A Parallel Narrowing Strategy[*]

Sergio Antoy[†] Rachid Echahed[‡] Michael Hanus[§]

TR 96-1

January 8, 1996

Abstract

We generalize to narrowing the notion of necessary set of redexes proposed by Sekar and Ramakrishnan for rewriting in weakly orthogonal, constructor-based rewrite systems. We define two strategies, one sequential and the other parallel, based on this notion. Both strategies are sound and complete. The parallel strategy is a conservative extensions of two optimal strategies. When applied to inductively sequential rewrite systems, it behaves as needed narrowing and when applied to ground terms it behaves as Sekar and Ramakrishnan rewriting strategy.

[*]This research has been supported in part by the National Science Foundation grant CCR-9406751, the French Centre National de la Recherche Scientifique (CNRS), the German Ministry for Research and Technology (BMFT) grant ITS 9103, and the ESPRIT Basic Research Working Group 6028 (Construction of Computational Logics).

[†]Portland State University, Department of Computer Science Portland, OR 97207, antoy@cs.pdx.edu.

[‡]IMAG-LSR, CNRS, BP 53, F-38041 Grenoble, France, echahed@imag.fr.

[§]Informatik II, RWTH Aachen, D-52056 Aachen, Germany, hanus@informatik.rwth-aachen.de.

A Parallel Narrowing Strategy

SERGIO ANTOY

Portland State University, Portland, Oregon

RACHID ECHAHED

IMAG-LSR, Grenoble, France

MICHAEL HANUS

RWTH Aachen, Germany

Abstract: We generalize to narrowing the notion of necessary set of redexes proposed by Sekar and Ramakrishnan for rewriting in weakly orthogonal, constructor-based rewrite systems. We define two strategies, one sequential and the other parallel, based on this notion. Both strategies are sound and complete. The parallel strategy is a conservative extensions of two optimal strategies. When applied to inductively sequential rewrite systems, it behaves as needed narrowing and when applied to ground terms it behaves as Sekar and Ramakrishnan rewriting strategy.

Categories and Subject Descriptors: D.1.1 [**Programming Techniques**]: Applicative (Functional) Programming; D.1.6 [**Programming Techniques**]: Logic Programming; D.3.3 [**Programming Languages**]: Language Constructs and Features—*Control structures*; D.3.4 [**Programming Languages**]: Processors—*Optimization*; F.4.2 [**Mathematical Logic and Formal Languages**]: Grammars and Other Rewriting Systems; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Trees*; I.1.1 [**Algebraic Manipulation**]: Expressions and Their Representation—*Simplification of expressions.*

General Terms: Algorithms, Languages, Performance, Theory.

Additional Key Words: Functional Logic Programming Languages, Rewrite Systems, Narrowing Strategies, Call-By-Need.

1 Introduction

The interest in integrating functional and logic programming has grown over the last decade, since the languages resulting from this integration are expected to have advantages of both paradigms. Most proposals with a sound and complete operational semantics for the integration of functional

Sergio Antoy was supported in part by the National Science Foundation under grant CCR-9406751.

Rachid Echahed was supported in part by the French Centre National de la Recherche Scientifique (GDR Programmation du CNRS) and by Portland State University.

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).

Authors' addresses: Sergio Antoy, Department of Computer Science, Portland State University, P.O. Box 751, Portland, OR 97207, U.S.A., antoy@cs.pdx.edu; Rachid Echahed, IMAG-LSR, CNRS, BP 53, F-38041 Grenoble, France, echahed@imag.fr; Michael Hanus, Informatik II, RWTH Aachen, D-52056 Aachen, Germany, hanus@informatik.rwth-aachen.de.

and logic programming languages (see [19] for a recent survey) are based on narrowing. Narrowing, originally introduced in automated theorem proving [35], *solves* equations by computing unifiers with respect to an equational theory [11]. 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 addition for natural numbers, which are represented by terms built with 0 and s:

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

To narrow the equation $Z + s(0) \approx s(s(0))$, rule R_2 is applied by instantiating Z to $s(X)$. To narrow the resulting equation, $s(X + s(0)) \approx s(s(0))$, R_1 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 [19]. Recently, an optimal narrowing strategy for inductively sequential rewrite systems (e.g., the rewrite system in Example 1) has been discovered by extending results from term rewriting to narrowing [3]. In this paper we continue with the same approach for a more general class of programs, namely those underlied by weakly orthogonal, constructor-based systems.

Example 2 Consider the following definition of Boolean disjunction known as *parallel-or*.

$$\begin{array}{rcll} X \vee \mathit{true} & \to & \mathit{true} & \mathrm{R}_1 \\ \mathit{true} \vee X & \to & \mathit{true} & \mathrm{R}_2 \\ \mathit{false} \vee \mathit{false} & \to & \mathit{false} & \mathrm{R}_3 \end{array} \qquad (1)$$

A significant difference of this system w.r.t. to the previous one is the overlapping of the first two rules. A consequence of the overlapping is that a term of the form $t_1 \vee t_2$ may be narrowed to normal form by narrowing either t_1 or t_2, although we do not know of any criterion to make this choice without look-ahead.

To place our results in a context, we briefly review relevant results about rewriting strategies. O'Donnell has shown [30] that the parallel outermost strategy is normalizing for almost orthogonal TRSs, hence for weakly orthogonal, constructor-based TRSs. In general, some reductions performed by this strategy could be avoided. This opportunity prompted two substantial improvements. Huet and Lévy have shown [20] that by restricting the class of TRSs to those strongly sequential there is an effective strategy that performs only unavoidable reductions. Sekar and Ramakrishnan [34] have refined O'Donnell's result in a different direction. Within the class of the weakly orthogonal, constructor-based TRSs, they have shown that it is possible to minimize the set of redexes that must be reduced in parallel in a term to compute its normal form. The resulting strategy, similar to Huet and Lévy's, does not take into account the right hand sides of the TRS's rules, and it is optimal among the strategies with this limitation.

Narrowing strategies, to date, mimic rewriting strategies only partially. Huet and Lévy's approach has been extended to narrowing for inductively-sequential TRSs with comparable properties. The resulting strategy, called *needed* [3], performs only unavoidable steps and turns out to be optimal also with respect to the computed unifiers. However, narrowing strategies for weakly orthogonal TRSs depart radically from O'Donnell's and Sekar and Ramakrishnan's approaches in that they are sequential. This departure has a major impact on the operational meaning of completeness of a strategy.

If a ground term t has a normal form, then both O'Donnell's and Sekar and Ramakrishnan's strategies compute the normal form of t, the latter generally more efficiently, by means of deterministic, parallel[1] steps. Narrowing t is equivalent to rewriting it, since we are assuming that t is ground. All the existing narrowing strategies that are known to be ground complete narrow t to its normal form by means of possibly *don't-know* non-deterministic, sequential steps. This notion of completeness is somewhat reductive in the sense that the implementations of these strategies *don't know* how to compute the normal form of t without a severe penalty in efficiency. We guess that this unsatisfying situation has been tolerated only because of other non-deterministic choices, e.g., the unifier of a step, occur in narrowing computations. However, this need not be the case for all ground and for some non-ground terms.

The subject of this paper is a parallel strategy—the first one to be proposed—for narrowing. Our strategy is sound and complete and can be implemented relatively efficiently by unification. It *always* computes the normal form of a ground term, if there exists one, without non-determinism and thus it is ideal for the implementation of functional logic programming languages. Our strategy narrows a necessary set of positions, which generally contains fewer than all the outermost narrowable positions of a term. By virtue of its definition, our parallel strategy falls back to the *needed narrowing strategy* [3] on the inductively sequential portions of a TRS, and consequently is optimal on these portions, and falls back to Sekar and Ramakrishnan's strategy on the ground terms, and consequently is optimal (in a weaker sense) on the ground portions of a computation, too.

The paper is organized as follows. Some preliminary definitions and notations are listed in the next section. Section 3 defines the *weakly needed rewriting strategy* which is a parallel rewriting strategy designed for the class of weakly orthogonal, constructor-based TRSs. In Section 4, we present a *sequential* narrowing strategy which is a natural extension of needed narrowing to overlapping TRSs. We define the *parallel narrowing strategy* in Section 5 and discuss its optimality in Section 6. Comparison with related work is given in Section 7. Section 8 contains our conclusions.

2 Preliminaries

We recall some key notions and notations about rewriting. We are consistent with the conventions of [8, 22]. 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 $\mathcal{V}ar(t) = \varnothing$. A term is called *linear* if it does not contain multiple occurrences of one variable. In the following

[1]In this context, *parallel* means that several, possibly different redexes are simultaneously reduced in a single step.

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 [31] and the functional logic languages *ALF* [17], *BABEL* [28], *K-LEAF* [14], *LPG* [5], *SLOG* [12]). 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 *pattern* if $f \in \mathcal{D}$ and $t_1,\ldots,t_n$ are constructor terms. A term $f(t_1,\ldots,t_n)$ $(n \geq 0)$ is called *operation-rooted term* (respectively *constructor-rooted term*) if $f \in \mathcal{D}$ (respectively $f \in \mathcal{C}$). 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 a pattern, 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 $\{x \mapsto \sigma(x) \mid x \in \mathcal{D}om(\sigma)\}$. We denote by $\mathcal{I}m(\sigma)$ the set of variables introduced by the substitution σ, i.e., $\mathcal{I}m(\sigma) = \bigcup_{x \in \mathcal{D}om(\sigma)} \mathcal{V}ar(\sigma(x))$. 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 (*ground*) *constructor substitution* if $\sigma(x)$ is a (ground) 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 $x \in \mathcal{X}$. We denote by *Sub* the set of all substitutions and by *id* the identity substitution. 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 $t \leq t'$, $t' \leq t$ and $\mathcal{V}ar(t) \cap \mathcal{V}ar(t') = \varnothing$. We define a renaming relation $\equiv$ over terms and substitutions as follows: we write $t \equiv t'$ iff $t \leq t'$ and $t' \leq t$; or equivalently, $t \equiv t'$ iff there exists an injective substitution (renaming substitution) β such that $\mathcal{D}om(\beta) \subseteq \mathcal{V}ar(t)$, $\forall x \in \mathcal{D}om(\beta), \beta(x) \in \mathcal{V}ar(t')$, $\beta(t) = t'$ and $\beta^{-1}(t') = t$. We write $\sigma \equiv \theta[V]$ iff there exists an injective substitution (renaming substitution) β such that $\beta\sigma = \theta[V]$ and $\sigma = \beta^{-1}\theta[V]$.

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 $p \cdot q$ to denote the position resulting from the concatenation of the positions p and q, i.e., we overload the symbol "$\cdot$." We write $p \leq q$ if p is *above* or is a *prefix* of q, i.e., there exists a position q' such that $q = p \cdot q'$, and we write $p \parallel q$ if the positions are *disjoint*, i.e., neither p is prefix of q nor q is prefix of p. (see [8] for details). We denote by $size(t)$, the size of a term t, i.e., $size(t) = cardinal(\{p \mid p \text{ is a non-variable position in } t\})$. The size of a substitution σ is defined as $size(\sigma) = \sum_{x \in \mathcal{D}om(\sigma)} size(\sigma(x))$.

We are now ready to define rewriting.

Definition 5 A *reduction step* is an application of a rewrite rule to a term, i.e., $t \to_{p,R} s$ if there exist a position p, a rewrite rule $R = l \to r$ 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 $q < p$. $\overset{*}{\to}$ denotes the transitive and reflexive closure of $\to$. $\overset{*}{\leftrightarrow}$ denotes the symmetric closure of $\overset{*}{\to}$. A term t is *reducible to* a term s if $t \overset{*}{\to} s$. A term t is called *irreducible* or in *normal form* if there is no term s with $t \to s$. A term s is a *normal form of* t if t is reducible to the irreducible term s. A term rewriting system $\mathcal{R}$ is called *terminating* if there are no infinite rewrite derivations w.r.t. $\mathcal{R}$.

The difficulty in applying reductions steps is the determination of the position where the reduction step is applied. For particular rewrite systems, e.g., the strongly sequential ones [20], it is possible to determine a single position where a reduction step must be performed in order to compute a normal form. However, for rewrite systems with overlapping left-hand sides, such a position may not exist (see [34] for an example), or we may not know how to find it without look-ahead, which defies the reason we want to find it in the first place. For instance, Example 2 shows a term in which it is not apparent which of two subterms should be reduced to compute a normal form. Both terms could be reduced at the same time, therefore, we define reduction multisteps.

Definition 6 Let $t \to_{p_i, l_i \to r_i} t_i$, for i in some set of *indices* $I = \{1,\ldots,n\}$, be a reduction step such that for any distinct i and j in I, p_i and p_j are disjoint. We say that t is reducible to t' in a *multistep*, denoted $t \to_{\{(p_i, l_i \to r_i)\}_{i \in I}} t'$, iff $t' = (\ldots((t[\sigma_1(r_1)]_{p_1})[\sigma_2(r_2)]_{p_2})\ldots[\sigma_n(r_n)]_{p_n})$ such that $\forall i \in I, \sigma_i(l_i) = t_{|p_i}$. We also call the multistep $t \to_{\{(p_i, l_i \to r_i)\}_{i \in I}} t'$ a *parallel rewriting step.*

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 [5, 14, 28]. This is done by narrowing.

Definition 7 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 $l \to r$ of a rewrite rule in $\mathcal{R}$ with $\mathcal{V}ar(t) \cap \mathcal{V}ar(l \to r) = \varnothing$ and a unifier σ of $t_{|p}$ and l such that $s = \sigma(t[r]_p)$. In this case we write $t \leadsto_{p, l \to r, \sigma} s$. If σ is a most general unifier of $t_{|p}$ and l, the narrowing step is called *most general.* We write $t_0 \overset{*}{\leadsto}_\sigma t_n$ if there is a narrowing derivation $t_0 \leadsto_{p_1, R_1, \sigma_1} t_1 \leadsto_{p_2, R_2, \sigma_2} \cdots \leadsto_{p_n, R_n, \sigma_n} t_n$ with $\sigma = \sigma_n \circ \cdots \circ \sigma_2 \circ \sigma_1$.

