma(t \approx t') \overset{*}{\to} u \approx u$. Since $\mathcal{R}$ is extended by the equality rules, $\sigma(t \approx t') \overset{*}{\to} \mathit{true}$ by Proposition 1. Consider the following rewriting sequence

$$s_0 \to_{p_1,R_1} s_1 \to_{p_2,R_2} s_2 \to_{p_3,R_3} \cdots$$

where $s_0 = \sigma(t \approx t')$, $(p_{i+1}, R_{i+1}) = \varphi(s_i, \mathcal{T}_i)$ and $\mathcal{T}_i$ is a definitional tree of the root of s_i for $i = 0, 1, 2, \ldots$. The following claims are easy to show by induction on the derivation steps in this sequence:

1. s_i has a constructor-rooted normal form (*true*).
2. If $s_i \neq \mathit{true}$, then the root of s_i is the operation symbol $\wedge$ or *true* (by the definition of equality rules).
3. If $s_i \neq \mathit{true}$, then $R_{i+1} \neq\ ?$ (claim 3 of Theorem 3) and $s_i|_{p_{i+1}}$ is an outermost-needed redex (claim 2 of Theorem 3).

Hence the derivation sequence is a well-defined outermost-needed rewriting derivation (as long as $s_i \neq \mathit{true}$). Since repeated rewriting of needed redexes in a term computes the term's normal form, if it exists [24], the sequence is finite and $s_n = \mathit{true}$ is the final term for some $n > 0$. We will show by induction on n that there exists a corresponding outermost-needed narrowing derivation

$$t \approx t' \rightsquigarrow_{p_1,R_1,\sigma_1} t_1 \rightsquigarrow_{p_2,R_2,\sigma_2} \cdots \rightsquigarrow_{p_n,R_n,\sigma_n} t_n$$

such that $t_n = \mathit{true}$ and $\sigma_n \circ \cdots \circ \sigma_1 \leq \sigma[V]$.

$n = 1$: If we apply Lemma 4 to the rewrite step $s_0 \to_{p_1,R_1} s_1$, we obtain an outermost-needed narrowing step $t \approx t' \rightsquigarrow_{p_1,R_1,\sigma_1} t_1$ and a constructor substitution σ' such that $\sigma' \circ \sigma_1 = \sigma[V]$ and $\sigma'(t_1) = s_1 = \mathit{true}$. Hence $\sigma_1 \leq \sigma[V]$ and $t_1 = \mathit{true}$ by the definition of equality rules.

$n > 1$: By Lemma 4 applied to the first rewrite step, there exist an outermost-needed narrowing step $t \approx t' \leadsto_{p_1, R_1, \sigma_1} t_1$ and a constructor substitution σ' such that $\sigma' \circ \sigma_1 = \sigma[V]$ and $\sigma'(t_1) = s_1$. Let $V_1 = \{y \in \mathcal{V}ar(\sigma_1(x)) \mid x \in V\}$. Applying the induction hypothesis to V_1, σ' and the derivation

$$s_1 \to_{p_2, R_2} \cdots \to_{p_n, R_n} s_n$$

yields an outermost-needed narrowing derivation

$$t_1 \leadsto_{p_2, R_2, \sigma_2} \cdots \leadsto_{p_n, R_n, \sigma_n} t_n$$

with $t_n = \mathit{true}$ and $\sigma_n \circ \cdots \circ \sigma_2 \leq \sigma'[V_1]$. Combining that with the first narrowing step, we obtain the required outermost-needed narrowing derivation with $\sigma_n \circ \cdots \circ \sigma_1 \leq \sigma[V]$ since $\sigma' \circ \sigma_1 = \sigma[V]$. □

The theorem justifies our earlier remark on the relationship between completeness and anticipated substitutions. Any anticipated substitution of a needed narrowing step is irrelevant or would eventually be done later in the derivation, and thus, it does not affect the completeness. Anticipating substitutions is appealing, even without the benefits related to the need of a step, since less general substitutions are likely to yield a smaller search space to compute the same set of solutions.

5 Optimality

In Section 3 we showed that our strategy computes only necessary steps. We now strengthen this characterization by showing that our strategy computes only necessary derivations of minimum length. First of all, we show that no redundant derivation is computed by λ. For this purpose we prove the following two technical propositions.

Proposition 2 *Let* $t_0 \leadsto_{p_1, l_1 \to r_1, \sigma_1} t_1 \ldots \leadsto_{p_n, l_n \to r_n, \sigma_n} t_n$ *be a narrowing derivation. Then,* $\forall x \in \mathcal{V}ar(t_n)$ *either* $x \in \mathcal{V}ar(t_0)$ *or* $\exists\, y \in \mathcal{V}ar(t_0)$ *such that* $x \in \mathcal{V}ar(\sigma_n \circ \ldots \circ \sigma_1(y))$.

Proof The proof is by induction on n. The base case, i.e. $n = 0$, is straightforward. Induction step: Let $t_0 \leadsto_{p_1, l_1 \to r_1, \sigma_1} t_1 \cdots t_n \leadsto_{p_{n+1}, l_{n+1} \to r_{n+1}, \sigma_{n+1}} t_{n+1}$ be a narrowing derivation. Let $x \in \mathcal{V}ar(t_{n+1})$. We distinguish two cases.

