Conditional Narrowing without Conditions*

Sergio Antoy

Department of Computer Science
Portland State University
P.O. Box 751, Portland, OR 97207, U.S.A.
antoy@cs.pdx.edu

Bernd Brassel Michael Hanus

Institut für Informatik
Christian-Albrechts-Universität Kiel
Olshausenstr. 40, D-24098 Kiel, Germany
Phone: +49-431/880-7271 / Fax: +49-431/880-7613
{bbr,mh}@informatik.uni-kiel.de

ABSTRACT

We present a new evaluation strategy for functional logic programs described by weakly orthogonal conditional term rewriting systems. Our notion of weakly orthogonal conditional rewrite system extends a notion of Bergstra and Klop and covers a large part of programs de£ned by conditional equations. Our strategy combines the ¤exibility of logic programming (computation of solutions for logic variables) with ef£cient evaluation methods from functional programming. In particular, it is the £rst known narrowing strategy for this class of programs that evaluates ground terms deterministically. This is achieved by a transformation of conditional term rewriting systems (CTRS) into unconditional ones which is sound and complete w.r.t. the semantics of the original CTRS. We show that the transformation preserves weak orthogonality for the terms of interest. This property allows us to apply a relatively ef£cient evaluation strategy for weakly orthogonal unconditional term rewriting systems (parallel narrowing) on the transformed programs.

Categories and Subject Descriptors

D.1.1 [**Programming Techniques**]: Applicative (Functional) Programming; D.1.6 [**Programming Techniques**]: Logic Programming; F.4.2 [**Mathematical Logic and Formal Languages**]: Grammars and other Rewriting Systems; I.2.2 [**Automatic Programming**]: Program Transformation

General Terms

Languages, Theory

Keywords

Functional Logic Programming, Narrowing, Evaluation Strategies, Conditional Rewriting

*This research has been partially supported by the DAAD/NSF grant INT-9981317, the German Research Council (DFG) grant Ha 2457/1-2 and the NSF grants CCR-0110496 and CCR-0218224.

In Proc. of the 5th International ACM SIGPLAN Conference on Principle and Practice of Declarative Programming (PPDP'03), pp. 20–31, Uppsala, Sweden, August 27–29, 2003.

1. MOTIVATION

Functional logic programming is an approach to combine the best features of functional and logic programming in a single programming model. A reasonable functional logic language should combine the (functional programming) idea of ef£cient evaluation by demand-driven and deterministic reduction, if possible, with the ¤exibility of logic programming by dealing with logic variables, partial data structures, and goal solving (see [18] for a survey on functional logic programming). Many language proposals (see, for instance, [18, 21, 26, 28, 35, 34]) have shown that such a combination is possible in principle. However, the choice of a "good" evaluation strategy is a crucial point in this combination. Simple *narrowing calculi* support completeness for large classes of programs but are often not an appropriate basis for a good evaluation strategy (compare also the discussion in [2] on this topic). On the other hand, sophisticated *narrowing strategies* have often stronger restrictions on the programs. For instance, innermost narrowing strategies require the termination of the rewrite relation in order to ensure completeness. Clearly, a termination requirement is too strong for practical programming since it inhibits the formulation of some programs (e.g., interpreters).[1] *Needed narrowing* [5] is a sound, complete, and optimal strategy for the class of inductively sequential programs [1] where functions are de£ned by *induction* on the term structure and can be evaluated *sequentially*. This class is also characterized as the strongly sequential constructor systems [20]. Although this covers a large part of typical declarative programs, it excludes functions de£ned by rules with overlapping left-hand sides (like the classical "parallel or" example). In order to support the latter kind of function de£nitions, *weakly needed narrowing* has been proposed [4, 27] which guesses the next reducible expression (*redex*) non-deterministically in case of functions de£ned by overlapping rules. The amount of nondeterminism is reduced in *parallel narrowing* [4] by guessing only variable instantiations and reducing disjoint outermost redexes in parallel. This has the positive effect that ground terms are evaluated in a deterministic manner. Weakly needed narrowing and parallel narrowing have been proved to be complete for the class of weakly orthogonal (see De£nition 10) constructor-based unconditional term rewriting systems (TRS).