Since the instantiation of the variables in the rule $l \to r$ 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 3 Referring to Example 1,

$$A + B \rightsquigarrow_{\Lambda, \mathrm{R}_2, \{A \mapsto s(0), B \mapsto 0\}} s(0+0)$$

and

$$A + B \rightsquigarrow_{\Lambda, \mathrm{R}_2, \{A \mapsto s(X)\}} s(X+B)$$

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

Padawitz [32] too distinguishes between narrowing and most general narrowing, but in most papers narrowing is intended as most general narrowing [19]. Most general narrowing has the advantage that most general unifiers are uniquely computable, whereas there exist many independent unifiers. However, as shown in [3], for optimal narrowing strategies it is crucial to drop the requirement for most general unifiers. This paper follows the same approach.

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* [14] and *BABEL* [28].

Definition 8 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 if a rewrite system is orthogonal, to be defined shortly, then it remains orthogonal by the addition of these rules. With these rules a solution of an equation is computed by narrowing it to *true*—an approach also taken in *K-LEAF* [14] and *BABEL* [28]. The following proposition shows the equivalence between reducibility to a same ground constructor term and reducibility to *true* using the equality rules.

Proposition 1 *[3] 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'.*

The following proposition, which is an obvious consequence of the definition of the equality rules, is important since our rewriting and narrowing strategies will be defined only for operation-rooted terms.[2]

Proposition 2 *Let $\mathcal{R}$ be a term rewriting system extended by the equality rules and $t_0 \to t_1 \to t_2 \to \cdots$ be a rewrite derivation w.r.t. $\mathcal{R}$ starting with an equation, i.e., $t_0 = (t \approx t')$. Then the root of each reducible term t_i in this derivation is the operation symbol $\approx$ or $\wedge$.*

To ensure the confluence of the rewrite relation, we also require weak orthogonality.

Definition 9 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*) (where $l' \to r'$ is not a variant of $l \to r$ in case of $p = \Lambda$). $\mathcal{R}$ is *weakly orthogonal* if it is left-linear and for each pair of rules $l \to r, l' \to r' \in \mathcal{R}$, non-variable subterm $l_{|p}$ of l, and mgu σ for $l_{|p}$ and l', the terms $\sigma(l[r']_p)$ and $\sigma(r)$ are identical. $\mathcal{R}$ is *almost orthogonal* if it is weakly orthogonal and for each pair of rules $l \to r, l' \to r' \in \mathcal{R}$, the only possible non-variable subterm of l that may unify with l' is l itself. Since we consider in the following only **C**onstructor-based, **A**lmost orthogonal, **T**erm rewriting systems, we write *CAT* for this class.

It is easy to see that for constructor-based systems almost and weak orthogonality are the same concept, since the left hand sides of the rules are patterns. The difference between these classes however is significant. The notion of *descendant*, well-known for orthogonal systems [20], is extended to almost orthogonal systems without difficulties.

Definition 10 Let $A = t \to_{u,l\to r} t'$ be a reduction 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

$$v \setminus A = \begin{cases} \varnothing & \text{if } u = v, \\ \{v\} & \text{if } u \not\leq v, \\ \{u \cdot p' \cdot q \ \text{ such that } \ r_{|p'} = x\} & \text{if } v = u \cdot p \cdot q \text{ and } l_{|p} = x, \text{ where } x \text{ is a variable.} \end{cases}$$

The set of *descendants* of a position v by a reduction sequence 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 reduction sequence 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 [23]. 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.

[2]This is for the sake of simplicity. The extension of our strategies to constructor-rooted terms is simple but requires an additional case distinction, see [1].

Example 4 Consider the operation that doubles its argument by means of an addition. The rules of addition are in Example 1.

$$double(X) \to X + X \qquad \mathrm{R}_3$$

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

$$double(0 \underline{+} 0) \to_{\Lambda,\mathrm{R}_3} (0 \underline{+} 0) + (0 \underline{+} 0)$$

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

3 Weakly needed rewriting

For inductively sequential systems there exists a narrowing strategy [3] that performs only steps that are needed for computing solutions of equations. This strategy may be considered as a natural extension to narrowing of the sequential rewrite strategy presented in [1]. In this paper we investigate the narrowing relation for the class of weakly orthogonal, constructor-based rewrite systems. We will propose, in the next two sections, two narrowing strategies for this class of TRSs. Both strategies are based on a parallel rewrite strategy that we refer to as *weakly needed rewriting*. This rewrite strategy has been sketched first in [1] and computes the same reduction sequences of [34], although the overall approach is quite different. In this section, we reformulate the weakly needed rewriting strategy and address some of its properties. We begin with some technical definitions.

A definitional tree is a hierarchical structure containing the rules (only the rules' left-hand sides really matter) of a defined operation of a rewrite system. Below we recall both the definition of *parallel* definitional tree and a few results that will come handy later on in our discussion.

The symbols *branch* and *rule* occurring in the next definition, are uninterpreted functions used to classify the nodes of the tree. A definitional tree can be seen as a partially ordered set of patterns with some additional constraints.

Definition 11 $\mathcal{T}$ is a *partial parallel definitional tree*, or *ppdt*, with pattern π iff the depth of $\mathcal{T}$ is finite and one of the following cases holds:

$\mathcal{T} = branch(\pi, \bar{o}, \bar{\bar{\mathcal{T}}})$, where π is a pattern, $\bar{o}$ is a list $o_1, \ldots, o_k$, $k > 0$, of occurrences of distinct variables of π, and $\bar{\bar{\mathcal{T}}}$ is a sequence $\bar{\mathcal{T}}_1, \ldots, \bar{\mathcal{T}}_k$ of sequences of *ppdt*s such that for all j in $\{1, \ldots, k\}$, $\bar{\mathcal{T}}_j = \mathcal{T}_{j_1}, \ldots, \mathcal{T}_{j_{k_j}}$, $c_{j_1}, \ldots, c_{j_{k_j}}$ are different constructors of the sort of $\pi_{|o_j}$, and for all i in $\{1, \ldots, k_j\}$, the pattern in the root of $\mathcal{T}_{j_i}$ is $\pi[c_{j_i}(X_1, \ldots, X_n)]_{o_j}$, where n is the arity of c_{j_i} and $X_1, \ldots, X_n$ are new variables.

$\mathcal{T} = rule(\pi \to r)$, where $\pi \to r$ is a variant of a rule of $\mathcal{R}$.

In the remainder of the paper we will use the notation $pattern(\mathcal{T})$ to denote the pattern argument of a *ppdt* $\mathcal{T}$.

Let $\mathcal{R}$ be a rewrite system. $\mathcal{T}$ is a *parallel definitional tree*, abbreviated *prdt*, of an operation f iff $\mathcal{T}$ is a *ppdt* such that $pattern(\mathcal{T}) = f(X_1, \ldots, X_n)$, where n is the arity of f and $X_1, \ldots, X_n$ are new distinct variables, and for every rule $l \to r$ of $\mathcal{R}$ with $l = f(t_1, \ldots, t_n)$ there exists a leaf $rule(l' \to r')$ of $\mathcal{T}$ such that l is a variant of l', and we say that the node $rule(l' \to r')$ *represents* the rule $l \to r$.

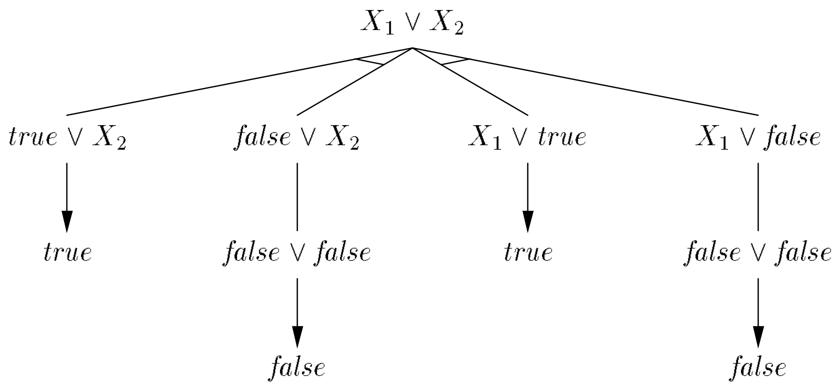

Figure 1: Pictorial representation of a parallel definitional tree of the operation *parallel-or* defined in display (1). The edges connecting a parent to children belonging to the same sequential component of the tree are joined together.

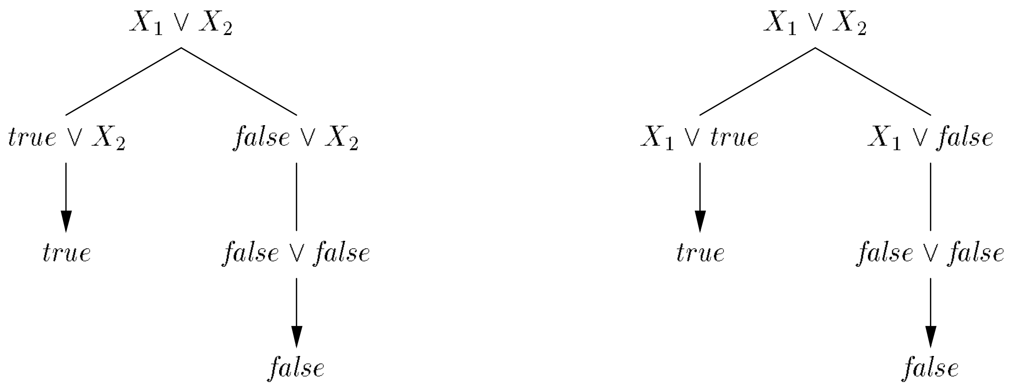

Figure 2: Pictorial representation of the sequential components of the parallel definitional tree of the operation *parallel-or* defined in display (1). Each component is a sequential definitional tree and is obtained by splitting the tree of Fig. 1 at the root.

A (partial) parallel definitional tree $\mathcal{T}$ is called (*partial*) *definitional tree*, abbreviated *pdt*, iff in each branch node of $\mathcal{T}$ the list of occurrences contains exactly one element[3]

Figure 1 pictorially represents the parallel definitional tree of the rules of the parallel-or shown in Example 2.

If the lhs of a rule $l \to r$ of a weakly orthogonal rewrite system is subsumed by the lhs of another, distinct rule, say $l' \to r'$, i.e., $l' < l$, then also $l' \to r' < l \to r$. Subsumed rules are *useless* and can be eliminated from a system without changing the rewrite relation [1, Lemma 18].

If we eliminate all the useless rules from a rewrite system $\mathcal{R}$, then every operation of the resulting system has a parallel definitional tree.

Theorem 1 [1, Th. 19] *If f is an operation of a CAT $\mathcal{R}$, then there exists a parallel definitional tree $\mathcal{T}$ of f such that the rules represented by $\mathcal{T}$ are all and only the useful rules defining f in $\mathcal{R}$.*

The proof of Theorem 1 is constructive, based on an algorithm that, for example, on input the rules of display 1 generates the tree shown in Fig. 1.

From now on, we assume that every rewrite system that we are dealing with has no useless rules.

A parallel definitional tree may be decomposed into a set of *sequential components* each of which is a (sequential) definitional tree. The sequential components of a *ppdt* $\mathcal{T}$ are obtained by the inverse of the "collapsing" operation discussed in [1].

Definition 12 If $\mathcal{T} = rule(l \to r)$, then $\mathcal{T}$ itself is the only *sequential component* of $\mathcal{T}$. If $\mathcal{T} = branch(\pi, \langle o_1, \ldots, o_k\rangle, \langle \mathcal{T}_1, \ldots, \mathcal{T}_k\rangle)$, for some $k > 0$, then $branch(\pi, o_j, \mathcal{T}')$ is a *sequential component* of $\mathcal{T}$ for all j in $1, \ldots, k$ and for all sequential components $\mathcal{T}'$ of $\mathcal{T}_j$.

Below, we recall the definition of needed rewriting. Needed rewriting is a strategy for *inductively sequential systems*, i.e., rewrite systems where each function has a definitional tree. Loosely speaking, the rewriting (and narrowing) strategies presented in this note are obtained by breaking up a CAT into its inductively sequential components, applying needed rewriting (or narrowing) to each component, and combining together the results of each application.

The needed rewriting 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. Throughout an interleaved descent down both t and $\mathcal{T}$, φ computes, whenever possible, a position p and a rule R such that t is reducible at p by rule R.

In the following, $\prec$ denotes the Noetherian ordering on $\mathcal{T}(\Sigma, \mathcal{X}) \times \mathcal{P}(\Sigma)$ (where $\mathcal{P}(\Sigma)$ is the set of all partial definitional trees over the signature Σ) defined by: $(t_1, \mathcal{T}_1) \prec (t_2, \mathcal{T}_2)$ if and only if either: (i) t_1 has fewer occurrences of defined operation symbols than t_2 or (ii) $t_1 = t_2$ and $\mathcal{T}_1$ is a proper subtree of $\mathcal{T}_2$.

Definition 13 The partial function φ takes two arguments, an operation-rooted term t and a partial definitional tree $\mathcal{T}$ such that $pattern(\mathcal{T}) \leq t$. If $\varphi(t, \mathcal{T})$ is defined, it yields a pair, (p, R), where p is a position of t and R is a rewrite rule applicable to t at p. The function φ is defined by induction on $\prec$ as follows.

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

In order to extend the strategy φ to CATs, we apply φ to all the sequential components of a *prdt* and combine the results together. The resulting strategy, denoted $\bar{\varphi}$, is defined as follows.