$x \in \mathcal{V}ar(t_n)$

By the induction hypothesis, we have either $x \in \mathcal{V}ar(t_0)$ or $\exists\, y \in \mathcal{V}ar(t_0)$ such that $x \in \mathcal{V}ar(\sigma_n \circ \cdots \circ \sigma_1(y))$. Since $x \in \mathcal{V}ar(t_{n+1})$, we deduce that $x \notin \mathcal{D}om(\sigma_{n+1})$. Hence, either $x \in \mathcal{V}ar(t_0)$ or $\exists\, y \in \mathcal{V}ar(t_0)$ such that $x \in \mathcal{V}ar(\sigma_{n+1} \circ \cdots \circ \sigma_1(y))$.

$x \notin \mathcal{V}ar(t_n)$

Thus, $x \in \mathcal{V}ar(l_{n+1})$. Since $x \in \mathcal{V}ar(t_{n+1})$, we have $x \notin \mathcal{D}om(\sigma_{n+1})$. Let q be the position of x in l_{n+1}. Then, we have $x = \sigma_{n+1}(l_{n+1})|_q = \sigma_{n+1}(t_n)|_{p_{n+1} \cdot q}$. This implies that $\exists\, z \in \mathcal{V}ar(t_n)$ such that $x \in \mathcal{V}ar(\sigma_{n+1}(z))$. By the induction hypothesis, either $z \in \mathcal{V}ar(t_0)$ or $\exists\, y \in \mathcal{V}ar(t_0)$ such that $z \in \mathcal{V}ar(\sigma_n \circ \cdots \circ \sigma_1(y))$. We distinguish these two subcases.

$z \in \mathcal{V}ar(t_0)$

In this case, $z \notin \mathcal{D}om(\sigma_i)$ for $i = i \cdots n$. Thus, $x \in \mathcal{V}ar(\sigma_{n+1} \circ \cdots \circ \sigma_1(z))$. Hence, the claim holds.

$\exists\, y \in \mathcal{V}ar(t_0)$ such that $z \in \mathcal{V}ar(\sigma_n \circ \cdots \circ \sigma_1(y))$

Since $x \in \mathcal{V}ar(\sigma_{n+1}(z))$, we have $x \in \mathcal{V}ar(\sigma_{n+1} \circ \cdots \circ \sigma_1(y))$. Hence, the claim holds. □

Proposition 3 *Let $\mathcal{R}$ be an inductively sequential rewrite system. Let t be a term, $V = \mathcal{V}ar(t)$ and (p_1, R_1, σ_1) and (p_2, R_2, σ_2) two distinct triples in $\lambda(t, \mathcal{T})$. Then, $\sigma_{1|V}$ and $\sigma_{2|V}$ are independent on V.*

Proof

The proof is by strong arithmetical induction on the number of occurrences of defined operation symbols in t with a nested structural induction on the *pdt* $\mathcal{T}$

$\mathcal{T} = rule(\pi, R)$, for some pattern π and rule R, or $\mathcal{T} = exempt(\pi)$, for some pattern π.

There are no distinct triples in $\lambda(t, \mathcal{T})$ and the claim vacuously holds.

$\mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k)$, for some pattern π, position o, and *pdt*s $\mathcal{T}_1, \ldots, \mathcal{T}_k$, for some $k \geq 0$.

We consider the three subcases of the definition of λ for *branch* nodes.

$t|_o$ is a variable, say x.

In this case t and $pattern(\mathcal{T}_i)$ unify for all $i = 1 \ldots k$. By the induction hypothesis, for every i, the substitutions of distinct triples in $\lambda(t, \mathcal{T}_i)$ are independent on V. Moreover, if $(p_i, R_i, \sigma_i) \in \lambda(t, \mathcal{T}_i)$ and $(p_j, R_j, \sigma_j) \in \lambda(t, \mathcal{T}_j)$ with $i \neq j$, then σ_i and σ_j are independent on V since the roots of $\sigma_i(x)$ and $\sigma_j(x)$ are different constructors. Thus, the claim holds.

$t|_o$ is a constructor-rooted.

By the definition of *pdt*, there exists one i in $\{1, \ldots k\}$ such that $pattern(\mathcal{T}_i)$ and t unify. By the definition of λ, $\lambda(t, \mathcal{T}) = \lambda(t, \mathcal{T}_i)$. By the induction hypothesis, the claim holds for $\lambda(t, \mathcal{T}_i)$ and thus for $\lambda(t, \mathcal{T})$ too.

$t|_o$ is operation-rooted.