In this paper we want to exploit the positive properties of parallel narrowing to develop a new strategy for *conditional* term rewriting

[1]Note that strict functional languages are based on innermost reduction (and some non-strict constructs like if-then-else) but do not have a termination requirement since they do not ensure completeness w.r.t. an equational interpretation of the de£ning rules. This requires a more careful construction of programs in comparison to languages with more sophisticated evaluation strategies, as discussed in [19].

systems (CTRS) that is deterministic on ground terms, i.e., inherits an important property from functional programming. Since we are interested in programming languages (rather than speci£cation languages), we do not require terminating TRSs but restrict ourselves to left-linear and constructor-based TRSs. Without always mentioning this explicitly, we assume that all considered TRSs belong to this class. The class of CTRSs is of considerable interest since it is often natural to express functions by the use of conditions as shown in the following example.

EXAMPLE 1. ***Conditional TRS for the absolute value***

$$\begin{array}{lll} abs(x) & \rightarrow & x \quad \textit{if } x \geq 0 \stackrel{*}{\rightarrow} \textit{true} \\ abs(x) & \rightarrow & -x \quad \textit{if } x < 0 \stackrel{*}{\rightarrow} \textit{true} \end{array}$$

The question is how to deal with the evaluation of conditions. In the deterministic world of functional programming, conditions are normally tested in a sequential if-then-else style. This works well for many functions, like abs. However, problems arise when the test of some (but not all) of the conditions might lead to in£nite computations. To understand this problem, refer to the next example.

EXAMPLE 2. ***A specialized multiplication***

$$\begin{array}{l} gmult(x,y) \rightarrow 0 \quad \textit{if } g(x) \stackrel{*}{\rightarrow} 0 \\ gmult(x,y) \rightarrow 0 \quad \textit{if } g(y) \stackrel{*}{\rightarrow} 0 \\ gmult(x,y) \rightarrow g(x) * g(y) \\ \qquad \textit{if } g(x) > 0 \wedge g(y) > 0 \stackrel{*}{\rightarrow} \textit{true} \end{array}$$

The function in Example 2 specializes the multiplication of terms that will be processed by the unary function g. It is not necessary to consider the details of g's de£nition. It is enough to remark that the computation of $g(x)$ can be long and tedious (or even in£nite) for some values of x. Thus, it makes sense to take a "shortcut" when the result of $g(x) * g(y)$ is known to be 0 before £nishing the evaluation for both arguments.[2] It is intended that $gmult$ has a well de£ned normal form if possible, even if one of the arguments is unde£ned (e.g., leads to an in£nite computation). Suppose that $g(0) \stackrel{*}{\rightarrow} 0$ and $g(\infty) \rightarrow g(\infty)$, *both* terms $gmult(0,\infty)$ and $gmult(\infty,0)$ should evaluate to 0.

It is easy to see that no sequential evaluation in a £xed order can obtain both results. Testing rule one £rst will lead into an in£nite computation with the term $gmult(\infty,0)$, while preferring the second rule does not lead to the desired result in the opposite case $gmult(0,\infty)$.

Logic programming implementations test conditions in a "don't know" nondeterministic manner. However, this does not solve the problem of possible in£nite computations but depends on the search strategy employed. Using a depth-£rst search (i.e., backtracking), the problem is identical to the functional case, since one has to decide which condition should be evaluated £rst. A breadth-£rst search gives rise to another problem. Beside being relatively expensive, a breadth-£rst search would still go on looking for possible answers after a £rst solution is found, thereby still computing in£nitely for *both* terms. Note that this in£nite computation is not necessary, as the £rst computed answer is the only interesting for $gmult$. This is due to the fact that $gmult$ is a *weakly orthogonal* CTRS (see De£nition 14 below).

[2]The apparent loss in ef£ciency because of computing $g(x)$ and $g(y)$ more than once can easily be remedied by employing a technique called *sharing* in the implementation of the functional logic language. Sharing allows one to de£ne that all the occurrences of $g(x)$ are identical (share the same value) consequently omitting repeated computations.

In this paper we present a different approach which avoids the described problems and yields the desired semantics. For this purpose, we will transform the conditional program into an unconditional one and then apply parallel narrowing to the transformed program. As mentioned above, parallel narrowing keeps the advantages of functional programming since the strategy works deterministically on ground terms. As we shall see, this advantage will also be present for conditional TRSs due to our transformation. The source systems of our transformation are the constructor-based conditional weakly orthogonal term rewriting systems. We impose an additional condition, De£nition 16, which does not actually restrict the source systems. We require that if two rules' left-hand sides overlap, then the rules' left-hand sides are equal (modulo a renaming of variables). Every source system can be easily transformed to satisfy this additional condition.