Definition 14 The function $\bar{\varphi}$ takes two arguments, an operation-rooted term t and a *ppdt* $\mathcal{T}$ such that $pattern(\mathcal{T}) \leq t$. The function $\bar{\varphi}$ yields a finite set of pairs, $\{(p_i, R_i)\}_{i \in I}$, where for all i in I, p_i is a position of t and R_i is a rewrite rule. Thus, let t be a term, $\mathcal{T}$ be a *ppdt* in the domain of $\bar{\varphi}$, φ the function defined in Definition 13, and

$$S = \{(p, R) \mid \mathcal{T}' \text{ is a sequential component of } \mathcal{T}, \varphi(t, \mathcal{T}'\ ) = (p, R)\}$$

[3]This corresponds to the definition given in [3] except that we ignore the *exempt* nodes.

We partition S into two disjoint sets S_1 and S_2 such that, for all distinct pairs $(p_1, R_1), (p_2, R_2) \in S_1$, p_1 and p_2 are disjoint and for all $(p, R) \in S_2$ there is some $(p', R') \in S_1$ with $p' \leq p$. Then $\bar{\varphi}(t, \mathcal{T}) = S_1$, i.e., $\bar{\varphi}(t, \mathcal{T})$ contains all pairs computed by the sequential components of $\mathcal{T}$ with disjoint outermost positions. This definition does not uniquely characterize S_1 if $(p, R_1), (p, R_2) \in S$ with $R_1 \neq R_2$. However, we can always select a uniquely defined subset by introducing a total ordering on rewrite system's rules. Since this does not influence the results of reduction multisteps (due to the triviality of overlapping), a stricter definition is unnecessary.

Example 5 Consider the rewrite system of Example 2 and the term $t = (\mathit{true} \vee (\mathit{true} \vee \mathit{true})) \vee (X \vee (\mathit{false} \vee \mathit{false})$. The rewrite derivation computed by $\bar{\varphi}$ is

$$t \rightarrow_{(1,R_2),(2\cdot 2,R_3)} \mathit{true} \vee (X \vee \mathit{false}) \rightarrow_{(\Lambda,R_2)} \mathit{true}$$

Sometimes we abbreviate this notation as follows

$$t \overset{\bar{\varphi}}{\rightarrow} \mathit{true} \vee (X \vee \mathit{false}) \overset{\bar{\varphi}}{\rightarrow} \mathit{true}$$

We are going to prove that first, unless we perform at least one reduction step computed by $\bar{\varphi}$ we cannot obtain the normal form of an equation and second, that if we perform all the steps computed by $\bar{\varphi}$ we do obtain the normal form (whenever it exists) of an equation.

Next, we lay the foundations for proving these key results. We introduce some technical definitions to simplify the statements that follow and prove a few facts about the defined concepts. We begin with arbitrary reductions.

Definition 15 Let $\mathcal{R}$ be a CAT and t a term. We call *arbitrary reduction* of t a reduction of t by a rule $l \rightarrow r$, where l is the left-hand side of a rule of $\mathcal{R}$ and r is any term. We write $l \leqslant_? t$ iff there exists some term t' that is both a descendant of t w.r.t. arbitrary reduction and an instance of l.

An arbitrary reduction is an abstraction used to capture the impossibility of reducing a term by certain rules, as shown by the next example.

Example 6 Consider the following rewrite rules defining the usual operation "less than or equal to" on the naturals

$$\begin{array}{rcll} 0 \leq X & \rightarrow & \mathit{true} & \mathrm{R}_1 \\ s(X) \leq 0 & \rightarrow & \mathit{false} & \mathrm{R}_2 \\ s(X) \leq s(Y) & \rightarrow & X \leq Y & \mathrm{R}_3 \end{array} \tag{2}$$

and the term $t = s(0) \leq (0 + 0)$. Let l_1 and l_2 denote the left-hand sides of R_1 and R_2. It is immediate to verify that t is not a redex, and that $l_1 \not\leqslant_? t$ and $l_2 \leqslant_? t$. Thus, no descendant of t will ever be reduced by R_1, whereas we cannot exclude that some descendant of t might be reduced by R_2. The latter claim holds for R_3 too.

Lemma 1 *Let $\mathcal{R}$ be a CAT, t an operation-rooted term, and $\mathcal{T}$ a sequential component of a parallel definitional tree of the root of t. Let π be the pattern of a subtree $\mathcal{T}'$ of $\mathcal{T}$ such that $\pi \leq t$. If $l \rightarrow r$ is a rule of $\mathcal{R}$ contained in a leaf of $\mathcal{T}$, but not represented by a leaf of $\mathcal{T}'$, then $l \not\leqslant_? t$.*

Proof The proof is by induction on the depth of $\mathcal{T}'$ in $\mathcal{T}$. The base case vacuously hold. Let *branch*$(\pi', o, \mathcal{T}_1, \ldots, \mathcal{T}_k)$ be the parent of $\mathcal{T}'$. For some i in $1, \ldots, k$, $\mathcal{T}_i = \mathcal{T}'$. The symbol of π at position o is a constructor, say c. The term t has the same constructor at position o. If $j \neq i$, then the pattern of $\mathcal{T}_j$ as well as the left-hand side of any rule $l' \to r'$ represented by a leaf of $\mathcal{T}_j$ have a constructor different from c at position o, and consequently $l' \not\leq_? t$. Thus, the claim stems from the induction hypothesis. □

Lemma 1 justifies the definition of φ. The second case of Definition 13 eliminates rules that cannot be applied at the root of a term t. φ repeats this computation until either it finds an applicable rule (case 1 of the definition) or it finds necessary to reduce a proper subterm of t (case 3).

The next lemma explains why the positions computed by $\bar{\varphi}$ are relevant—unless a reduction is performed at one of these positions, the (constructor) normal form of a term cannot be computed.

Definition 16 Let $\mathcal{R}$ be a CAT, t an operation-rooted term, and $\mathcal{T}$ a parallel definitional tree of the root of t. We say that a position p is *computed by* $\bar{\varphi}$ (on t and $\mathcal{T}$) iff (p, R) is in $\bar{\varphi}(t, \mathcal{T})$ for some rule R in $\mathcal{R}$. We say that a redex $t_{|p}$ is *computed by* $\bar{\varphi}$ (on t and $\mathcal{T}$) iff (p, R) is in $\bar{\varphi}(t, \mathcal{T})$ for some rule R in $\mathcal{R}$.

Lemma 2 *Let $\mathcal{R}$ be a CAT, t an operation-rooted term, and $\mathcal{T}$ a parallel definitional tree of the root of t. No descendant of t can be reduced to a constructor-rooted term unless a descendant of $t_{|p}$ is reduced to a constructor-rooted term for some position p computed by $\bar{\varphi}$.*

Proof If t is a redex, then Λ is the only position computed by $\bar{\varphi}$. In almost orthogonal systems the redex patterns of a term overlap at the root or not at all. Thus, internal reductions do not change both the redexness and the root of t and the claim is immediately verified. If t is not a redex, we prove the claim by structural induction on t. Base case: the claim holds vacuously, since t is an irreducible constant. Inductive case: let f be the root of t, and let $\{q_i\}_{i \in I}$ be the set of positions computed by $\bar{\varphi}$, where I is the set of indices. For all $i \in I$, let $q_i = o_i \cdot p_i$, where o_i and p_i are uniquely determined by the third case of Definition 13. Observe that the root of $t_{|o_i}$ is a defined operation. Observe also that every rule $l \to r$ defining f, that might reduce a descendant of t is, by Lemma 1, represented by a leaf of the node of $\mathcal{T}$ containing o_i. In the left-hand side of any such rule, the symbol at position o_i is a constructor. Thus, no descendant of t can be reduced to a constructor-rooted term unless a descendant of $t_{|o_i}$ is reduced to a constructor-rooted term for at least one i in I. By the induction hypothesis, for all i in I, no descendant of $t_{|o_i}$ can be reduced to a constructor-rooted term unless a descendant of $t_{|o_i|p'}$ is reduced to a constructor-rooted term for some p' computed by $\bar{\varphi}$ on $t_{|o_i}$ and $\mathcal{T}_i$, where $\mathcal{T}_i$ is a definitional tree of the root of $t_{|o_i}$. From the definition of $\bar{\varphi}$, $o_i \cdot p' = o_j \cdot p_j$, for some j in I. Since $t_{|o_i|p'} = t_{|o_i \cdot p'}$, the claim holds by transitivity. □

The above result shows the necessity of reducing *some* position computed by $\bar{\varphi}$, though it may not be obvious which one, for computing the normal form of a term. The next lemma is the foundation for showing that this set of positions is in some sense complete. If we reduce *all* the positions computed by $\bar{\varphi}$, we make some progress toward the computation of normal forms. We measure this notion of progress by the cost function, *cost*, on coinitial multiderivations defined in [3, Def. 8]. Informally, we measure the cost of a multistep of a derivation of a term t as if t were represented as a graph [4, 36].

We denote with a semicolon the concatenation of rewriting or narrowing steps and/or derivations.

Lemma 3 *Let $\mathcal{R}$ be a CAT, t_0 a term, and $t_0 \overset{A_1}{\twoheadrightarrow} t_1 \overset{A_2}{\twoheadrightarrow} t_2 \ldots \overset{A_n}{\twoheadrightarrow} t_n$ a multiderivation normalizing t_0. Let B be a set of disjoint redexes in t_0 such that in any derivation of t_0 to normal form a residual of a redex of B is contracted and let $t_0 \overset{B_0}{\twoheadrightarrow} u_0$ be a multiderivation such that in u_0 there are no descendants of the redexes of B. There exists a multiderivation C normalizing u_0 such that $cost(B_0; C) - cost(B_0) < cost(A)$.*

Proof Define, by induction on n, the multiderivation $C = C_1; \ldots; C_n$ and the family of multiderivations B_i, i in $1, \ldots, n$, as follows: $C_i = A_i \setminus B_{i-1}$ and $B_i = B_{i-1} \setminus A_i$ (see diagram below)

$$\begin{tikzcd}
t_0 \arrow[r, "A_1"] \arrow[d, "B_0"'] & t_1 \arrow[r, "A_2"] \arrow[d, "B_1"'] & \cdots \arrow[r, "A_n"] & t_n \arrow[d, "B_n"'] \\
u_0 \arrow[r, "C_1"] & u_1 \arrow[r, "C_2"] & \cdots \arrow[r, "C_n"] & u_n
\end{tikzcd}$$

and let $t_i \overset{B_i}{\twoheadrightarrow} u_i$, $0 \leq i \leq n$. Since t_n is a normal form, by the commutativity of the diagram, B_n contracts an empty set of redexes and $u_n = t_n$. We now prove the claim about the cost of the derivations. For every i in $1, \ldots, n$, $cost(C_i) \leq cost(A_i)$, since any redex contracted by C_i is the descendant by B_{i-1} of a redex contracted by A_i. By the assumption on B, there exists a step A_j, j in $1, \ldots, n$, that contracts some descendant of some redex, say r, of B. However, no descendant of r is contracted in C_j, since no descendant of r occurs in u_0, and consequently in u_{i-1}, for i in $1, \ldots, n$. Thus, $cost(C_j) < cost(A_j)$ and the claim follows. $\square$

We now use the above results to explore some relationships between $\bar{\varphi}$ and normalizing strategies for CATs.

Theorem 2 *Let $\mathcal{R}$ be a CAT extended by the equality rules, and t an equation $t' \approx t''$ which is reducible to the constant 'true'.*

1. [Necessity] *Every strategy normalizing t must reduce a descendant of t at some position computed by $\bar{\varphi}$.*

2. [Sufficiency] *The strategy $\mathcal{S}$ that reduces the descendants of the redexes computed in t by $\bar{\varphi}$ is normalizing.*

Proof

1. [Necessity] The claim stems by strengthening the antecendent and weakening the consequent of Lemma 2.

2. [Sufficiency] Suppose that $\mathcal{S}$ were not normalizing and that $B_{10}; B_{20}; \ldots$ were an infinite $\mathcal{S}$ derivation of t. Let $t_{00} = t$ and $A_{01}; A_{02}; \ldots; A_{0n}$ be a derivation normalizing t. Consider the following diagram, constructed as in Lemma 3, and, for all i, let A_i denote the derivation

$A_{i1}; A_{i2}; \ldots; A_{in}$.

$$\begin{tikzcd}
t_{00} \arrow[r, "A_{01}"] \arrow[d, "B_{10}"'] & t_{01} \arrow[r, "A_{02}"] \arrow[d, "B_{11}"'] & \cdots \arrow[r, "A_{0n}"] & t_{0n} \arrow[d, "B_{1n}"'] \\
t_{10} \arrow[r, "A_{11}"] \arrow[d, "B_{20}"'] & t_{11} \arrow[r, "A_{12}"] \arrow[d, "B_{21}"'] & \cdots \arrow[r, "A_{1n}"] & t_{1n} \arrow[d, "B_{2n}"'] \\
\vdots & \vdots & & \vdots
\end{tikzcd}$$

For every i, there exists a $d(i)$ such that every redex computed by $\bar{\varphi}$ in t_{i0} has no descendant in $t_{d(i),0}$. By the necessity claim, one redex in this set is reduced in A_i, and consequently $cost(B_{i0}; \ldots; B_{d(i),0}; A_{d(i)}) - cost(B_{i0}; \ldots; B_{d(i),0}) < cost(A_i)$. By considering the horizontal derivations at indices $0, d(0), d^2(0), \ldots$ one would obtain an infinite sequence of strictly decreasing non-negative integers, which is impossible. Thus, $B_{10}; B_{20}; \ldots$ is finite and $\mathcal{S}$ is normalizing. $\square$

The proof of the sufficiency claim also shows that $\mathcal{S}$ is hypernormalizing. The order in which the redexes computed by $\bar{\varphi}$ are reduced and whether other reduction are interspersed with the reductions of these redexes are irrelevant factors as far as computing the normal form of t is concerned.

4 Weakly needed narrowing

In this section we study our first narrowing strategy for weakly orthogonal, constructor-based TRSs. This strategy is sequential and could be seen as a natural extension to overlapping TRSs of needed narrowing [3].

In order to define the narrowing steps starting from a term, we use the sequential components of a parallel definitional tree. Loosely speaking, we apply the function λ defined in [3, Def. 6] to all the sequential components of a *prdt* and combine the results together. Since the function λ computes optimal narrowing derivations for inductively sequential programs (these are programs where each parallel definitional tree has exactly one sequential component), our strategy is a conservative extension of an optimal strategy.