By the definition of λ, π and t unify. Let $\tau = mgu(t, \pi)$. Since $t|_o$ is operation-rooted, so is $\tau(t|_o)$. Let $\mathcal{T}'$ be a definitional tree of the root of $\tau(t|_o)$. $\tau_{|V}$ is a constructor substitution since the patterns of definitional trees are linear innermost terms. Thus, $t|_o$ contains fewer defined operation symbols than t. Therefore, by the induction hypothesis on $t|_o$, if (p_1, R_1, σ_1) and (p_2, R_2, σ_2) are distinct triples in $\lambda(\tau(t|_o), \mathcal{T}')$, then $\sigma_{1|\mathcal{V}ar(\tau(t|_o))}$ and $\sigma_{2|\mathcal{V}ar(\tau(t|_o))}$ are independent on $\mathcal{V}ar(\tau(t|_o))$. Therefore, $\sigma_1 \circ \tau_{|V}$ and $\sigma_2 \circ \tau_{|V}$ are independent on V. Hence, the claim holds. □

The next theorem claims that no redundant derivation is computed by λ.

Theorem 5 (Independence of solutions) *Let $\mathcal{R}$ be an inductively sequential rewrite system extended by the equality rules, e an equation to solve and $V = \mathcal{V}ar(e)$. Let $e \overset{+}{\leadsto}_\sigma$ true and $e \overset{+}{\leadsto}_{\sigma'}$ true be two distinct derivations computed by λ. Then, σ and σ' are independent on V.*

Proof First, we prove the claim when the initial steps of $e \overset{+}{\leadsto}_\sigma$ *true* and $e \overset{+}{\leadsto}_{\sigma'}$ *true* differ. By our assumption, the derivations that we are considering are of the forms $e \leadsto_{\sigma_1} e_1 \overset{*}{\leadsto}_{\sigma_2}$ *true* and $e \leadsto_{\sigma_3} e'_1 \overset{*}{\leadsto}_{\sigma_4}$ *true*. This implies that σ_1 and σ_3 belong to distinct triples in $\lambda(e, \mathcal{T})$, where $\mathcal{T}$ is a definitional tree of $\approx$. By Proposition 3, the substitutions $\sigma_{1|V}$ and $\sigma_{3|V}$ are independent on V. By definition of independent substitutions, there exists a variable, x in V such that $\sigma_1(x)$ and $\sigma_3(x)$ are not unifiable. Since $\sigma = \sigma_2 \circ \sigma_1$ and $\sigma' = \sigma_4 \circ \sigma_3$, then $\sigma(x)$ and $\sigma'(x)$ are not unifiable. Hence, $\sigma_{|V}$ and $\sigma'_{|V}$ are independent on V

Now, we consider the general case. By our assumption, the derivations that we are considering are of the forms $e \overset{+}{\leadsto}_{\sigma_1} e_i \overset{+}{\leadsto}_{\sigma_2}$ *true* and $e \overset{+}{\leadsto}_{\sigma_1} e_i \overset{+}{\leadsto}_{\sigma_3}$ *true*, for some $i > 1$. The sub-derivations $e_i \overset{+}{\leadsto}_{\sigma_2}$ *true* and $e_i \overset{+}{\leadsto}_{\sigma_3}$ *true* start from the same equation e_i and their initial steps differ. We have proved that in this case $\sigma_{2|\mathcal{V}ar(e_i)}$ and $\sigma_{3|\mathcal{V}ar(e_i)}$ are independent on $\mathcal{V}ar(e_i)$. Hence, by definition of independent substitutions, there exists a variable y in $\mathcal{V}ar(e_i)$ such that $\sigma_{2|\mathcal{V}ar(e_i)}(y)$ and $\sigma_{3|\mathcal{V}ar(e_i)}(y)$ do not unify. By Proposition 2, either $y \in V$ or $\exists\, z \in V$ such that $y \in \mathcal{V}ar(\sigma_1(z))$. We distinguish these two cases.
Case 1. $y \in V$. Then, $y \notin \mathcal{D}om(\sigma_1)$ and, consequently, $\sigma_2 \circ \sigma_1(y)$ and $\sigma_3 \circ \sigma_1(y)$ dot not unify. Since, $\sigma_{|V} = \sigma_2 \circ \sigma_{1|V}$ and $\sigma'_{|V} = \sigma_3 \circ \sigma_{1|V}$, the claim holds.
Case 2. $\exists\, z \in V$ such that $y \in \mathcal{V}ar(\sigma_1(z))$. Since $\sigma_{2|\mathcal{V}ar(e_i)}(y)$ and $\sigma_{3|\mathcal{V}ar(e_i)}(y)$ dot not unify and $y \in \mathcal{V}ar(\sigma_1(z))$, we deduce that $\sigma_2 \circ \sigma_1(z)$ and $\sigma_3 \circ \sigma_1(z)$ dot not unify. Since, $\sigma_{|V} = \sigma_2 \circ \sigma_{1|V}$ and $\sigma'_{|V} = \sigma_3 \circ \sigma_{1|V}$, the claim holds. □

We now discuss the cost and length of a derivation computed by our strategy.

If p is a needed position of some term t, then in any narrowing derivation of t to a constructor term there is at least one step associated with p. If this step is delayed and p is not outermost, then several descendants of p may be created and several steps may become necessary to narrow this set of descendants, e.g., see Example 3. However, from a practical standpoint, if terms are appropriately represented, the cost of narrowing t at (some descendant of) p is largely independent of where the step occurs in the derivation of t. We formalize this viewpoint, which leads to another optimality result for our strategy.