This paper is organized as follows. Section 2 gives de£nitions for some standard notions and clari£es our notation. Section 3 surveys some former work on transforming conditional TRSs to unconditional ones and then introduces our own transformation, followed by proofs of its key properties. In Sections 3 and 4 we apply our transformation to conditional functional logic programs and prove that important properties of the conditional programs will be preserved by the transformation. These properties enable us to apply parallel narrowing on the result of our transformation, as we will show in Section 5. Finally, Section 6 contains our conclusions along with further discussion of related work based on the results of sections 3-5.

2. PRELIMINARIES

The following de£nitions are quite standard but necessary to £x our notations. We de£ne term rewriting, both conditional and unconditional, and narrowing together with the necessary notions. We use the standard notations according to [7, 12, 25].

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

Next we de£ne the notion of a *constructor-based term rewriting system*. In such a TRS there is a distinction between reducible terms and pure data structures. This distinction is made by dividing the set of operation symbols of a signature into *constructors* and *de£ned operations*. Data structures are built with constructors (*constructor terms*) and the actual rewriting logic is formulated with de£ned operations.

DEFINITION 2. **(Constructor(-Rooted) Term, Pattern, Constructor-Based TRS)** *The set of operations* Ω *of a signature* Σ *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* de£ned 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 a* 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 *(*constructor-

rooted*) if* $f \in \mathcal{D}$ *(*$f \in \mathcal{C}$*). A* constructor-based term rewriting system *is a set of* rewrite rules, $l \to r$*, such that* l *and* r *have the same sort,* l *is a pattern, and* $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$. ◇

As we shall see in the next section, constructor-based TRSs are easier to handle than general TRSs, and consequently most functional logic programs are constructor-based.[3] Therefore, we de£ne a program as follows:

DEFINITION 3. **(Left-Linear TRS, Program)** *A constructor-based term rewriting system is called* left-linear *if for each of its rewrite rules* $l \to r$ *the pattern* l *is linear. A* program *is a left-linear constructor-based rewriting system. In the following,* $\mathcal{R}$ *denotes a program and* $\Sigma_{\mathcal{R}}$ *its signature.* ◇

The main purpose of this paper is to give a transformation for *conditional* term rewriting systems in order to apply parallel narrowing on the resulting unconditional TRS. We therefore have to introduce the syntax of CTRSs:

DEFINITION 4. **((Constructor-Based) Conditional TRS, (Un-) Conditional Rule)** *A* conditional term rewriting system *(CTRS) is a set of* rewrite rules *which have either the form "*$l \to r$ *if* $u \stackrel{*}{\to} v$*" or "*$l \to r$*" such that* l *and* r*,* u *and* v *have the same sort, respectively, and* $\mathcal{V}ar(r) \cup \mathcal{V}ar(u) \cup \mathcal{V}ar(v) \subseteq \mathcal{V}ar(l)$*. The rules of the form "*$l \to r$ *if* $u \stackrel{*}{\to} v$*" are called* conditional rules*, the remaining rules are called* unconditional*. In the following, CTRSs will be denoted as* $\mathfrak{R}$*.*[4] *If for each rule in* $\mathfrak{R}$ l *is a pattern and for each conditional rule in* $\mathfrak{R}$ v *is a ground constructor term, then* $\mathfrak{R}$ *is called* constructor-based*.*
In this paper, we will exclusively be concerned with left-linear CTRSs. ◇

Our de£nition of conditional TRSs seems rather restricted. However, a few considerations show that the restrictions are not too serious: First, the de£nition allows only a single condition for each rule. Using tuples, multiple conditions of the form

$$l \to r \text{ if } u_0 \stackrel{*}{\to} v_0 \wedge \ldots \wedge u_n \stackrel{*}{\to} v_n$$

can easily be expressed with a single condition

$$l \to r \text{ if } (u_0, \ldots, u_n) \stackrel{*}{\to} (v_0, \ldots, v_n)$$

Second, the restriction to conditions where a term is tested for reducibility to a ground constructor term is also reasonable since it covers the equation solving capabilities of current functional logic languages with a lazy operational semantics, like Curry [21] or TOY [28]. These languages are based on *strict equality* "$\approx$" tests in conditions ($t_1 \approx t_2$ holds if t_1 and t_2 are reducible to the same ground constructor term). Strict equality can be de£ned as a binary operation by a set of orthogonal rewrite rules (e.g., see [5]). In this case, a conditional rule with a strict equality condition "$u \approx v$" can be expressed as $l \to r$ *if* $(u \approx v) \stackrel{*}{\to} true$. Similarly, non-left-linear rules $f(x, x) \to r$ can be transformed into $f(x, y) \to r$ *if* $x \approx y \stackrel{*}{\to} true$. For a discussion that this elimination of non-linearity makes good sense cf. [2, section 4.1].