Similar to rewriting, we establish first that the elimination of useless rules from a weakly orthogonal TRS does not change the solutions of equations.

Lemma 4 *If $\mathcal{R}$ is a weakly orthogonal rewrite system and $\mathcal{R}'$ is obtained from $\mathcal{R}$ by removing any useless rule, then, for every equation e, σ is a solution of e in $\mathcal{R}$ if and only if σ is a solution of e in $\mathcal{R}'$.*

Proof Since any rewrite relation is closed under substitution, removing useless rules from $\mathcal{R}$ does not change the rewrite relation as well as the congruence it generates. So, the congruences $\overset{*}{\leftrightarrow}_{\mathcal{R}}$

and $\overset{*}{\leftrightarrow}_{\mathcal{R}'}$ are the same. Therefore, a substitution σ is a solution of an equation e w.r.t. $\mathcal{R}$ if and only if σ is a solution of e w.r.t. $\mathcal{R}'$. □

Before defining our sequential narrowing strategy, we recall the definition of the function λ (in contrast to [3], here we omit the failure elements with the symbol "?"). λ takes two arguments, an operation-rooted term t and a partial definitional tree $\mathcal{T}$ of the root of t, and non-deterministically returns a triple, $(p, l \to r, \sigma)$, where p is a position of t, $l \to r$ is a rule of $\mathcal{R}$ and σ is a substitution. If $(p, l \to r, \sigma) \in \lambda(t, \mathcal{T})$, then $t \leadsto_{p, l \to r, \sigma} \sigma(t[r]_p)$ is a narrowing step.

Definition 17 The function λ takes two arguments, an operation-rooted term t and a partial definitional tree $\mathcal{T}$ such that $pattern(\mathcal{T})$ 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 a rewrite rule and σ is a unifier of $pattern(\mathcal{T})$ and t. Thus, let t be a term and $\mathcal{T}$ a partial definitional tree in the domain of λ. The function λ is defined to yield least sets of triples satisfying the following conditions (note that we use an induction on $\prec$ similarly to Definition 13).

$$\lambda(t,\mathcal{T}) \supseteq \begin{cases} \{(\Lambda, l \to r, mgu(t,l))\} & \text{if } \mathcal{T} = rule(l \to r); \\ \lambda(t, \mathcal{T}_i) & \text{if } \mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k), \\ & \quad t \text{ and } pattern(\mathcal{T}_i) \text{ unify, for some } i; \\ \{(o \cdot p, R, \sigma \circ \tau)\} & \text{if } \mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k), \\ & \quad t_{|o} \text{ is operation-rooted,} \\ & \quad \tau = mgu(t, \pi), \\ & \quad \mathcal{T}' \text{ is a definitional tree of the root of } \tau(t_{|o}) \text{ , and} \\ & \quad (p, R, \sigma) \in \lambda(\tau(t_{|o}), \mathcal{T}'). \end{cases}$$

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 (parallel) definitional trees always contain new variables if they are used in a narrowing step.

Now we are ready to define our sequential narrowing strategy $\bar{\lambda}$.

Definition 18 The function $\bar{\lambda}$ takes two arguments, an operation-rooted term t and a *ppdt* $\mathcal{T}$ such that $pattern(\mathcal{T})$ and t unify. Let $\mathcal{T}_1, \ldots, \mathcal{T}_n$ be the sequential components of the *ppdt* $\mathcal{T}$. Then, the function $\bar{\lambda}$ yields a set of triples of the form (p, R, σ) where p is a position, R is a rewrite rule and σ is a substitution, defined by

$$\bar{\lambda}(t, \mathcal{T}) = \cup_{i=1}^{n} \lambda(t, \mathcal{T}_i)$$

We call *weakly needed step* any narrowing step computed by $\bar{\lambda}$.

Example 7 We compute the set of weakly needed narrowing steps of the term $t = X \vee Y$. Let $\mathcal{T}$ denote the parallel definitional tree of "$\vee$" pictorially represented in Fig. 1. Let $\mathcal{T}_l$ and $\mathcal{T}_r$ be the sequential components of $\mathcal{T}$ pictorially represented in the left and right sides respectively of Fig. 2. Using Definition 17 we obtain

$$\begin{aligned} \lambda(t, \mathcal{T}_l) &= \{(\Lambda, \mathrm{R}_2, \{X \mapsto true\}), (\Lambda, \mathrm{R}_3, \{Y \mapsto false, X \mapsto false\})\} \\ \lambda(t, \mathcal{T}_r) &= \{(\Lambda, \mathrm{R}_1, \{Y \mapsto true\}), (\Lambda, \mathrm{R}_3, \{Y \mapsto false, X \mapsto false\})\} \end{aligned}$$

According to the previous definition, $\bar{\lambda}(t, \mathcal{T})$ is the following set:

$$\{(\Lambda, \mathrm{R}_2, \{X \mapsto \mathit{true}\}), (\Lambda, \mathrm{R}_1, \{Y \mapsto \mathit{true}\}), (\Lambda, \mathrm{R}_3, \{Y \mapsto \mathit{false}, X \mapsto \mathit{false}\})\}$$

The resulting weakly needed narrowing steps are:

$$\begin{array}{l} t \leadsto_{\Lambda, \mathrm{R}_2, \{X \mapsto \mathit{true}\}} \mathit{true} \\ t \leadsto_{\Lambda, \mathrm{R}_1, \{Y \mapsto \mathit{true}\}} \mathit{true} \\ t \leadsto_{\Lambda, \mathrm{R}_3, \{X \mapsto \mathit{false}, Y \mapsto \mathit{false}\}} \mathit{false} \end{array}$$

Now consider the additional rule

$$f(a) \;\rightarrow\; \mathit{true} \qquad \mathrm{R}_4$$

and the term $t' = f(X) \vee f(X)$. Then the results w.r.t. the sequential components $\mathcal{T}_l$ and $\mathcal{T}_r$ are

$$\begin{array}{l} \lambda(t', \mathcal{T}_l) = \{(1, \mathrm{R}_4, \{X \mapsto a\})\} \\ \lambda(t', \mathcal{T}_r) = \{(2, \mathrm{R}_4, \{X \mapsto a\})\} \end{array}$$

According to the previous definition, $\bar{\lambda}(t', \mathcal{T})$ is

$$\{(1, \mathrm{R}_4, \{X \mapsto a\}), (2, \mathrm{R}_4, \{X \mapsto a\})\}$$

which specifies the following narrowing steps

$$\begin{array}{l} t' \leadsto_{1, \mathrm{R}_4, \{X \mapsto a\}} \mathit{true} \vee f(a) \\ t' \leadsto_{2, \mathrm{R}_4, \{X \mapsto a\}} f(a) \vee \mathit{true} \end{array}$$

Theorem 3 (Soundness of weakly needed narrowing) *Let $\mathcal{R}$ be a CAT extended by the equality rules. If $t \approx t' \overset{*}{\leadsto}_\sigma$ true is a narrowing derivation computed by $\bar{\lambda}$, then σ is a solution for $t \approx t'$.*

Proof The claim can be proved as usual (see, e.g., [2, Theorem 2]). □

Next we prove the completeness of narrowing with $\bar{\lambda}$. For this purpose, we will show a strong relation between the functions φ and λ. First, we show a relationship between these functions w.r.t. the sequential components of the parallel definitional trees.

Lemma 5 *Let $\mathcal{R}$ be a CAT. Let t be an operation-rooted term, $\mathcal{T}$ be a sequential component of a parallel definitional tree of the root of t and σ be a constructor substitution. If $\varphi(\sigma(t), \mathcal{T}) = (p, R)$, then there exists a substitution θ such that*

1. $(p, R, \theta) \in \lambda(t, \mathcal{T})$
2. $\theta \leq \sigma[\mathcal{V}ar(t)]$

Proof The proof is by Noetherian induction on $\prec$. We consider the cases of the definition of φ.

Base case: consider $(t, \mathcal{T})$ where t is an operation-rooted term and $\mathcal{T} = \mathit{rule}(l \rightarrow r)$, for some rule $l \rightarrow r$. Hence $(p, R) = (\Lambda, l \rightarrow r)$ and $l \leq \sigma(t)$. This implies the existence of a substitution ϕ with $\phi(l) = \sigma(t)$. Hence l and t are unifiable (we assume that l and t are variable disjoint, otherwise take a new variant of the parallel definitional tree) and there exists a most general unifier θ of l and t with $\theta \leq \sigma[\mathcal{V}ar(t)]$. By the definition of λ, $(p, R, \theta) \in \lambda(t, \mathcal{T})$.

Induction step: consider $(t, \mathcal{T})$ where t is an operation-rooted term and $\mathcal{T} = \mathit{branch}(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k)$, 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.

$\sigma(t)_{|o}$ is constructor-rooted.

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

$\sigma(t)_{|o}$ is operation-rooted.

By the definition of φ, $\pi \leq \sigma(t)$. $\sigma(t)$ and $\mathit{pattern}(\mathcal{T}_i)$ do not unify for each i in $\{1, \ldots k\}$ since $\sigma(t)_{|o}$ is operation-rooted but $\mathit{pattern}(\mathcal{T}_i)$ 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 $\pi \leq \sigma(t)$, there exists a most general unifier τ of π and $\sigma(t)$ with $\tau \leq \sigma[\mathcal{V}ar(t)]$ (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 pattern 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. 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 defined operation symbols than t. Hence we can apply the induction hypothesis to $(\tau(t_{|o}), \mathcal{T}')$ and σ'. If $\varphi(\sigma'(\tau(t_{|o})), \mathcal{T}') = (p', R')$, there exists a substitution θ' such that $(p', R', \theta') \in \lambda(\tau(t_{|o}), \mathcal{T}')$ and $\theta' \leq \sigma'[\mathcal{V}ar(\tau(t_{|o}))]$. By the definition of λ, $(o \cdot p', R', \theta' \circ \tau) \in \lambda(t, \mathcal{T})$, i.e., $(p, R, \theta' \circ \tau) \in \lambda(t, \mathcal{T})$. $\theta' \leq \sigma'[\mathcal{V}ar(\tau(t_{|o}))]$ implies $\theta' \leq \sigma'[\mathcal{V}ar(\tau(t))]$ since θ' instantiates only variables from $\tau(t_{|o})$ and new variables of the definitional tree. Hence $\theta' \circ \tau \leq \sigma' \circ \tau[\mathcal{V}ar(t)]$ which is equivalent to $\theta' \circ \tau \leq \sigma[\mathcal{V}ar(t)]$.

$\sigma(t)_{|o}$ is a variable x.

This case cannot occur since $\varphi(\sigma(t), \mathcal{T})$ is supposed to be defined. □

The following lemma shows how to lift a single reduction step using $\bar{\varphi}$ to a narrowing step using $\bar{\lambda}$.

Lemma 6 *Let $\mathcal{R}$ be a CAT. 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 parallel definitional tree of the root of t. If $\sigma(t) \rightarrow_{p,R} s$ with $(p, R) \in \bar{\varphi}(\sigma(t), \mathcal{T})$, then there exist a narrowing step $t \leadsto_{p,R,\theta} t'$ and a constructor substitution σ' such that $(p, R, \theta) \in \bar{\lambda}(t, \mathcal{T})$, $\sigma'(t') = s$ and $\sigma' \circ \theta = \sigma[V]$.*

Proof Let R be $l \rightarrow r$. Since $\sigma(t) \rightarrow_{p,R} s$, 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 definition of $\bar{\varphi}$, there exists a sequential component $\mathcal{T}'$ of $\mathcal{T}$ with $(p, R) = \varphi(\sigma(t), \mathcal{T}')$. By Lemma 5, there is a triple $(p, R, \theta) \in \lambda(t, \mathcal{T}')$ with $\theta \leq \phi[\mathcal{V}ar(t)]$. Hence $(p, R, \theta) \in \bar{\lambda}(t, \mathcal{T})$ and 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. σ' is a constructor substitution since $\phi_{|V}$ is. Finally, $\sigma'(t') = \sigma'(\theta(t[r]_p)) = \phi(t[r]_p) = \sigma(t)[\rho(r)]_p = s$. □

The following lemma shows that for each term t which has a normal form, there exists a reduction sequence that normalizes t by reducing at each step one position computed by $\bar{\varphi}$.

Lemma 7 *Let $\mathcal{R}$ be a CAT extended by the equality rules and t an equation $t' \approx t''$ that is reducible to the constant 'true'. Then, there exists a derivation*

$$t = t_0 \rightarrow_{(p_1,R_1)} t_1 \cdots \rightarrow_{(p_n,R_n)} t_n$$

such that $\forall i \in \{1,\ldots,n\}, (p_i,R_i) \in \bar{\varphi}(t_{i-1},\mathcal{T}_{i-1})$ where $\mathcal{T}_{i-1}$ is a parallel definitional tree of the root of t_{i-1}, and t_n is the constant 'true'.

Proof The proof is based on a setup identical to that of Theorem 2. Let $t_{00} = t$, let $A_{01}; A_{02}; \ldots; A_{0n}$ be a derivation normalizing t, and, for all i, let A_i denote the derivation $A_{i1}; A_{i2}; \ldots; A_{in}$. By point 1 of Theorem 2, there exists a minimum index j such that $t_{0j} \rightarrow_{(p',R)} t_{0,j+1}$ where (p,R) is computed by $\bar{\varphi}$ on t_{00} and p' is a descendant of p. We define B_{10} as the step $t_{00} \rightarrow_{(p,R)} t_{10}$ and likewise $B_{20}, B_{30}, \ldots$ Observe that $cost(B_{10};A_1) - cost(B_{10}) < cost(A_0)$, since A_1 does not reduce any descendant of p in t. Similar to Theorem 2, for all $i > 0$, $cost(B_{10};\ldots;B_{i0};A_i) - cost(B_{10};\ldots;B_{i0}) < cost(A_{i-1})$. If $B_{10};B_{20};\ldots$ were infinite, one would obtain an infinite sequence of strictly decreasing non-negative integers, which is impossible. $\square$

Now we can prove the completeness of weakly needed narrowing. In fact we show the completeness w.r.t. 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 [21], and lazy narrowing is complete only for constructor substitutions [14, 28]. The following theorem states the completeness of our strategy $\bar{\lambda}$.