Definition 15 Let $t \leadsto_{p^i, l^i \to r^i, \sigma^i} t^i$, for i in some set of *indices* $I = \{1, \ldots, n\}$, be a narrowing step such that for any distinct i and j in I, p^i and p^j are disjoint and $\sigma^i \circ \sigma^j = \sigma^j \circ \sigma^i$. We say that t is narrowable to t' in a *multistep*, denoted $t \leadsto_{\langle p^i, l^i \to r^i, \sigma^i \rangle_{i \in I}} t'$, iff $t' = \circ_{\,i \in I}\, \sigma^i(((t[r^1]_{p^1})[r^2]_{p^2}) \ldots [r^n]_{p^n})$, where $\circ_{\,i \in I}\, \sigma^i$ denotes the composition $\sigma^n \circ \ldots \circ \sigma^2 \circ \sigma^1$ (the order is irrelevant.)

When we want to emphasize the difference between a step as defined in Definition 6 and a multistep, we refer to the former as *elementary*. Otherwise, we identify an elementary step with a multistep in which the set of narrowed positions has just one element. A narrowing multistep can be thought of as a set of elementary steps performed in parallel. In fact, the conditions that we impose on the positions and substitutions of each elementary step from which a multistep is defined imply that in a multistep the order in which substitutions are composed and positions are narrowed is irrelevant.

To claim that our strategy is optimal, we assign a "cost" to both a step and a derivation. By convention, an elementary step has unit cost. However, it does not seem appropriate, for practical reasons, to set the cost of a multistep equal to the number of positions narrowed in the step. We will justify our choice after giving our definition of cost.

For any set I and equivalence relation $\sim$ on I, $|I|$ denotes the cardinality of I, and $I/\!\sim$ denotes the quotient of I modulo $\sim$.

Definition 16 Let $\alpha = t_0 \leadsto_{\langle p_1^i, R_1^i, \sigma_1^i \rangle_{i \in I_1}} t_1 \leadsto_{\langle p_2^i, R_2^i, \sigma_2^i \rangle_{i \in I_2}} \cdots$ be a narrowing (multi)derivation. The symbol $\sim_n$ denotes the equivalence relation on I_n defined as follows: for any i and j in I_n, $i \sim_n j$ iff the subterms identified by these indices have a common ancestor, more precisely, there exists some m, less than n, such that for some position q in t_m, both $\circ_{\,k \in I_{n+1}} \sigma_{n+1}^k(t_n|_{p_{n+1}^i})$ and $\circ_{\,k \in I_{n+1}} \sigma_{n+1}^k(t_n|_{p_{n+1}^j})$ are descendants of $\circ_{\,k \in I_{m+1}} \sigma_{m+1}^k(t_m|_q)$.

We call a *family* any maximal subset of equivalent indices. The *cost* of the n-th step of α is the number of families in I_n, i.e., $|I_n/\!\sim_n|$. The *cost* of α, denoted $cost(\alpha)$, is the total cost of its steps.

We say that a family is *complete* iff it cannot be enlarged, and we say that a step is *complete* iff all its families are complete, more precisely, I_n is *complete* iff if i is in I_n, then for any position q of $\circ_{k \in I_n} \sigma_{n+1}^k(t_{n-1})$ such that p_n^i and q have a common ancestor in some term of α, there exists some j in I_n such that $q = p_n^j$. We say that a derivation is *complete* iff all its steps are complete.

If I is the set of indices of a narrowing step and i and j belong to I, then $i \sim j$ iff p_i and p_j are, using an anthropomorphic metaphor, blood related. A complete derivation is characterized by narrowing complete "families," i.e., sets containing all the pairwise blood related subterms of a term. Note that the blood related subterms of a term are all equal and that their positions are pairwise disjoint, thus all of them can be included in a multistep. Our choice of cost measure is suggested by the observation that if $t \rightsquigarrow_{p,R,\sigma} t'$, and q and p are blood related positions, then narrowing t at q "when t is being narrowed at p" involves no additional computation of a substitution and/or a rule, and consequently no additional computation of a substituting term (the instantiation of the right side of a rule,) since the reducts of blood related subterms are all equal, too. This implies that all the members of a family could be "shared" in the representation of t. When this is being done (as in efficient implementations of narrowing [19]), a multistep entailing a whole family does not differ, in practice, from an elementary step.

Theorem 6 *If $\alpha = t \overset{*}{\rightsquigarrow}_\sigma u$ is a complete outermost-needed narrowing multiderivation of a term t into a constructor term u, then α has minimum cost. I.e., for any multiderivation $\beta = t \overset{*}{\rightsquigarrow}_\sigma u$, $cost(\alpha) \leq cost(\beta)$.*

The foundations for a detailed proof would take us too far beyond the boundaries of this paper. Thus, we present only a proof outline.