[3] See, for instance, the Equational Interpreter [32] and the functional logic languages *ALF* [17], *BABEL* [31], *Curry* [21], *K-LEAF* [15], *LPG* [10], *SLOG* [14], and *TOY* [28].

[4] There is a reason for the notation of $\mathcal{R}$ for an unconditional and $\mathfrak{R}$ for a conditional TRS and it is thus easy to memorize: Both Rs stand for "rewriting system" and choosing the old style letter to denote a CTRS is a metaphor for our aim to transform TRSs into CTRSs; in the ¤ow of time old things change to new ones and $\mathfrak{R}$ becomes $\mathcal{R}$.

Finally, it will be discussed in the last section that omitting extra variables for local de£nitions like let-clauses (cf. [33]) is not a restriction, since let-clauses can be eliminated by standard lambda lifting [24].

We now turn to the semantics of TRSs, but we need a few more notions to de£ne rewriting. One of the key operations on terms is *substitution*:

DEFINITION 5. **((Constructor) Substitution, Instance, Variant, Uni£er)** *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* $x \in X_s$ *such that its* domain $\mathcal{D}om(\sigma) = \{x \in \mathcal{X} \mid \sigma(x) \neq x\}$ *is £nite. We frequently identify a substitution* σ *with the set* $\{x \mapsto \sigma(x) \mid x \in \mathcal{D}om(\sigma)\}$*. Substitutions are extended to morphisms on* $\mathcal{T}(\Sigma, \mathcal{X})$ *by* $\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$ *for every term* $f(t_1, \ldots, t_n)$*. A substitution* σ *is called a* constructor substitution *if* $\sigma(x)$ *is a constructor term for all* $x \in \mathcal{D}om(\sigma)$*. A term* t' *is an* instance *of* t *if there is a substitution* σ *with* $t' = \sigma(t)$*. If* t *is an instance of* t' *and vice versa, we call* t' *a* variant *of* t*. A* uni£er *of two terms* s *and* t *is an idempotent substitution* σ *with* $\sigma(s) = \sigma(t)$. ◇

When dealing with terms, one wants to be able to identify unambiguously each of its parts (subterms). This is done by de£ning the *positions* of a term:

DEFINITION 6. **(Occurrence, Position,** Λ**,** $t|_p$**,** $t[s]_p$**,** $p \leq q$**,** $p \parallel q$**,** $p \cdot q$**)** *An* occurrence *or* position *is a sequence of positive integers identifying a subterm in a term. For every term* t*, the empty sequence is denoted by* Λ *and identi£es* 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, identi£es the subterm of* t_i *at* p*. The subterm of* t *at* p *is denoted by* $t|_p$ *and the result of* replacing $t|_p$ *with* s *in* t *is denoted by* $t[s]_p$*. If* p *and* q *are positions, we write* $p \leq q$ *if* p *is a* pre£x *of* q *and we say that* p *is* above q*. We write* $p \parallel q$ *if the positions are* disjoint*, i.e., none of them is above the other (see [12] for details). The expression* $p \cdot q$ *denotes the position resulting from the concatenation of the positions* p *and* q*, i.e., the symbol "*$\cdot$*" is overloaded.* ◇

Now we are ready to de£ne rewriting.

DEFINITION 7. **(Reduction** $t \to_{p,R} s$**, Redex,** $\stackrel{*}{\to}$**, Normal Form)** *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.* $\stackrel{*}{\to}$ *denotes the transitive and re¤exive closure of* $\to$*. A term* t *is* reducible to *a term* s *if* $t \stackrel{*}{\to} s$*. 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*. When considering two TRSs* $\mathcal{R}$ *and* $\mathcal{R}'$ *simultaneously, we write* $\stackrel{*}{\to}_{\mathcal{R}}$ *and* $\stackrel{*}{\to}_{\mathcal{R}'}$ *to distinguish the rewrite relation de£ned by* $\mathcal{R}$ *and* $\mathcal{R}'$*, respectively.* ◇