Theorem 4 (Completeness of weakly needed narrowing) *Let $\mathcal{R}$ be a CAT 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 narrowing derivation $t \approx t' \overset{*}{\rightsquigarrow}_{\sigma'}$ true computed by $\bar{\lambda}$ such that $\sigma' \leq \sigma[V]$.*

Proof By Definition 8 and Proposition 1, $\sigma(t \approx t')$ is reducible to *true*. By Lemma 7, there exists a derivation

$$\sigma(t \approx t') = s_0 \rightarrow_{(p_1,R_1)} s_1 \rightarrow_{(p_2,R_2)} \cdots \rightarrow_{(p_n,R_n)} s_n \tag{3}$$

such that $(p_i,R_i) \in \bar{\varphi}(s_{i-1},\mathcal{T}_{i-1})$, where $\mathcal{T}_{i-1}$ is a parallel definitional tree of the root of t_{i-1}, for $i \in \{1,\ldots,n\}$, and s_n is the constant *true*. By Proposition 2, s_i is operation-rooted, for $i \in \{1,\ldots,n-1\}$. We prove by induction on n, the length of the derivation (3), the existence of a corresponding narrowing derivation

$$t \approx t' = t_0 \rightsquigarrow_{(p_1,R_1,\theta_1)} t_1 \cdots \rightsquigarrow_{(p_n,R_n,\theta_n)} true \tag{4}$$

such that for $i \in \{1,\ldots,n\}, (p_i,R_i,\theta_i) \in \bar{\lambda}(t_{i-1},\mathcal{T}_{i-1})$ and $\theta_n \circ \cdots \circ \theta_1 \leq \sigma[V]$.

Base case: Consider $n = 1$. In this case, derivation (3) is reduced to one step rewriting $s_0 \rightarrow_{(p_1,R_1)}$ *true*. By Lemma 6, there exists a narrowing step $t_0 \rightsquigarrow_{(p_1,R_1,\theta_1)}$ *true* with $(p_1,R_1,\theta_1) \in \bar{\lambda}(t_0,\mathcal{T}_0)$ and $\theta_1 \leq \sigma[V]$.

Induction step: Consider $n > 1$. By Lemma 6 applied to the first reduction step, there exists a narrowing step $t_0 \rightsquigarrow_{(p_1,R_1,\theta_1)} t_1$ and a constructor substitution σ' with $(p_1,R_1,\theta_1) \in \bar{\lambda}(t_0,\mathcal{T}_0)$,

$\sigma' \circ \theta_1 = \sigma[V]$ and $\sigma'(t_1) = s_1$. Let $V_1 = (V - \mathcal{D}om(\theta_1)) \cup \mathcal{I}m(\theta_1)$. By induction hypothesis applied to V_1, σ' and the derivation

$$s_1 \rightarrow_{(p_2,R_2)} \cdots \rightarrow_{(p_n,R_n)} s_n \ ,$$

there exists a narrowing derivation

$$t_1 \rightsquigarrow_{(p_2,R_2,\theta_2)} t_2 \cdots \rightsquigarrow_{(p_n,R_n,\theta_n)} true \tag{5}$$

such that $(p_i, R_i, \theta_i) \in \bar{\lambda}(t_{i-1}, \mathcal{T}_{i-1})$ for $i \in \{2, \ldots, n\}$ and $\theta_n \circ \cdots \circ \theta_2 \leq \sigma'[V_1]$. Combining the first narrowing step $t_0 \rightsquigarrow_{(p_1,R_1,\theta_1)} t_1$ with derivation (5), we obtain the required derivation (4) with $\theta_n \circ \cdots \circ \theta_1 \leq \sigma[V]$ since $\sigma' \circ \theta_1 = \sigma[V]$. □

If we consider again the term t' in Example 7, we can observe that, to narrow t' to *true*, the strategy $\bar{\lambda}$ computes four distinct derivations with the same substitution $\{X \mapsto a\}$, i.e.,

$$\begin{array}{l}
t' \rightsquigarrow_{1,\mathrm{R}_4,\{X\mapsto a\}} true \vee f(a) \rightsquigarrow_{\Lambda,\mathrm{R}_2,id} true \\
t' \rightsquigarrow_{1,\mathrm{R}_4,\{X\mapsto a\}} true \vee f(a) \rightsquigarrow_{2,\mathrm{R}_4,id} true \vee true \rightsquigarrow_{\Lambda,\mathrm{R}_2,id} true \\
t' \rightsquigarrow_{2,\mathrm{R}_4,\{X\mapsto a\}} f(a) \vee true \rightsquigarrow_{\Lambda,\mathrm{R}_1,id} true \\
t' \rightsquigarrow_{2,\mathrm{R}_4,\{X\mapsto a\}} f(a) \vee true \rightsquigarrow_{1,\mathrm{R}_4,id} true \vee true \rightsquigarrow_{\Lambda,\mathrm{R}_2,id} true
\end{array}$$

In order to avoid such redundant computations we propose a parallel narrowing strategy for weakly orthogonal, constructor-based TRSs in the next section.

5 Parallel Narrowing

Classic narrowing may be defined in two steps as follows: t narrows to t' iff there exists a substitution σ such that the term $\sigma(t)$ rewrites to t' using some rewrite rule $l \rightarrow r$. It is clear from this informal definition that the substitution σ is a *unifier* of the left-hand side l and the subterm of t that has been narrowed. From this informal definition, narrowing differs from rewriting only by the instantiation step. Now, if we generalize this idea to parallel rewriting, i.e., if we replace the rewriting step, in the narrowing relation, by a parallel rewriting step, we obtain a new relation that we call *parallel narrowing.* The definition below formalizes the idea that we just sketched and defines a parallel narrowing step as an instantiation followed by a parallel rewriting step.

Definition 19 Let $\mathcal{R}$ be a term rewriting system, $\mathcal{S}$ a parallel rewriting strategy, t a term and σ a substitution such that $\sigma(t)$ is reducible. We define *parallel narrowing* as a binary relation over terms, denoted by $t \overset{\mathcal{S}}{\approx\!\!>}_\sigma t'$, and defined as follows: $t \overset{\mathcal{S}}{\approx\!\!>}_\sigma t'$ iff $\sigma(t) \overset{\mathcal{S}}{\rightarrow} t'$.

Given a parallel rewriting strategy, the definition of the induced parallel narrowing relation does not specify how narrowing substitutions are computed. This is the rôle of a *parallel narrowing strategy.*

Definition 20 Let $\mathcal{R}$ be a term rewriting system, $\mathcal{S}$ a parallel rewriting strategy. A *parallel narrowing strategy* $\mathcal{N}_\mathcal{S}$ is a function from terms to sets of substitutions, $\mathcal{N}_\mathcal{S} : \mathcal{T}(\Sigma, \mathcal{X}) \rightarrow 2^{Sub}$. A substitution σ is in $\mathcal{N}_\mathcal{S}(t)$ only if there exists a term t' such that $t \overset{\mathcal{S}}{\approx\!\!>}_\sigma t'$. We denote the parallel narrowing relation w.r.t. strategy $\mathcal{N}_\mathcal{S}$ by $\overset{\mathcal{N}_\mathcal{S}}{\approx\!\!>}$.

Throughout this section parallel narrowing is defined upon the parallel rewriting strategy $\bar{\varphi}$. Below we define the parallel narrowing strategy $\bar{\bar{\lambda}}$. There are two main differences w.r.t. weakly needed narrowing: parallel narrowing may disregard some unifiers computed by weakly needed narrowing, and at every narrowing step a necessary set of redexes of the instantiated term is reduced in parallel.

Definition 21 Let $\mathcal{R}$ be a CAT, t an operation-rooted term, $\mathcal{T}$ a parallel definitional tree of the root of t. We define the parallel narrowing strategy $\bar{\bar{\lambda}}$ as follows[4]

$$\begin{aligned}\bar{\bar{\lambda}}(t,\mathcal{T}) = \{\sigma_{|\mathcal{V}ar(t)} \text{ modulo renaming } \mid\, & \exists\,(p,R,\sigma)\in\bar{\lambda}(t,\mathcal{T}),\ \forall\,(q,R',\theta)\in\bar{\lambda}(t,\mathcal{T}),\\ & (\theta\le\sigma[\mathcal{V}ar(t)] \text{ and } \theta\not\equiv id[\mathcal{V}ar(t)] \Rightarrow \sigma\equiv\theta[\mathcal{V}ar(t)]) \text{ and}\\ & (\theta\equiv id[\mathcal{V}ar(t)] \text{ and } q\le p\Rightarrow \sigma_{|\mathcal{V}ar(t)}\equiv id\}.\end{aligned}$$

Intuitively, a substitution σ belongs to $\bar{\bar{\lambda}}(t,\mathcal{T})$ iff σ is either the identity or a minimal substitution (w.r.t. $\le$) among the non-identity substitutions computed by $\bar{\lambda}(t,\mathcal{T})$. Furthermore, whenever two triples (p,R,id) and (q,R',θ) belong to $\bar{\lambda}(t,\mathcal{T})$ with p being a prefix of q ($p\le q$), the substitution θ is not considered by the strategy $\bar{\bar{\lambda}}$. Note that instantiations and positions eliminated by $\bar{\bar{\lambda}}$ will eventually come back later in a derivation, if they are necessary for computing the constructor normal form of a term.

Example 8 Consider the following rewrite rules:

$$\begin{array}{rcll} f(X,s(Y)) & \to & s(0) & \mathrm{R}_1\\ f(0,Y) & \to & s(0) & \mathrm{R}_2\\ h(s(s(X))) & \to & s(0) & \mathrm{R}_3\\ g(X) & \to & s(g(X)) & \mathrm{R}_4\end{array}$$

and the term $t = f(g(X),f(h(Y),f(0,h(s(Y)))))$. One can easily verify that for some parallel definitional tree $\mathcal{T}$

$$\begin{aligned}&\bar{\lambda}(t,\mathcal{T}) = \{(1,\mathrm{R}_4,id),(2{\cdot}1,\mathrm{R}_3,\{Y\mapsto s(s(Y_1))\}),(2{\cdot}2,\mathrm{R}_2,id),(2{\cdot}2{\cdot}2,\mathrm{R}_3,\{Y\mapsto s(Y_2)\})\}\\ &\bar{\bar{\lambda}}(t,\mathcal{T}) = \{id\}\end{aligned}$$

The unifier $\{Y\mapsto s(s(Y_1))\}$ is discarded since it is an instance of $\{Y\mapsto s(Y_2)\}$. The unifier $\{Y\mapsto s(Y_2)\}$ is discarded since its application would create a non-outermost redex.

Proposition 3 (Soundness of the strategy $\bar{\bar{\lambda}}$) *Let $\mathcal{R}$ be a CAT extended by the equality rules. The substitution $\theta_n\circ\cdots\circ\theta_1$ deduced from a derivation computed by $\bar{\bar{\lambda}}$*

$$t\approx t' \overset{\bar{\bar{\lambda}}}{\rightsquigarrow}_{\theta_1} \cdots \overset{\bar{\bar{\lambda}}}{\rightsquigarrow}_{\theta_n} \mathit{true}$$

is a solution of the equation $t\approx t'$.

Proof Let T_i be the i-th term in the considered parallel narrowing derivation. Let P_i be the set of positions of subterms that have been reduced at the i-th step in the considered parallel narrowing derivation, i.e., $P_i=\{p\mid \exists(p,R)\in\bar{\varphi}(\theta_i(T_{i-1}),\mathcal{T}_{i-1})\}$ where $\mathcal{T}_{i-1}$ is a *prdt* of the root of T_{i-1}. Then it is easy to see that we obtain the following parallel rewrite derivation

$$\theta_n\circ\cdots\circ\theta_1(t\approx t')\to_{P_1}\cdots\to_{P_n}\mathit{true}$$

[4]The set notation $\{\sigma_{|\mathcal{V}ar(t)}$ modulo renaming $\mid\cdots\}$ means that this set must not contain two substitutions σ_1,σ_2 with $\sigma_1\equiv\sigma_2[\mathcal{V}ar(t)]$.

where the relation $t \rightarrow_P t'$ means that the term t' is obtained from t by rewriting in parallel the subterms of t at the positions in P. $\Box$

In order to prove the completeness of the strategy $\bar{\bar{\lambda}}$, we need some technical results that we formulate below.

Proposition 4 *Let $\mathcal{R}$ be a CAT, t an operation-rooted term, p a position of t, R a rewrite rule in $\mathcal{R}$, σ a constructor substitution and $\mathcal{T}$ a sequential component of a* prdt *of the root of t such that $\varphi(t, \mathcal{T}) = (p, R)$. Then $\varphi(\sigma(t), \mathcal{T}) = (p, R)$.*

Proof The proof is by Noetherian induction on $\prec$. We consider the cases of the definition of φ.

Base case: consider $(t, \mathcal{T})$ where t is an operation-rooted term and $\mathcal{T} = rule(\pi, R')$ for some pattern π and rule R'. In this case $\varphi(t, \mathcal{T}) = (p, R) = (\Lambda, R')$ and $\pi \leq t$. As $\pi \leq \sigma(t)$ we deduce that $\varphi(\sigma(t), \mathcal{T}) = (p, R') = (\Lambda, R')$.

Induction step: consider $(t, \mathcal{T})$ where t is an operation-rooted term and $\mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k)$, 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 φ for *branch* nodes.

$t_{|o}$ is constructor-rooted.