Proof Outline. Huet and Lèvy [24] formalize, in orthogonal systems, the idea of performing two reduction sequences back-to-back, say A and B, beginning with the *same* term. Performing B after doing A is denoted by $A \sqcup B$. Maranget [33] defines, in labeled term rewriting systems, a notion of complete reduction based on families and on a cost measure which inspired ours. He proves the following non-trivial, but intuitive, results: (1) if A is outermost-needed complete and B is complete, then $cost(A \sqcup B) \leq cost(B \sqcup A)$, i.e., in comparing complete reduction sequences it is never more costly to begin with outermost-needed steps, and (2) every reduction sequence A has an associated complete sequence, $\bar{A}$, and $cost(\bar{A}) \leq cost(A)$, i.e., in comparing steps it is never more costly to include all blood related members. From these results it is relatively easy to show that no reduction sequence of a term to its normal form is cheaper than the outermost-needed complete sequence.

Coming to the proof of our theorem, we lift α to its canonical rewrite sequence, $A = \sigma(t) \overset{*}{\to} u$, and likewise β to B, and adapt the proofs in [33] to our situation. □

Completeness is essential to achieve minimum cost. In fact, if we stick to elementary derivations, the outermost-needed strategy yields the *longest* derivation among those that narrow terms only at needed positions. Families of blood related subterms are created only by non-right-linear rules. The following example highlight these issues.

Example 9 We follow up on Example 3. It is immediate to verify that an outermost-needed narrowing elementary derivation (actually a reduction sequence, since the term is ground) of $double(0+0)$ has 4 steps. The following elementary derivation is shorter.

$$double(0+0) \rightsquigarrow_{1,\mathrm{R}_4,\{\}} double(0) \rightsquigarrow_{\Lambda,\mathrm{R}_6,\{\}} 0+0 \rightsquigarrow_{\Lambda,\mathrm{R}_4,\{\}} 0$$

This derivation is shorter than the outermost-needed one, because its first step narrows the subterm at position 1. By contrast, the first step of the outermost-needed derivation narrows the initial term at the root. This step yields two descendants of position 1, which are both needed. Sharing these blood related subterms would save one step.

Elementary steps are easier to understand and to implement than multisteps. To achieve optimality, we need multisteps only as far as blood related terms are concerned. Full sharing of blood related subterms implies that no family ever contains more than a single member, in practice, and thus any *elementary* step becomes trivially complete. In turn, this equates derivations of minimum cost with those of minimum length. Techniques for rewriting "terms" with shared subterms go under the name of term graph rewriting [47] and adapting them to narrowing, for the systems we are considering, poses no major problem [3].

6 Related work

There are three research topics related to our work: (1) the concept of *need* as the foundation of laziness, (2) strategies for using narrowing in programming, and (3) implementations of narrowing in Prolog.

6.1 Narrowing and need

Seminal studies on the concept of *need* in rewriting appear in [24, 39]. Subsequent variations and extensions, e.g., [7, 21, 27, 30, 33, 40, 41, 45, 48], do not address narrowing, but limit the discussion to rewriting. We have introduced a concept of *need* for narrowing that extends a similar concept for rewriting. We have shown that the concept of need for narrowing is inherently more complicated than that for rewriting. In orthogonal systems, a reduction step has one degree of freedom, the selection of the position, but a narrowing step has two, *both* the position *and* the unifier.

We have discussed only inductively sequential systems. Further research will extend this class to strongly sequential and/or weakly orthogonal systems. The extension to weakly orthogonal systems would weaken our strong optimality result, but include additional non-determinism. Sekar and Ramakrishnan [45] propose *necessary sets* as a generalization of the notion of need for weakly orthogonal systems. Antoy [1] suggests rewriting necessary sets of redexes using *parallel* definitional trees and a function analogous to λ. This approach can be extended to narrowing without major problems.

6.2 Narrowing strategies

The trade-off between power and efficiency is central to the use of narrowing, especially in programming. To this aim, several narrowing strategies, e.g., [6, 9, 12, 13, 14, 15, 16, 18, 20, 22, 31, 35, 36, 37, 38, 44, 49] have been proposed. The notion of completeness has evolved accordingly. Plotkin's classic formulation [43] has been relaxed to completeness w.r.t. ground solutions (e.g. [15]) or completeness w.r.t. *strict* equality and domain-based interpretations, as in [16, 37]. The latter appear more appropriate for narrowing as the computational paradigm of functional logic programming languages in the presence of infinite data structures and computations.

We briefly recall the underlying ideas of a few major strategies and compare them with ours using the following example. We choose a strongly terminating rewrite system with completely defined operations, otherwise all the eager strategies would be immediately excluded. At the end of this section, we summarize the characteristics of these major strategies in a table.