Rewriting in conditional TRSs is de£ned by extending unconditional rewriting by the evaluation of conditions:

DEFINITION 8. **(Conditional Rewriting)** *The* rewrite relation $\stackrel{*}{\to}_{\mathfrak{R}}$ *for a CTRS* $\mathfrak{R}$ *is de£ned by induction:*
$\to_{\mathfrak{R}_0} = \to_{\mathcal{R}}$*, where* $\mathcal{R}$ *is the set of unconditional rules in* $\mathfrak{R}$*.*
$s \to_{\mathfrak{R}_{n+1}} t$ *iff either* $s \to_{\mathfrak{R}_n} t$ *or there exists a rule* $l \to r$ *if* $u \stackrel{*}{\to} v$ *in* $\mathfrak{R}$ *with* $s \to_{l \to r} t$ *and* $u \stackrel{*}{\to}_{\mathfrak{R}_n} v$*.*
Then $s \stackrel{*}{\to}_{\mathfrak{R}} t$ *iff there is a natural number* n *with* $s \stackrel{*}{\to}_{\mathfrak{R}_n} t$*.* ◇

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 logic 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 [15, 31]. This is done by narrowing.

DEFINITION 9. **(Narrowing $t \leadsto_{p,l\to r,\sigma} s$, $t \overset{*}{\leadsto}_{\sigma} s$)** *A term t is* narrowable *to a term s if there exist a non-variable position p of t (i.e., $t|_p \notin \mathcal{X}$), a variant $l \to r$ of a rewrite rule in $\mathcal{R}$ with $\mathcal{V}ar(t) \cap \mathcal{V}ar(l \to r) = \emptyset$ and a uni£er σ of $t|_p$ and l such that $s = \sigma(t[r]_p)$. In this case we write $t \leadsto_{p,l\to r,\sigma} s$.* ◇

Finally, we need the notion of orthogonal programs:

DEFINITION 10. **((Weak) Orthogonality, Overlapping Rule)** *A program $\mathcal{R}$ is called* orthogonal *if, for all distinct rules $l_1 \to r_1$ and $l_2 \to r_2$, chosen with no variables in common, there is no uni£er for l_1 and l_2. If there is a uni£er of l_1 and l_2, the rules are called* overlapping. *$\mathcal{R}$ is called* weakly orthogonal *if all uni£ers of l_1 and l_2 are also uni£ers of r_1 and r_2.*[5] ◇

We will transfer the notion of (weak) orthogonality to conditional TRSs. Since our notion is not standard, we will present the de£nition later in Section 4.

3. TRANSFORMATION OF CONDITIONAL TERM REWRITING SYSTEMS

A lot of research has been invested in conditional TRSs, particularly, in attempts to transform them into unconditional ones. Early work has been done by Bergstra and Klop [8]. The transformation presented there has been later shown to be unsound in [13] (see also [36, 11]). Further transformations were introduced in [16] and [22] but they demand terminating conditional TRSs. As mentioned in Section 1, a termination requirement is too restrictive for functional logic programming.

In [9] it was shown that CTRSs have in a sense more expressive power since they are more compact: an equivalent unconditional system has to use more function symbols or function symbols with greater arity to express the same content. Nevertheless, it is possible to construct an equivalent (see Theorems 1 and 3 to see in detail what the notion "equivalence" means in this context) TRS for a given CTRS. It is the aim of this section to introduce such a transformation for constructor-based left-linear CTRSs. Further related work will be discussed in the last section when we have some results to remark upon.

Antoy [2] has presented a transformation for arbitrary constructor-based CTRSs into the class of "overlapping inductively sequential" TRSs. These are inductively sequential TRSs with possibly multiple right-hand sides for a single rule. Since this transformation maps different conditional rules into multiple right-hand sides that are non-deterministically evaluated, the transformation does not lead to a deterministic evaluation of ground terms. Therefore, our transformation is based on the idea to join all conditions into a single structure. This idea was proposed by Viry [36] for general CTRSs so that we present his transformation £rst:

DEFINITION 11. **(Transformation of Viry, t^X, $t^\perp$, $\bar{t}$, Instantiating Rule, Reducing Rule)** *Let $\mathfrak{R}$ be a conditional term rewriting system. For each n-ary de£ned function $f \in \mathfrak{R}$ let $\rho_{f,1}, \ldots, \rho_{f,m}$ be the conditional and $\rho_{f,m+1}, \ldots, \rho_{f,m+k}$ the unconditional rules de£ning f in $\mathfrak{R}$. Each de£ned n-ary function f is replaced by an $(n+m)$-ary function f (i.e., for each conditional rule an argument is added) and the new set of unconditional rules $\mathcal{R}_\mathcal{V}$ contains for each conditional rule*

$$\rho_{f,i} : f(t_1, \ldots, t_n) \to r_{f,i} \text{ if } u_{f,i} \overset{*}{\to} v_{f,i}$$

two unconditional rules

$$\begin{aligned}\rho'_{f,i} : f(t_1^X, \ldots, t_n^X \mid y_1, \ldots, \quad \perp_i, \quad \ldots, y_m)\\ \to f(t_1^X, \ldots, t_n^X \mid y_1, \ldots, [u_{f,i}^\perp, \vec{x}], \ldots, y_m)\end{aligned}$$

(the notation $\perp_i$ denotes $\perp$ at argument i) and

$$\rho''_{f,i} : f(z_1, \ldots, z_n \mid y_1, \ldots, [v_{f,i}^X, \vec{x}], \ldots, y_m) \to r_{f,i}^\perp,$$

where $\vec{x} = (x_1, \ldots, x_j)$ if $\mathcal{V}ar(r_{f,i}) = \{x_1, \ldots, x_j\}$, for some arbitrary, but £xed, ordering of the variables. For replacing the unconditional rules, one needs new variables. X denotes an in£nite set of fresh variables. Each unconditional rule $\rho_{f,i} : f(t_1, \ldots, t_n) \to r_{f,i}$ is replaced by the rule

$$\rho''_{f,i} : f(t_1^X, \ldots, t_n^X \mid y_1, \ldots, y_m) \to r_{f,i}^\perp$$

The new conditional arguments are £lled via the mappings t^X and $t^\perp$ of type $\mathcal{T}(\Sigma_\mathfrak{R}, X) \to \mathcal{T}(\Sigma_{\mathcal{R}_\mathcal{V}}, X)$. t^X and $t^\perp$ put fresh variables and the symbol $\perp$ at these new arguments, respectively:

$$t^X = \begin{cases} t & \text{if } t \in X \\ f(t_1^{X_1}, \ldots, t_n^{X_n} \mid y_1, \ldots, y_m) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

where $X_1, \ldots, X_n, \{y_1, \ldots, y_m\}$ are pairwise disjoint subsets of X with $X_1, \ldots, X_n$ in£nite.

$$t^\perp = \begin{cases} t & \text{if } t \in X \\ f(t_1^\perp, \ldots, t_n^\perp \mid \perp, \ldots, \perp) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

The symbol $\perp$ is a constructor not in Σ and the grouping of terms with parentheses and square brackets is only syntactic sugar to avoid the introduction of new symbols. The mapping $\bar{t} : \mathcal{T}(\Sigma_{\mathcal{R}_\mathcal{V}}, X) \to \mathcal{T}(\Sigma_\mathfrak{R}, X)$ deletes the conditional arguments in a term t:

$$\bar{t} = \begin{cases} t & \text{if } t \in X \\ f(\bar{t_1}, \ldots, \bar{t_n}) & \text{if } t = f(t_1, \ldots, t_n \mid s_1, \ldots, s_m) \end{cases}$$

The ρ'-rules will be called instantiating rules, *because they instantiate the new conditional positions, and the ρ''-rules will be called* reducing rules, *because they correspond to the reduction steps of the original system.* ◇

EXAMPLE 3. *Transforming the rules of Example 1 leads to the following TRS:*

$$\begin{aligned} abs(x \mid \perp, y_2) &\to abs(x \mid [x \geq 0, (x)], y_2) \\ abs(x \mid y_1, \perp) &\to abs(x \mid y_1, [x < 0, (x)]) \\ abs(y_0 \mid [true, (x)], y_2) &\to x \\ abs(y_0 \mid y_1, [true, (x)]) &\to -x \end{aligned}$$

The following two theorems by Viry are of interest for the remaining paper.

THEOREM 1. **(Viry [36])** *The transformation $\mathcal{V}$ is sound and complete for all conditional rewrite systems, i.e.,*

1. *for each derivation $t \overset{*}{\to}_\mathfrak{R} t'$ there is a derivation $t^\perp \overset{*}{\to}_{\mathcal{R}_\mathcal{V}} t'^\perp$ (completeness),*

[5] Note that our de£nition of (weak) orthogonality makes use of the restriction to constructor-based TRSs. In such systems overlapping of different rules can only occur at root position.