By the definition of *pdt*, there exists some i in $\{1, \ldots, k\}$ such that $pattern(\mathcal{T}_i) \leq t$. By transitivity of $\leq$, we also have $pattern(\mathcal{T}_i) \leq \sigma(t)$. Then, $\varphi(t, \mathcal{T}) = \varphi(t, \mathcal{T}_i)$ and $\varphi(\sigma(t), \mathcal{T}) = \varphi(\sigma(t), \mathcal{T}_i)$. By induction hypothesis, $\varphi(t, \mathcal{T}_i) = \varphi(\sigma(t), \mathcal{T}_i)$ and thus $\varphi(t, \mathcal{T}) = \varphi(\sigma(t), \mathcal{T})$.

$t_{|o}$ is operation-rooted.

Let $\mathcal{T}'$ be a definitional tree of the root of $t_{|o}$, $\varphi(t_{|o}, \mathcal{T}') = (p_1, R_1)$ and $\varphi(\sigma(t)_{|o}, \mathcal{T}') = (p_2, R_2)$. By the definition of φ, $\varphi(t, \mathcal{T}) = (o \cdot p_1, R_1)$ and $\varphi(\sigma(t), \mathcal{T}) = (o \cdot p_2, R_2)$. By induction hypothesis, $\varphi(\sigma(t)_{|o}, \mathcal{T}') = \varphi(t_{|o}, \mathcal{T}')$, i.e., $(p_1, R_1) = (p_2, R_2)$. Thus, $\varphi(t, \mathcal{T}) = \varphi(\sigma(t), \mathcal{T})$. $\Box$

Proposition 5 *Let $\mathcal{R}$ be a CAT, t an operation-rooted term, σ a constructor substitution and $\mathcal{T}$ a parallel definitional tree of the root of t such that for all $(p, R, \theta) \in \bar{\lambda}(t, \mathcal{T})$, either $\theta_{|\mathcal{V}ar(t)} = id$ or $\theta \not\leq \sigma[\mathcal{V}ar(t)]$. Then, $\bar{\varphi}(t, \mathcal{T}) = \bar{\varphi}(\sigma(t), \mathcal{T})$.*

Proof Let $\mathcal{T}'$ be a sequential component of $\mathcal{T}$. We show that $\varphi(t, \mathcal{T}') = \varphi(\sigma(t), \mathcal{T}')$. For that we use the following two implications: $(p, R) = \varphi(t, \mathcal{T}') \Rightarrow (p, R) = \varphi(\sigma(t), \mathcal{T}')$ and $(p, R) = \varphi(\sigma(t), \mathcal{T}') \Rightarrow (p, R) = \varphi(t, \mathcal{T}')$. The first implication has been proven in Proposition 4. To prove the second implication, let us assume that $(p, R) = \varphi(\sigma(t), \mathcal{T}')$. Then, by Lemma 5, there exists a triple (p, R, θ) in $\lambda(t, \mathcal{T}')$ such that $\theta \leq \sigma[\mathcal{V}ar(t)]$. Thus, from the hypothesis, we infer that $\theta_{|\mathcal{V}ar(t)} = id$. Therefore, $\theta(t) = t$ and consequently $\varphi(t, \mathcal{T}') = \varphi(\sigma(t), \mathcal{T}') = (p, R)$. $\Box$

Proposition 6 *Let $\mathcal{R}$ be a CAT, t an operation-rooted term in normal form, $\mathcal{T}$ a parallel definitional tree of the root of t, σ a constructor substitution and (p, R) a pair in $\bar{\varphi}(\sigma(t), \mathcal{T})$. Then, there exists a substitution θ such that $(p, R, \theta) \in \bar{\lambda}(t, \mathcal{T})$, $\theta \leq \sigma[\mathcal{V}ar(t)]$ and $size(\theta_{|\mathcal{V}ar(t)}) \neq 0$.*

Proof $(p, R) \in \bar{\varphi}(\sigma(t), \mathcal{T})$ implies the existence of a sequential component of $\mathcal{T}$, say $\mathcal{T}'$, such that $(p, R) = \varphi(\sigma(t), \mathcal{T}')$. From Lemma 5, there exists a substitution θ such that

1. $(p, R, \theta) \in \lambda(t, \mathcal{T}')$ and thus $(p, R, \theta) \in \bar{\lambda}(t, \mathcal{T})$ and
2. $\theta \leq \sigma[\mathcal{V}ar(t)]$.

To complete the proof, we show by contradiction that $size(\theta_{|\mathcal{V}ar(t)}) \neq 0$. Assume that $size(\theta_{|\mathcal{V}ar(t)}) = 0$. Then, the substitution $\theta_{|\mathcal{V}ar(t)}$ maps variables to variables. Since the left-hand sides of $\mathcal{R}$ are linear, we deduce that $\varphi(t, \mathcal{T}') = \varphi(\theta(t), \mathcal{T}')$. Thus, $\varphi(t, \mathcal{T}') = (p, R)$ since $\varphi(\theta(t), \mathcal{T}') = (p, R)$. This contradicts the assumption that t is in normal form, i.e., $\varphi(t, \mathcal{T}')$ is not defined. □

Lemma 8 *Let $\mathcal{R}$ be a CAT extended by the equality rules, t_0 a term*[5] *headed either by the operation $\approx$ or $\wedge$ and $\mathcal{T}_0$ a parallel definitional tree of the root of t_0. Let σ_0 be a ground constructor substitution such that $size(\sigma_0) \neq 0$ and $\sigma_0(t_0)$ reduces to the constructor term 'true'. Then, there exists a $\overset{\bar{\varphi}}{\rightarrow}$-derivation*

$$t_0 \overset{\bar{\varphi}}{\rightarrow} t_1 \overset{\bar{\varphi}}{\rightarrow} \cdots \overset{\bar{\varphi}}{\rightarrow} t_n$$

such that $n \geq 0$ and either:

- *t_n is the constructor term 'true', or*
- *$\{(p, R, \theta) \in \bar{\lambda}(t_n, \mathcal{T}_n) \mid \theta \leq \sigma_0[\mathcal{V}ar(t_n)]$ and $size(\theta_{|\mathcal{V}ar(t_n)}) \neq 0\} \neq \varnothing$.*

Proof Let us consider the $\overset{\bar{\varphi}}{\rightarrow}$-derivation issued from t_0.

$$t_0 \overset{\bar{\varphi}}{\rightarrow} t_1 \overset{\bar{\varphi}}{\rightarrow} \cdots \qquad (6)$$

This derivation is either finite or not.

Case where derivation (6) is finite: Let t_n be the last element of derivation (6). t_n is the normal form of t_0 since $\bar{\varphi}$ is normalizing. As $\sigma_0(t_0)$ reduces to *true* and $\mathcal{R}$ is confluent, we deduce that either t_n is the constant *true* or $\sigma_0(t_n)$ reduces to *true*. In the last case, by Proposition 6, there exists a triple (p, R, θ) in $\bar{\lambda}(t_n, \mathcal{T}_n)$ such that $\theta \leq \sigma_0[\mathcal{V}ar(t_n)]$ and $size(\theta_{|\mathcal{V}ar(t_n)}) \neq 0$.

Case where derivation (6) is infinite: Assume that there is no term t_n in derivation (6) such that $\bar{\lambda}(t_n, \mathcal{T}_n)$ includes a triple (p, R, θ) with $\theta \leq \sigma_0[\mathcal{V}ar(t_n)]$ and $size(\theta_{|\mathcal{V}ar(t_n)}) \neq 0$. Then, by Proposition 5, $\bar{\varphi}(t_n, \mathcal{T}_n) = \bar{\varphi}(\sigma_0(t_n), \mathcal{T}_n)$. Therefore, the $\overset{\bar{\varphi}}{\rightarrow}$-derivation issued from $\sigma_0(t_0)$ is infinite. Thus, $\sigma_0(t_0)$ has no normal form which contradicts the hypothesis. □

Lemma 9 *Let $\mathcal{R}$ be a CAT extended by the equality rules, t_0 a term headed either by the operation $\approx$ or $\wedge$ and σ_0 a constructor substitution such that $\sigma_0(t_0)$ reduces to the constructor term 'true'. Then, there exists a $\overset{\bar{\bar{\lambda}}}{\rightsquigarrow}$-derivation issued from t_0*

$$t_0 \overset{\bar{\bar{\lambda}}}{\rightsquigarrow}_{\theta_1} t_1 \overset{\bar{\bar{\lambda}}}{\rightsquigarrow}_{\theta_2} t_2 \cdots \overset{\bar{\bar{\lambda}}}{\rightsquigarrow}_{\theta_n} true$$

such that $n \geq 1$ and $\theta_n \circ \ldots \circ \theta_1 \leq \sigma_0[\mathcal{V}ar(t_0)]$.

[5] In fact, this result holds for any operation-rooted term.

Proof The proof is by induction on the size of σ_0

Base case: Assume that the size of the substitution σ_0 is 0. That is to say, σ_0 maps variables to variables. Since $\bar{\varphi}$ is normalizing (see Theorem 2), the left-hand sides of the rules in $\mathcal{R}$ are left-linear and by hypothesis $\sigma_0(t_0)$ reduces to '*true*', we have the following $\overset{\bar{\varphi}}{\rightarrow}$-derivation:

$$t_0 \overset{\bar{\varphi}}{\rightarrow} t_1 \overset{\bar{\varphi}}{\rightarrow} \cdots \overset{\bar{\varphi}}{\rightarrow} true \tag{7}$$

From Lemma 5, we deduce that for each term t_i in derivation (7) but the last one ($true$), there exists at least a position p_{i+1}, a rule R_{i+1}, and a substitution θ_{i+1} with $\theta_{i+1|\mathcal{V}ar(t_i)} = id$ such that $(p_{i+1}, R_{i+1}, \theta_{i+1}) \in \bar{\lambda}(t_i, \mathcal{T}_i)$ with $\mathcal{T}_i$ a parallel definitional tree of the root of t_i. Thus, derivation (7) is also a $\overset{\bar{\bar{\lambda}}}{\rightsquigarrow}$-derivation

$$t_0 \overset{\bar{\bar{\lambda}}}{\rightsquigarrow}_{\theta_1} t_1 \overset{\bar{\bar{\lambda}}}{\rightsquigarrow}_{\theta_2} \cdots \overset{\bar{\bar{\lambda}}}{\rightsquigarrow}_{\theta_n} true$$

such that $\theta_n \circ \cdots \circ \theta_1 = id[\mathcal{V}ar(t_0)]$. Hence, $\theta_n \circ \cdots \circ \theta_1 \leq \sigma_0[\mathcal{V}ar(t_0)]$.

Induction step: Assume that the size of the substitution σ_0 is not 0. From Lemma 8, there exists a $\overset{\bar{\varphi}}{\rightarrow}$-derivation

$$t_0 \overset{\bar{\varphi}}{\rightarrow} t_1 \overset{\bar{\varphi}}{\rightarrow} \cdots \overset{\bar{\varphi}}{\rightarrow} t_n \tag{8}$$

such that $n \geq 0$ and either:

t_n is the constructor term '*true*'.

As in the base case, from the $\overset{\bar{\varphi}}{\rightarrow}$-derivation we deduce that for each term t_i but the last one, there exists at least a position p_{i+1}, a rule R_{i+1} and a substitution θ_{i+1} with $\theta_{i+1|\mathcal{V}ar(t_i)} = id$ such that $(p_{i+1}, R_{i+1}, \theta_{i+1}) \in \bar{\lambda}(t_i, \mathcal{T}_i)$ with $\mathcal{T}_i$ a parallel definitional tree of the root of t_i. Thus we obtain the following $\overset{\bar{\bar{\lambda}}}{\rightsquigarrow}$-derivation

$$t_0 \overset{\bar{\bar{\lambda}}}{\rightsquigarrow}_{\theta_1} t_1 \overset{\bar{\bar{\lambda}}}{\rightsquigarrow}_{\theta_2} \cdots \overset{\bar{\bar{\lambda}}}{\rightsquigarrow}_{\theta_n} t_n$$

such that $\theta_n \circ \cdots \circ \theta_1 = id[\mathcal{V}ar(t_0)]$. Hence, $\theta_n \circ \cdots \circ \theta_1 \leq \sigma_0[\mathcal{V}ar(t_0)]$.

or:

The set $A = \{\sigma_{|\mathcal{V}ar(t_n)} \mid \exists\, (p, R, \sigma) \in \bar{\lambda}(t_n, \mathcal{T}_n), \sigma \leq \sigma_0[\mathcal{V}ar(t_n)]$ and $size(\sigma_{|\mathcal{V}ar(t_n)}) \neq 0\}$ is not empty.

Since the set A is finite, we can define θ_{n+1} as a minimal substitution in A. By definition of the strategy $\bar{\bar{\lambda}}$, $\theta_{n+1} \in \bar{\bar{\lambda}}(t_n, \mathcal{T}_n)$. By Lemma 5, derivation (8) is also a $\overset{\bar{\bar{\lambda}}}{\rightsquigarrow}$-derivation.

$$t_0 \overset{\bar{\bar{\lambda}}}{\rightsquigarrow}_{\theta_1} t_1 \overset{\bar{\bar{\lambda}}}{\rightsquigarrow}_{\theta_1} \cdots \overset{\bar{\bar{\lambda}}}{\rightsquigarrow}_{\theta_n} t_n \tag{9}$$

In this case, we have $t_n \overset{\bar{\bar{\lambda}}}{\rightsquigarrow}_{\theta_{n+1}} t_{n+1}$. Let us consider the set $P = \bar{\varphi}(\theta_{n+1}(t_n), \mathcal{T}_n)$. From Lemma 1 in [21], we deduce the existence of a constructor substitution σ_1 such that $\sigma_0(t_n) \rightarrow_P \sigma_1(t_{n+1})$ and $\sigma_1 \circ \theta_{n+1} = \sigma_0[\mathcal{V}ar(t_n)]$. Since $size(\theta_{n+1}) \neq 0$, we deduce from $\sigma_1 \circ \theta_{n+1} = \sigma_0[\mathcal{V}ar(t_n)]$ that $size(\sigma_1) < size(\sigma_0)$. Since $\mathcal{R}$ is confluent, $\sigma_1(t_{n+1})$ reduces also to $true$. Then, by the induction hypothesis, there exists a $\overset{\bar{\bar{\lambda}}}{\rightsquigarrow}$-derivation

$$t_{n+1} \overset{\bar{\bar{\lambda}}}{\rightsquigarrow}_{\theta_{n+2}} \cdots \overset{\bar{\bar{\lambda}}}{\rightsquigarrow}_{\theta_m} true \tag{10}$$

such that $m \geq n+2$ and $\theta_m \circ \cdots \circ \theta_{n+2} \leq \sigma_1[\mathcal{V}ar(t_{n+1})]$. Combining derivations (9) and (10), we obtain the required $\overset{\bar{\lambda}}{\rightsquigarrow}$-derivation with $\theta_m \circ \cdots \circ \theta_1 \leq \sigma_0[\mathcal{V}ar(t_0)]$. □

From Lemma 9 we can easily infer the completeness of the strategy $\bar{\bar{\lambda}}$.