Example 10 The symbols a, b, and c are constructors, whereas f and g are defined operations.

$$
\begin{array}{lll}
f(a) \to a & \mathrm{R}_1 \\
f(b(X)) \to b(f(X)) & \mathrm{R}_2 \\
f(c(X)) \to a & \mathrm{R}_3
\end{array}
\qquad
\begin{array}{rl}
g(a,X) \to b(a) & \mathrm{R}_4 \\
g(b(X),a) \to a & \mathrm{R}_5 \\
g(b(X),b(Y)) \to c(a) & \mathrm{R}_6 \\
g(b(X),c(Y)) \to b(a) & \mathrm{R}_7 \\
g(c(X),Y) \to b(a) & \mathrm{R}_8
\end{array}
$$

The equation to solve is $g(X, f(X)) \approx c(a)$. Our strategy computes only three derivations, only one of which yields a solution.

$$
\begin{array}{l}
g(X,f(X)) \approx c(a) \ \leadsto_{1,\mathrm{R}_4,\{X \mapsto a\}} \ b(a) \approx c(a) \\
g(X,f(X)) \approx c(a) \ \leadsto_{1,\mathrm{R}_8,\{X \mapsto c(X_1)\}} \ b(a) \approx c(a) \\
g(X,f(X)) \approx c(a) \ \leadsto_{1.2,\mathrm{R}_2,\{X \mapsto b(X_1)\}} \ g(b(X_1), b(f(X_1))) \approx c(a) \ \overset{*}{\leadsto}_{\{\}} \ \mathit{true}
\end{array}
$$

Basic narrowing [25] avoids positions introduced by the instantiations of previous steps. Its completeness, and that of its variations, e.g., [6, 20, 22, 31, 35, 38], is known for convergent rewrite systems (see [35] for a systematic study.) This strategy may perform useless steps and computes an infinite search space for our benchmark example.

Innermost narrowing [15] narrows only innermost terms. It is ground complete only for strongly terminating constructor-based systems with completely defined operations. It may perform useless steps and it computes an infinite number of derivations for our benchmark example.

Outermost narrowing [12, 13] narrows outermost operation-rooted terms. This strategy is ground complete only for a restrictive class of rewrite systems. It computes no solution for our benchmark example.

Outer narrowing [49] selects an inner position only when a step at an outer position is impossible. This strategy is complete for constructor-based systems. Outer narrowing behaves as needed narrowing on the benchmark example, however the strategy is not characterized as computing needed steps. Furthermore, [49] describes the enumeration of derivations for E-matching, but not the computation of derivations for general E-unification.

Lazy narrowing [9, 16, 18, 37, 36, 44], similar to outer, narrows an inner term only when the step is demanded to narrow an outer term. For these strategies, the qualifier "lazy" is used as a synonym of "outermost" or "demand driven," rather than in the technical sense we propose. The completeness of these strategies is generally expensive to achieve: [18] requires an ad-hoc implementation of backtracking, with the potential of evaluating some term several times; [16] requires flattening of functional nesting and a specialized WAM-like machine in which terms are dynamically reordered; [37] requires a transformation of the rewrite system which, for our benchmark example, increases the number of operations and lengthen the derivations.

In the requirement columns of the following table C, T and CB denote "confluence", "termination" and "constructor-based systems", respectively.

Strategy	Requirements				Completeness	Optimality	
	C	T	CB	others		independent solutions	length of derivations
basic [25]	X	X			complete		
innermost [15]	X	X	X	completely defined operations	ground complete		
outermost [12, 13]	X	X	X	uniformity	ground complete	(X)	
outer [49]	X		X		complete	(X)	
lazy [9, 16, 37, 44]	X		X	nonambiguity	complete w.r.t. strict equality		
needed	X		X	inductive sequentiality	complete w.r.t. strict equality	X	X

LSE narrowing, a refinement of basic narrowing with additional redundancy tests, ensures that the identical solution (up to variable renaming) is not computed by two different derivations. However, it may be the case that one solution is an instance of another. Hence different solutions computed by LSE narrowing are not independent. Outermost narrowing computes independent solutions if the rules satisfy additional conditions [12, Theorem 3]. Outer narrowing computes independent solutions but only for the E-matching problem.

To summarize, the distinguishing features of our strategy are the following: with respect to eager strategies, completeness for non-terminating rewrite systems; with respect to the so-called lazy strategies, a sharp characterization of laziness; with respect to any strategy, optimality and ease of computation.

6.3 Narrowing in Prolog

Implementations of narrowing in Prolog [2, 8, 26, 32] are proposed as a prototypical and portable integration of functional and logic languages. For example, [8, 26] have been proposed as an alternative to the specialized machines required for *K-LEAF* [16] and *BABEL* [37] respectively. The most recent proposals [2, 32] are based on definitional trees and appear to compute needed steps for inductively sequential systems, although both methods neither formalize nor claim this property. The scheme in [2] computes λ directly by pattern matching. The patterns involved in the computation of λ are a superset of those contained in a definitional tree. This is suggested by claim 1 of Theorem 1 that shows a "strong" need for the positions computed using λ—not only the terms at these positions must be eventually narrowed, but they must be eventually narrowed to *head normal forms*. The resulting implementation takes advantage of this characteristic and its performance appears to be superior to the other proposals.

7 Concluding remarks

We have proposed a new narrowing strategy obtained by extending to narrowing the well-known notion of *need* for rewriting. Need for narrowing appears harder to handle than need for rewriting—to compute a needed narrowing step one must also look ahead a potentially infinite number of substitutions. Remarkably, there is an efficiently algorithm for this computation in inductively sequential systems.

We have contained our discussion to narrowing operation-rooted terms. This limitation shortens our discussion and suffices for solving equations. Extending our results also to constructor-rooted terms is straightforward. To compute an outermost-needed narrowing step of a constructor-rooted term it suffices to compute an outermost-needed narrowing step of any of its maximal operation-rooted subterms.