2. *for each derivation* $t^\perp \xrightarrow{*}_{\mathcal{R}_\mathcal{V}} t'$ *there is a derivation* $t \xrightarrow{*}_{\mathfrak{R}} \bar{t}'$ *(soundness).*

Note that, by de£ning soundness this way, it is implied that only a certain part of the target system $\mathcal{R}_\mathcal{V}$ is relevant for simulating $\mathfrak{R}$.[6] If arbitrary terms of the target system are considered, soundness can not be ensured. For instance, in the system of Example 3, the term $abs(x \mid [true, (x)], [true, (x)])$ is contained in $\mathcal{T}(\Sigma_{\mathcal{R}_\mathcal{V}}, X)$. It is immediate to see that this term could cause problems in the context of the source system since it denotes that both $x \geq 0$ and $x < 0$ could be evaluated to $true$. Therefore, we de£ne the *terms of interest*:

DEFINITION 12. **(Terms of Interest, Bottom Term)** *Let* $\mathcal{R}_\mathcal{V}$ *be the result of applying Viry's transformation to a CTRS* $\mathfrak{R}$*. Then the* terms of interest $\mathcal{T}_{\mathcal{R}_\mathcal{V}} \subset \mathcal{T}(\Sigma_{\mathcal{R}_\mathcal{V}}, \mathcal{X})$ *are all terms with uninstantiated conditional positions (*bottom terms*) closed under derivation:* $\mathcal{T}_{\mathcal{R}_\mathcal{V}} := \{u \mid \exists t \in \mathcal{T}(\Sigma_\mathfrak{R}, \mathcal{X}) \wedge t^\perp \xrightarrow{*} u\}$. $\diamond$

It is easy to see that this de£nition excludes the example $abs(x \mid [true, (x)], [true, (x)])$ (unless in the semantics of the source system both $x \geq 0$ and $x < 0$ can really be evaluated to $true$ for some x).

The following proposition is important for the use of terms of interest:

PROPOSITION 1. *For each* $t \in \mathcal{T}_{\mathcal{R}_\mathcal{V}}$ *each subterm of* t *is also in* $\mathcal{T}_{\mathcal{R}_\mathcal{V}}$ *and for each substitution* σ *which replaces variables only with terms of interest,* $\sigma(t)$ *is also in* $\mathcal{T}_{\mathcal{R}_\mathcal{V}}$*.*

An interesting question is the preservation of con¤uence, i.e., given a con¤uent $\mathfrak{R}$, is $\mathcal{R}_\mathcal{V}$ con¤uent, too? This cannot be ensured in general, as shown in the following example.

EXAMPLE 4. *Consider the con¤uent CTRS* $\mathfrak{R}$ *de£ned by the rules*

$$\begin{array}{lcll} f(g(x)) & \to & x & \text{if } x \xrightarrow{*} 0 \\ g(g(x)) & \to & g(x) & \end{array}$$

The corresponding TRS $\mathcal{R}_\mathcal{V}$ *is*

$$\begin{array}{lcl} f(g(x) \mid \perp) & \to & f(g(x) \mid [x, (x)]) \\ f(y_0 \mid [0, (x)]) & \to & x \\ g(g(x)) & \to & g(x) \end{array}$$

Note that $g(x)^X = g(x)$ *since no rule de£ning* g *is conditional. Now there is an* $\mathcal{R}_\mathcal{V}$*-derivation yielding two different normal forms (contracted redexes are underlined):*

$$\begin{array}{lll} f(\underline{g(g(0))} \mid \perp) & \to f(\underline{g(0) \mid \perp}) & \\ & \to \underline{f(g(0) \mid [0, (0)])} & \\ & \to 0 & \\ \\ \underline{f(g(g(0)) \mid \perp)} & \to f(\underline{g(g(0))} \mid [g(0), (g(0))]) & \\ & \to f(g(0) \mid [g(0), (g(0))]) & \end{array}$$

Both derivations are attempts to simulate the $\mathfrak{R}$*-derivation* $f(g(g(0))) \to f(g(0)) \to 0$ *but the last step can be simulated only by the £rst* $\mathcal{R}_\mathcal{V}$*-derivation.*

Viry suggests to solve the problem by restricting the possible derivations in $\mathcal{R}_\mathcal{V}$. By evaluating the conditional positions £rst, he wants to ensure con¤uence. In terms of [36], such a derivation is called *conditional eager* (cf. [36, De£nition 4.1]). The second derivation of Example 4 is not conditional eager, as the term $g(g(0))$ is rewritten although the conditional positions are already instantiated.