Theorem 5 (Completeness of the strategy $\bar{\bar{\lambda}}$) *Let $\mathcal{R}$ be a CAT extended by the equality rules. Let σ be a constructor substitution that is a solution of an equation $t \approx t'$ and V a set of variables containing $\mathcal{V}ar(t) \cup \mathcal{V}ar(t')$. Then there exists a derivation computed by $\bar{\bar{\lambda}}$*

$$t \approx t' \overset{\bar{\bar{\lambda}}}{\rightsquigarrow}_{\theta_1} \cdots \overset{\bar{\bar{\lambda}}}{\rightsquigarrow}_{\theta_n} \mathit{true}$$

such that $\theta_n \circ \cdots \circ \theta_1 \leq \sigma[V]$.

The strategy $\bar{\bar{\lambda}}$ may perform some redundant computations when the considered term rewriting system is terminating, as shown in the following example:

Example 9 Consider the rewrite rules R_1, R_2 and R_3 of Example 8. Let $t = f(h(s(s(Y))), f(h(Y), h(s(Y))))$. Then, for an appropriate *prdt* $\mathcal{T}$, $\bar{\bar{\lambda}}(t, \mathcal{T}) = \{id, \{Y \mapsto s(Y_2)\}\}$. If we develop the search space of t, we will compute twice the substitution $\{Y \mapsto s(Y_2)\}$. However, if we consider only minimal substitutions (including identity) in the sets computed by $\bar{\bar{\lambda}}$, we will compute only once the substitution $\{Y \mapsto s(Y_2)\}$ for the considered term t.

Next we improve the strategy $\bar{\bar{\lambda}}$ and define a new parallel narrowing strategy, denoted by $\bar{\bar{\lambda}}_\downarrow$, which avoids some redundant computations performed by $\bar{\bar{\lambda}}$ for terminating TRSs.

Definition 22 Let $\mathcal{R}$ be a terminating CAT, t an operation-rooted term, $\mathcal{T}$ a parallel definitional tree of the root of t. The parallel narrowing strategy $\bar{\bar{\lambda}}_\downarrow$ is defined by

$$\begin{aligned}\bar{\bar{\lambda}}_\downarrow(t, \mathcal{T}) = \{\sigma \text{ modulo renaming} \mid \exists\ (p, R, \sigma) \in \bar{\lambda}(t, \mathcal{T}) \text{ and } \forall\ (q, R', \theta) \in \bar{\lambda}(t, \mathcal{T}),\\ (\theta \leq \sigma[\mathcal{V}ar(t)] \Rightarrow \sigma \equiv \theta[\mathcal{V}ar(t)])\}\ .\end{aligned}$$

Notice that $\bar{\bar{\lambda}}_\downarrow(t, \mathcal{T})$ is always included in $\bar{\bar{\lambda}}(t, \mathcal{T})$.

Example 10 Let us consider the term t given in Example 9. There is only one narrowing derivation developed by the strategy $\bar{\bar{\lambda}}_\downarrow$, starting from t. This derivation, given below, computes the substitution $\{Y \mapsto s(Y_1)\}$.

$$\begin{aligned}f(h(s(s(Y))), f(h(Y), h(s(Y)))) &\overset{\bar{\bar{\lambda}}_\downarrow}{\rightsquigarrow}_{id} && f(s(0), f(h(Y), h(s(Y))))\\ &\overset{\bar{\bar{\lambda}}_\downarrow}{\rightsquigarrow}_{\{Y \mapsto s(Y_1)\}} && f(s(0), f(h(s(Y_1)), s(0)))\\ &\overset{\bar{\bar{\lambda}}_\downarrow}{\rightsquigarrow}_{id} && f(s(0), s(0))\\ &\overset{\bar{\bar{\lambda}}_\downarrow}{\rightsquigarrow}_{id} && s(0)\end{aligned}$$

The completeness of the strategy $\bar{\bar{\lambda}}_\downarrow$ is based on the following lemma.

Lemma 10 *Let $\mathcal{R}$ be a terminating CAT, t_0 a term headed either by the operation $\approx$ or $\wedge$ and σ_0 a constructor substitution such that $\sigma_0(t_0)$ reduces to the constructor term 'true'. Then, there exists a $\overset{\bar{\bar{\lambda}}_{\downarrow}}{\rightsquigarrow}$-derivation issued from t_0*

$$t_0 \overset{\bar{\bar{\lambda}}_{\downarrow}}{\rightsquigarrow}_{\theta_1} t_1 \overset{\bar{\bar{\lambda}}_{\downarrow}}{\rightsquigarrow}_{\theta_2} \cdots \overset{\bar{\bar{\lambda}}_{\downarrow}}{\rightsquigarrow}_{\theta_n} true$$

such that $n \geq 1$ and $\theta_n \circ \cdots \circ \theta_1 \leq \sigma_0[\mathcal{V}ar(t_0)]$.

Proof The proof is by induction on the size of σ_0

Base case: The base case, i.e. $size(\sigma_0) = 0$, is the same as in Lemma 9.

Induction step: Assume that the size of the substitution σ_0 is not 0. Since $\bar{\varphi}$ is normalizing and $\mathcal{R}$ is terminating, the normal form of t_0 exists and can be obtained by the following $\xrightarrow{\bar{\varphi}}$-derivation

$$t_0 \xrightarrow{\bar{\varphi}} t_1 \xrightarrow{\bar{\varphi}} \cdots \xrightarrow{\bar{\varphi}} t_n \tag{11}$$

It is clear that derivation (11) is also a $\bar{\bar{\lambda}}_{\downarrow}$-derivation. Now if t_n is the constructor term '*true*', the claim is obvious. Otherwise, $\sigma_0(t_n)$ reduces to '*true*'. Define the set S as

$$\{(p, R, \theta) \mid (p, R, \theta) \in \bar{\lambda}(t_n, \mathcal{T}_n) \text{ and } \theta \leq \sigma_0[\mathcal{V}ar(t_n)] \text{ and } size(\theta_{|\mathcal{V}ar}(t_n)) \neq 0\}\,.$$

Since t_n is in normal form, S is not empty by Proposition 6. Then we can choose θ_{n+1} so that there exists a triple $(p_{n+1}, R_{n+1}, \theta_{n+1})$ in S and θ_{n+1} is minimal among the substitutions occurring in S, i.e., $\forall\ (q, R, \beta) \in S, \beta \not\leq \theta_{n+1}$. Then $\theta_{n+1} \in \bar{\bar{\lambda}}_{\downarrow}(t_n)$. Thus we obtain $t_n \overset{\bar{\bar{\lambda}}_{\downarrow}}{\rightsquigarrow}_{\theta_{n+1}} t_{n+1}$. Let us consider the set of positions $P = \bar{\varphi}(\theta_{n+1}(t_n), \mathcal{T}_n)$. From Lemma 1 in [21], we deduce the existence of a constructor substitution σ_1 such that $\sigma_0(t_n) \rightarrow_P \sigma_1(t_{n+1})$ and $\sigma_1 \circ \theta_{n+1} = \sigma_0[\mathcal{V}ar(t_n)]$. Thus, since $size(\theta_{n+1}) \neq 0$, we deduce from $\sigma_1 \circ \theta_{n+1} = \sigma_0[\mathcal{V}ar(t_n)]$ that $size(\sigma_1) < size(\sigma_0)$. Since $\mathcal{R}$ is confluent, $\sigma_1(t_{n+1})$ reduces also to *true*. Then, by induction hypothesis, there exists a $\rightsquigarrow$-derivation.

$$t_{n+1} \overset{\bar{\bar{\lambda}}_{\downarrow}}{\rightsquigarrow}_{\theta_{n+2}} \cdots \overset{\bar{\bar{\lambda}}_{\downarrow}}{\rightsquigarrow}_{\theta_m} true \tag{12}$$

such that $m \geq n+2$ and $\theta_m \circ \cdots \circ \theta_{n+2} \leq \sigma_1[\mathcal{V}ar(t_{n+1})]$. Combining derivations (11) and (12), we obtain the required $\rightsquigarrow$-derivation with $\theta_m \circ \cdots \circ \theta_1 \leq \sigma_0[\mathcal{V}ar(t_0)]$. □

From Lemma 10 we can easily infer the completeness of the strategy $\bar{\bar{\lambda}}_{\downarrow}$.

Theorem 6 (Completeness of the strategy $\bar{\bar{\lambda}}_{\downarrow}$) *Let $\mathcal{R}$ be a terminating CAT extended by the equality rules. Let σ be a constructor substitution that is a solution of an equation $t \approx t'$ and V a set of variables containing $\mathcal{V}ar(t) \cup \mathcal{V}ar(t')$. Then there exists a derivation computed by $\bar{\bar{\lambda}}_{\downarrow}$*

$$t \approx t' \overset{\bar{\bar{\lambda}}_{\downarrow}}{\rightsquigarrow}_{\theta_1} \cdots \overset{\bar{\bar{\lambda}}_{\downarrow}}{\rightsquigarrow}_{\theta_n} true$$

such that $\theta_n \circ \cdots \circ \theta_1 \leq \sigma[V]$.

Notice that the parallel narrowing strategy $\bar{\bar{\lambda}}_\downarrow$ is not complete for non-terminating TRSs, as shown in the following example.

Example 11 Consider the following non-terminating TRS:

$$\begin{array}{rll} f(X,0) \to 0 & & \mathrm{R}_1 \\ f(0,X) \to 0 & & \mathrm{R}_2 \\ h(0) \to 0 & & \mathrm{R}_3 \\ g(X) \to g(s(X)) & & \mathrm{R}_4 \end{array}$$

Let u be the term $f(g(X),h(Y))$. The strategy $\bar{\bar{\lambda}}_\downarrow$ computes only one derivation starting from u which is

$$\begin{array}{rll} f(g(X),h(Y)) & \overset{\bar{\bar{\lambda}}_\downarrow}{\rightsquigarrow}_{id} & f(g(s(X)),h(Y)) \\ & \overset{\bar{\bar{\lambda}}_\downarrow}{\rightsquigarrow}_{id} & f(g(s(s(X))),h(Y)) \\ & \overset{\bar{\bar{\lambda}}_\downarrow}{\rightsquigarrow}_{id} & f(g(s(s(s(X)))),h(Y)) \\ & \overset{\bar{\bar{\lambda}}_\downarrow}{\rightsquigarrow}_{id} & \cdots \end{array}$$

However, the term u can be narrowed, using the strategy $\bar{\bar{\lambda}}$, to the constructor term 0 as shown below.

$$f(g(X),h(Y)) \overset{\bar{\bar{\lambda}}}{\rightsquigarrow}_{\{Y\mapsto 0\}} f(g(s(X)),0) \overset{\bar{\bar{\lambda}}}{\rightsquigarrow}_{id} 0$$

6 Optimality

In this section we discuss two optimality results of our narrowing strategies. Inductively sequential systems are a subclass of weakly orthogonal, constructor-based systems. Within our framework it is convenient to look at the differences between these two classes in terms of definitional trees. An inductively sequential operation f has a parallel definitional tree $\mathcal{T}$ with exactly one sequential component, i.e., $\mathcal{T}$ itself is a (sequential) definitional tree. Both weakly needed narrowing and parallel narrowing behave as needed narrowing when they operate on such a tree.

Theorem 7 *Let $\mathcal{R}$ be a CAT, t an operation-rooted term whose defined operations symbols are all inductively sequential. Then, for appropriate definitional trees for the operations in t, the narrowing steps of t computed by both weakly needed narrowing and parallel narrowing are the same as the narrowing steps of t computed by needed narrowing.*

Proof Each operation in t has a sequential definitional tree. Suppose that these trees are used to compute weakly needed or parallel narrowing steps. Let S, $\bar{S}$, and $\bar{\bar{S}}$ be the set of steps computed on t by needed narrowing, weakly needed narrowing and parallel narrowing respectively. It is easy to verify from the definitions of λ and $\bar{\lambda}$ that S and $\bar{S}$ differ at most for triples in S of the form $(p, ?, \sigma)$ which are absent from $\bar{S}$. These triples do not denote a narrowing step, rather, they are used in needed narrowing to detect exceptional computations [3, p. 273]. Since needed narrowing computes only independent substitutions, from the definition of $\bar{\bar{\lambda}}$ we obtain that $\bar{S} = \bar{\bar{S}}$, too. □

We now turn our attention to the behavior of parallel narrowing on ground equations.

Theorem 8 *The parallel narrowing strategy is (deterministically) normalizing on ground equations.*

Proof Let t be a ground equation. Weakly needed narrowing applied to t computes a substitution that is the identity on the variables of t. Hence, by Definition 19 a parallel narrowing step of t is a rewriting step of t according to $\bar{\varphi}$. The claim is thus a direct consequence of Theorem 2. $\square$

The above results show that parallel narrowing is a conservative extension of two optimal strategies, needed narrowing on inductively sequential systems and rewriting necessary sets on ground terms.

The strong optimality results of needed narrowing do not hold for weakly needed and parallel narrowing. In particular, we recall that rewriting and/or narrowing needed positions is not always possible in almost orthogonal TRSs, since such positions generally do not exist [34]. Furthermore, computing only independent unifiers seems unlikely, too, without look-ahead.