We have shown how our strategy is easily implemented by pattern matching, and we have reported, in the previous section, its good performance in Prolog with respect to other similar attempts. We have also shown that our strategy computes only independent and optimal derivations. Although all the previously proposed lazy strategies have the latter as their primary goal, our strategy is the only one for which this result is formalized and proved.

We want to conclude with a general assessment of the "overall quality" of the narrowing strategy used by a programming language. The key factor is the trade-off between the size of the class of rewrite systems for which the strategy is complete and the efficiency of its computations. We prove both completeness *and* optimality for inductively sequential systems. We believe that it is possible to extend our result to strongly sequential systems and, in a weaker form, to weakly orthogonal systems.

Acknowledgement

Aart Middeldorp suggested us how to prove [34] our conjecture that the classes of inductively sequential systems and constructor-based strongly sequential systems are the same.

References

[1] S. Antoy. Definitional trees. In *ALP'92*, pages 143–157. Springer LNCS 632, 1992.

[2] S. Antoy. Lazy rewriting in logic programming. Technical Report 90-17, Rev. 2, Portland State University, Portland, OR, 1992. (Submitted for publication).

[3] H. Barendregt, M. van Eekelen, J. Glauert, R. Kenneway, and M. Sleep. Term graph rewriting. In *PARLE'87*, pages 141–158. Springer LNCS 259, 1987.

[4] M. Bellia and G. Levi. The relation between logic and functional languages: a survey. *Journal of Logic Programming*, 3(3):217–236, 1986.

[5] D. Bert and R. Echahed. Design and implementation of a generic, logic and functional programming language. In *ESOP-86*, pages 119–132. Springer LNCS 213, 1986.

[6] A. Bockmayr, S. Krischer, and A. Werner. An optimal narrowing strategy for general canonical systems. In *Proc. of the 3rd Intern. Workshop on Conditional Term Rewriting Systems*, pages 483–497. Springer LNCS 656, 1992.

[7] G. Boudol. Computational semantics of term rewriting systems. In M. Nivat and J. C. Reynolds, editors, *Algebraic methods in semantics*, chapter 5. Cambridge University Press, Cambridge, UK, 1985.

[8] P. H. Cheong. Compiling lazy narrowing into Prolog. *New Generation Computing*, 1992. (to appear).

[9] J. Darlington and Y. Guo. Narrowing and unification in functional programming - an evaluation mechanism for absolute set abstraction. In *Proc. of the Conference on Rewriting Techniques and Applications*, pages 92–108. Springer LNCS 355, 1989.

[10] D. DeGroot and G. Lindstrom, editors. *Logic Programming, Functions, Relations, and Equations.* Prentice Hall, 1986.

[11] N. Dershowitz and J. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science B: Formal Methods and Semantics*, chapter 6, pages 243–320. North Holland, Amsterdam, 1990.

[12] R. Echahed. On completeness of narrowing strategies. In *Proc. CAAP'88*, pages 89–101. Springer LNCS 299, 1988.

[13] R. Echahed. Uniform narrowing strategies. In *Proceedings of the Third International Conference on Algebraic and Logic Programming*, pages 259–275, Volterra, Italy, September 1992.

[14] M. J. Fay. First-order unification in an equational theory. In *Proc. 4th Workshop on Automated Deduction*, pages 161–167, Austin (Texas), 1979. Academic Press.

[15] L. Fribourg. SLOG: A logic programming language interpreter based on clausal superposition and rewriting. In *Proc. IEEE Internat. Symposium on Logic Programming*, pages 172–184, Boston, 1985.

[16] E. Giovannetti, G. Levi, C. Moiso, and C. Palamidessi. Kernel LEAF: a logic plus functional language. *The Journal of Computer and System Sciences*, 42:139–185, 1991.

[17] J. A. Goguen and J. Meseguer. Eqlog: Equality, types, and generic modules for logic programming. In D. DeGroot and G. Lindstrom, editors, *Logic Programming, Functions, Relations, and Equations*, pages 295–363. Prentice Hall, 1986.

[18] W. Hans, R. Loogen, and S. Winkler. On the interaction of lazy evaluation and backtracking. In *Proc. of the 4th International Symposium on Programming Language Implementation and Logic Programming*, pages 355–369. Springer LNCS 631, 1992.

[19] M. Hanus. Compiling logic programs with equality. In *Proc. of the 2nd Int. Workshop on Programming Language Implementation and Logic Programming*, pages 387–401. Springer LNCS 456, 1990.

[20] A. Herold. Narrowing techniques applied to idempotent unification. Technical Report SR-86-16, SEKI, 1986.

[21] C. M. Hoffmann and M. J. O'Donnell. Implementation of an interpreter for abstract equations. In *11th ACM Symposium on the Principle of Programming Languages*, Salt Lake City, 1984.

[22] S. Hölldobler. *Foundations of Equational Logic Programming.* Springer LNCS 353, 1989.

[23] G. Huet and J.-M. Hullot. Proofs by induction in equational theories with constructors. *JCSS*, 25:239–266, 1982.

[24] G. Huet and J.-J. Lévy. Computations in orthogonal term rewriting systems. In J.-L. Lassez and G. Plotkin, editors, *Computational logic: essays in honour of Alan Robinson.* MIT Press, Cambridge, MA, 1991. Previous version: Call by need computations in non-ambiguous linear term rewriting systems, Technical Report 359, INRIA, Le Chesnay, France, 1979.

[25] J.-M. Hullot. Canonical forms and unification. In *Proc. 5th Conference on Automated Deduction*, pages 318–334. Springer LNCS 87, 1980.

[26] J. A. Jiménez-Martín, J. Mariño-Carballo, and J. J. Moreno-Navarro. Efficient implementation of lazy narrowing into PROLOG. In *LOPSTR'92*, 1993. Previous version: Some Techniques for the Efficient Implementation of Lazy Narrowing, Technical Report -FIM.75/LyS/92, Facultad de Informatica, Universidad Politecnica de Madrid, 1992.

[27] J. R. Kennaway. Sequential evaluation strategies for parallel-or and related reduction systems. *Annals of Pure and Applied Logic*, 43:31–56, 1989.

[28] J. R. Kennaway. The specificity rule for lazy pattern-matching in ambiguous term rewrite systems. In *Third European Symp. on Programming*, pages 256–270, 1990. *LNCS* 432.

[29] J. W. Klop. Term Rewriting Systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science, Vol. II*, pages 1–112. Oxford University Press, 1992. Previous version: Term rewriting systems, Technical Report CS-R9073, Stichting Mathematisch Centrum, Amsterdam, 1990.

[30] J. W. Klop and A. Middeldorp. Sequentiality in orthogonal term rewriting systems. *Journal of Symbolic Computation*, pages 161–195, 1991. Previous version: Technical Report CS-R8932, Stichting Mathematisch Centrum, Amsterdam, The Netherlands, 1989.

[31] S. Krischer and A. Bockmayr. Detecting redundant narrowing derivations by the LSE-SL reducibility test. In *Proc. RTA'91*. Springer LNCS 488, 1991.

[32] R. Loogen, F. Lopez Fraguas, and M. Rodríguez Artalejo. A demand driven computation strategy for lazy narrowing. In *Proc. of the 5th International Symposium on Programming Language Implementation and Logic Programming*, pages 184–200. Springer LNCS 714, 1993.

[33] L. Maranget. Optimal derivation in weak lambda-calculi and in orthogonal terms rewriting systems. In *17th Annual Symp. on Principles of Prog. Languages*, pages 255–269. ACM, 1990.

[34] A. Middeldorp, August 1993. Personal Communication.

[35] A. Middeldorp and E. Hamoen. Counterexamples to completeness results for basic narrowing (extended abstract). In *Proceedings of the Third International Conference on Algebraic and Logic Programming*, pages 244–258, Volterra, Italy, September 1992.

[36] J. J. Moreno-Navarro, H. Kuchen, R. Loogen, and M. Rodríguez-Artalejo. Lazy narrowing in a graph machine. In *Proc. Second International Conference on Algebraic and Logic Programming*, pages 298–317. Springer LNCS 463, 1990.

[37] J. J. Moreno-Navarro and M. Rodríguez-Artalejo. Logic programming with functions and predicates: The language BABEL. *Journal of Logic Programming*, 12:191–223, 1992.

[38] W. Nutt, P. Réty, and G. Smolka. Basic narrowing revisited. *Journal of Symbolic Computation*, 7:295–317, 1989.

[39] M. J. O'Donnell. *Computing in Systems Described by Equations.* Springer LNCS 58, 1977.

[40] M. J. O'Donnell. *Equational Logic as a Programming Language.* MIT Press, 1985.

[41] M. J. Oyamaguchi. Nv-sequentiality: A decidable condition for call-by-need computations in term rewriting systems. *SIAM Journal on Computation*, 22(1):114–135, 1993.

[42] P. Padawitz. *Computing in Horn Clause Theories*, volume 16 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1988.

[43] G.D. Plotkin. Building-in equational theories. *Machine Intelligence*, 7:73–90, 1972.

[44] U. S. Reddy. Narrowing as the operational semantics of functional languages. In *Proc. IEEE Internat. Symposium on Logic Programming*, pages 138–151, Boston, 1985.

[45] R. C. Sekar and I. V. Ramakrishnan. Programming in equational logic: Beyond strong sequentiality. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 230–241, Philadelphia, PA, June 1990.

[46] J. R. Slagle. Automated theorem-proving for theories with simplifiers, commutativity, and associativity. *Journal of the ACM*, 21(4):622–642, 1974.

[47] M. R. Sleep, M. J. Plasmeijer, and M. C. J. D. van Eekelen, editors. *Term Graph Rewriting Theory and Practice*. J. Wiley & Sons, Chichester, UK, 1993.

[48] S. Thatte. A refinement of strong sequentiality for term rewriting with constructors. *Information and Computation*, 72:46–65, 1987.

[49] J.-H. You. Enumerating outer narrowing derivations for constructor-based term rewriting systems. *Journal of Symbolic Computation*, 7:319–341, 1989.