Example 12 Consider the parallel-or of Example 2 together with the following operations

$$\begin{aligned} f(0,X) &\to X \\ h(0) &\to \mathit{true} \end{aligned}$$

and the equation $\mathit{true} \approx f(X,h(Y)) \vee f(Y,h(X))$. Parallel narrowing computes two derivations of t beginning with different unifiers, eventually to discover that they yield the same substitution.

7 Related work

In this section we compare our parallel narrowing strategy with other narrowing strategies proposed for constructor-based weakly orthogonal rewrite systems. There are also many narrowing strategies for rewrite systems which are not necessarily constructor-based and weakly orthogonal, like innermost [12], outermost [9, 10], basic [21], or LSE narrowing [6]. However, all these strategies require the termination of the rewrite relation which is difficult to check[6] and immediately excludes typical functional programming techniques like infinite data structures. Therefore, we are interested in narrowing strategies for rewrite systems which permit non-terminating rules. In order to ensure the confluence of the rewrite relation, constructor-based and weakly orthogonal rules are a natural requirement. For this class of rewrite systems, lazy narrowing has been proposed in [7, 14, 28, 27, 33]. Similarly to lazy evaluation in functional languages, lazy narrowing evaluates an inner term only when its value is demanded to narrow an outer term. In contrast to functional languages, a naive version of lazy narrowing may evaluate the same argument several times due to the non-deterministic choice of a function's rewrite rules. Therefore, several methods have been proposed aiming at evaluating arguments commonly demanded by all rules before the non-deterministic choice [3, 15, 25, 29]. The currently best strategy is needed narrowing [3] since it is the only one which is shown to be optimal w.r.t. the length of derivations and the number of computed solutions. Needed narrowing is defined for inductively sequential systems, where in every term that, roughly speaking, is not fully evaluated there always exists a needed position. We have shown in Theorem 7 that parallel narrowing is a conservative extension of an optimal narrowing strategy.

[6]Since the termination property of a rewrite system is undecidable, there are only sufficient criteria for it.

In case of overlapping rules,[7] the situation is more difficult since an argument may be demanded by some rule but not demanded by another rule for the same function. Naive lazy narrowing is a complete but often inefficient strategy for overlapping rules as the following example shows.

Example 13 Consider the following term rewriting system:

$$\begin{array}{rll@{\qquad\qquad}rl}
one(0) \to s(0) & \mathrm{R}_1 & & f(0,X) \to 0 & \mathrm{R}_3 \\
one(s(X)) \to one(X) & \mathrm{R}_2 & & f(X,0) \to 0 & \mathrm{R}_4
\end{array}$$

In order to solve the goal $f(0, one(X)) \approx 0$, lazy narrowing selects a rule and evaluates the demanded arguments to head normal form (i.e., constructor-rooted term or variable) before the rule is applied. If rule R_3 is selected, the first argument of $f(0, one(X))$ is demanded, and the second argument is demanded if rule R_4 is selected. Unfortunately, there are infinitely many narrowing derivations of $one(X)$ to a head normal form—for every $n \geq 0$, $one(X) \overset{*}{\rightsquigarrow}_{\{X \mapsto s^n(0)\}} s(0)$. Therefore, lazy narrowing has an infinite search space since both rules R_3 and R_4 are tried.

In order to avoid this drawback of lazy narrowing, there are at least three methods aiming to improve lazy narrowing for overlapping rewrite rules:

1. Dynamic cut [26]
2. Lazy narrowing with simplification [18]
3. Parallel narrowing

Loogen and Winkler [26] propose the *dynamic cut* which ignores subsequent alternative rules for narrowing if a rule is applicable without binding of goal variables (note that the dynamic cut has been developed in the context of a sequential implementation of narrowing by backtracking). In the previous example, rule R_3 is the first rule which is used to narrow the left-hand side of the goal $f(0, one(X)) \approx 0$. Since the argument 0 is already in head normal form, narrowing produces the trivial goal $0 \approx 0$ without binding the goal variable X. Therefore, the alternative rule R_4 is ignored which avoids the infinite search space.

The effect of the dynamic cut is subsumed by our parallel narrowing strategy since parallel narrowing prefers deterministic reductions at the root of a function call. This is made precise by the following proposition.

Proposition 7 *Let $\mathcal{R}$ be a CAT, t an operation-rooted term and $l \to r \in \mathcal{R}$ a rule with $\sigma(l) = t$ for some substitution σ. Then $t \overset{\bar{\bar{\lambda}}}{\approx\!\!\!\rightsquigarrow}_{id} \sigma(r)$ is the only parallel narrowing step starting at t.*

Proof Let $\mathcal{T}$ be a parallel definitional tree of the root of t. Since all rules of the root of t are contained in $\mathcal{T}$, there is a sequential component $\mathcal{T}'$ of $\mathcal{T}$ containing the rule $l \to r$. W.l.o.g. we assume that $l \to r$ and t have no variables in common, otherwise take a variant of $l \to r$. Since $\sigma(l) = t$ and φ computes outermost redexes [2, Theorem 3], $\varphi(t, \mathcal{T}') = (\Lambda, \sigma)$ (w.l.o.g. we assume that $\mathcal{D}om(\sigma) \subseteq \mathcal{V}ar(l)$). Moreover, $(\Lambda, l \to r, \sigma) \in \lambda(t, \mathcal{T}') \subseteq \bar{\lambda}(t, \mathcal{T})$. Since $\sigma_{|\mathcal{V}ar(t)} = id$, $\bar{\bar{\lambda}}(t, \mathcal{T}) = \{id\}$ by definition of $\bar{\bar{\lambda}}$. By definition of $\overset{\bar{\bar{\lambda}}}{\approx\!\!\!\rightsquigarrow}$, there exists exactly one parallel narrowing step for t, namely $t \overset{\bar{\bar{\lambda}}}{\approx\!\!\!\rightsquigarrow}_{id} \sigma(r)$. □

[7]There exist also non-overlapping rewrite systems which are not inductively sequential, but they seem to be not relevant for application programs.

This proposition also shows another advantage of our parallel narrowing strategy in comparison to the dynamic cut: parallel narrowing is independent of the order of rewrite rules. If there is a rewrite rule applicable to the root without instantiating goal variables, parallel narrowing performs exactly one step which is indeed a reduction step. However, the dynamic cut only discards alternative rules *after* the current rule. For instance, the dynamic cut has no effect for the goal $f(one(X), 0) \approx 0$ w.r.t. Example 13. This disadvantage is omitted in [18] where the combination of lazy narrowing with possible reduction steps between narrowing steps is proposed. For instance, a reduction step applied to the left-hand side of the goal $f(one(X), 0) \approx 0$ reduces it to the trivial goal $0 \approx 0$ and avoids the infinite search space. Attempts for reduction steps are also made at inner positions if they are demanded. For instance, the goal $f(f(one(X), 0), X) \approx 0$ is also reduced to $0 \approx 0$ before a non-deterministic narrowing step can take place. In order to ensure the completeness of this *lazy narrowing with simplification* strategy, a terminating subset of all rewrite rules is used for reduction. For a terminating CAT, lazy narrowing with simplification is subsumed by parallel narrowing: if a simplification step is applicable at some demanded subterm of t, the identity substitution id is contained in the set $\bar{\bar{\lambda}}(t, \mathcal{T})$ (similarly to Proposition 7) and thus $\bar{\bar{\lambda}}_{\downarrow}(t, \mathcal{T}) = \{id\}$, i.e., the parallel narrowing strategy $\bar{\bar{\lambda}}_{\downarrow}$ also performs a deterministic reduction step. In the presence of non-terminating rules, parallel narrowing does not subsume lazy narrowing with simplification as the following example shows.

Example 14 Consider the following non-terminating CAT:

$$\begin{array}{rll} f(0, X, Y) \to 0 & & \mathrm{R}_1 \\ f(X, 0, Y) \to 0 & & \mathrm{R}_2 \\ f(X, Y, 0) \to 0 & & \mathrm{R}_3 \\ g(X) \to g(s(X)) & & \mathrm{R}_4 \\ h(0) \to 0 & & \mathrm{R}_5 \end{array}$$

Let t be the term $f(g(X), h(Y), h(0))$ and $\mathcal{T}$ be a parallel definitional tree of f. Then $\bar{\bar{\lambda}}(t, \mathcal{T}) = \{id, \{Y \mapsto 0\}\}$, i.e., there are two parallel narrowing steps:

$$\begin{array}{lll} f(g(X), h(Y), h(0)) & \overset{\bar{\bar{\lambda}}}{\leadsto}_{id} & f(g(s(X)), h(Y), 0) \\ f(g(X), h(Y), h(0)) & \overset{\bar{\bar{\lambda}}}{\leadsto}_{\{Y \mapsto 0\}} & f(g(s(X)), 0, 0) \end{array}$$

Thus, parallel narrowing reduces t to 0 in two different ways. Lazy narrowing with simplification has a fully deterministic behavior and performs the following reduction steps:

$$f(g(X), h(Y), h(0)) \;\to\; f(g(X), h(Y), 0) \;\to\; 0$$

On the other hand, lazy narrowing with simplification has the same behavior as lazy narrowing if simplification is not applicable. For instance, consider the rules of Example 2 and the term $X \vee X$. Since no simplification rules are applicable, there are three possible lazy narrowing steps. However, parallel narrowing allows only two possible narrowing steps.

Although lazy narrowing with simplification and parallel narrowing are not directly comparable in the presence of non-terminating rules, parallel narrowing has an important advantage over all classic lazy narrowing strategies. Since the parallel narrowing strategy $\bar{\bar{\lambda}}$ computes only the identity substitution on ground terms, parallel narrowing evaluates ground goals in a fully deterministic way.

Moreover, it always computes the normal form of a ground term if it exists (see Theorem 8). On the other hand, lazy narrowing strategies perform non-deterministic steps even for ground goals. As a consequence, lazy narrowing may fail to compute normal forms in a sequential implementation. Thus, parallel narrowing is an ideal strategy for functional logic languages.

Parallel narrowing is not intended as a strategy to implement functional logic languages on parallel architectures. Although there is some potential for parallel implementation, the parallelism of this strategy is too fine-grained since the parallel reduction processes must be synchronized after each parallel narrowing step. This is in contrast to the AND-parallel narrowing implementation presented in [24] where independent subterms are evaluated in parallel. However, due to the fact that parallel narrowing reduces the number of non-deterministic choices in narrowing steps (compared to classic narrowing), parallel narrowing is useful to improve OR-parallel implementations of narrowing [16].

8 Conclusions

We have presented a new narrowing strategy for constructor-based weakly orthogonal rewrite systems. Since this class includes non-terminating systems, it is the basis of the functional component of many integrated functional logic languages. The main idea of our narrowing strategy is the parallel evaluation of necessary sets of redexes. This leads to a generalization of Sekar and Ramakrishnan's work on rewriting to narrowing. Parallel narrowing is a conservative extension of an optimal narrowing strategy, needed narrowing [3], to weakly orthogonal rewrite systems. Furthermore, parallel narrowing is the only known narrowing strategy for possibly non-terminating and overlapping rewrite rules which evaluates ground terms without any non-deterministic choice. This strategy can be implemented relatively efficiently, since narrowing steps are computed by local computations based on unification.[8] Therefore, this strategy is ideal for functional logic languages.

References

[1] S. Antoy. Definitional trees. In *Proc. of the 4th Intl. Conf. on Algebraic and Logic programming*, pages 143–157. Springer LNCS 632, 1992.

[2] S. Antoy, R. Echahed, and M. Hanus. A needed narrowing strategy. Technical report MPI-I-93-243, Max-Planck-Institut für Informatik, Saarbrücken, 1993.

[3] S. Antoy, R. Echahed, and M. Hanus. A needed narrowing strategy. In *Proc. 21st ACM Symposium on Principles of Programming Languages*, pages 268–279, Portland, 1994.

[4] 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.

[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.

[8]An implementation of parallel narrowing based on the compilation of functional logic programs into Prolog is described in [13].

[7] 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.

[8] 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.

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

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

[11] 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.

[12] 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.

[13] D. Genius. Comparison and implementation of narrowing strategies in Prolog (in German). Master's thesis, RWTH Aachen, 1996.

[14] 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.

[15] 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.

[16] W. Hans, F. Sáenz, and St. Winkler. Exploiting Expression- and Or-parallelism for a Functional Logic Language. In *Proc. GULP-PRODE*, Salerno (Italy), 1995. Abstract in Proc. PLILP'95, Springer LNCS 982, pages 457–458 (Poster Session).

[17] 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.

[18] M. Hanus. Combining lazy narrowing and simplification. In *Proc. of the 6th International Symposium on Programming Language Implementation and Logic Programming*, pages 370–384. Springer LNCS 844, 1994.

[19] M. Hanus. The integration of functions into logic programming: From theory to practice. *Journal of Logic Programming*, 19&20:583–628, 1994.

[20] 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.

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

[22] 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.

[23] 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.

[24] H. Kuchen, J.J. Moreno-Navarro, and M.V. Hermenegildo. Independent and-parallel implementation of narrowing. In *Proc. of the 4th International Symposium on Programming Language Implementation and Logic Programming*, pages 24–38. Springer LNCS 631, 1992.

[25] 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.

[26] R. Loogen and S. Winkler. Dynamic detection of determinism in functional logic languages. *Theoretical Computer Science 142*, pages 59–87, 1995.

[27] 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.

[28] 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.

[29] J.J. Moreno-Navarro, H. Kuchen, J. Marino-Carballo, S. Winkler, and W. Hans. Efficient lazy narrowing using demandedness analysis. In *Proc. of the 5th International Symposium on Programming Language Implementation and Logic Programming*, pages 167–183. Springer LNCS 714, 1993.

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

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

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

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

[34] R. C. Sekar and I. V. Ramakrishnan. Programming in equational logic: Beyond strong sequentiality. *Information and Computation*, 104(1):78–109, May 1993.

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

[36] 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.