Restricting the possible derivations in this way, Viry presents a theorem about preserving con¤uence:

THEOREM 2. **($\perp$-Con¤uence [36])** *If in* $\mathcal{R}_\mathcal{V}$ *all terms derived from* $\perp$*-terms by conditional eager derivations (i.e., proving conditions £rst) can be joined w.r.t.* $^-$*, then* $\mathcal{R}_\mathcal{V}$ *is called* $\perp$-con¤uent*. If* $\mathfrak{R}$ *is con¤uent, then* $\mathcal{R}_\mathcal{V}$ *is* $\perp$*-con¤uent.*

Unfortunately, this theorem does not hold: Revisiting Example 4 but taking *only the £rst step* of the second derivation, i.e.,

$$\underline{f(g(g(0)) \mid \perp)} \to f(g(g(0)) \mid [g(0), (g(0))])$$

one can see that the derivation *so far* is trivially conditional eager (cf. [36, De£nition 4.1]), because there is no remaining derivation which could rewrite an unconditional position of $g(g(0))$. But it is already too late: the resulting term cannot be joined with its counterpart $f(g(0) \mid \perp)$.

The main problem of the system in Example 4 is that the rules are not constructor-based. However, for functional logic programming, the class of considered systems can be further restricted since we are interested in the class of weakly orthogonal, constructor-based CTRSs.[7] We want to show that a slightly modi£ed transformation preserves con¤uence for this class. We £rst show that the target system of a constructor-based CTRS is also constructor-based.

LEMMA 1. *If* $\mathfrak{R}$ *is constructor-based, then* $\mathcal{R}_\mathcal{V}$ *is constructor-based.*

PROOF. The signatures of $\mathfrak{R}$ and $\mathcal{R}_\mathcal{V}$ differ only in the arity of the functions and a few new symbols without de£ning rules. In particular, $\perp$ is a new constructor and the pair notation $[u, (\vec{x})]$ is only intended to ease reading. Instead of the $m + k$ rules of $\mathfrak{R}$

$$\begin{array}{l} \rho_{f,i} : f(l_{f,i_1}, \ldots, l_{f,i_n}) \to r_{f,i} \text{ if } u_{f,i} \xrightarrow{*} v_{f,i} \\ \rho_{f,j} : f(l_{f,j_1}, \ldots, l_{f,j_n}) \to r_{f,j}, \end{array}$$

where $1 \leq i \leq m$ and $m < j \leq m + k$, we now have $2m + k$ rules of the form

$$\begin{array}{ll} \rho'_{f,i} : f(t_1^X, \ldots, t_n^X & \mid y_1, \ldots, \quad \perp_i, \quad \ldots, y_m) \\ \quad \to f(t_1^X, \ldots, t_n^X & \mid y_1, \ldots, [u_{f,i}^\perp, \vec{x}], \ldots, y_m) \\ \rho''_{f,i} : f(z_1, \ldots, z_n & \mid y_1, \ldots, [v_{f,i}^X, \vec{x}], \ldots, y_m) \to r_{f,i}^\perp \\ \rho''_{f,j} : f(l_{f,j_1}^X, \ldots, l_{f,j_n}^X & \mid y_1, \quad \ldots, \quad y_m) \to r_{f,j}^\perp, \end{array}$$

where $1 \leq i \leq m$ and $m < j \leq m+k$. According to the de£nition of a constructor-based CTRS, the terms $v_{f,i}$ are ground constructor terms (hence, $v_{f,i}^X = v_{f,i}$). As $\perp$ is a constructor symbol, for every pattern l, the terms l^X and $l^\perp$ are also patterns. Therefore, each left-hand side of $\mathcal{R}_\mathcal{V}$ is a pattern. $\square$

It is easy to see that $\mathcal{R}_\mathcal{V}$ *is left linear if* $\mathfrak{R}$ *is left linear*, since all the variables introduced by ^{X} are fresh by de£nition. Now we will optimize the transformation to the case of interest of this paper. First we consider the variable vector $\vec{x}$. Saving the context in which a condition was instantiated is essential for the soundness of the transformation, as shown in Example 6 below. However, we will show that saving the context can be avoided if the source CTRS is constructor-based and left linear. Furthermore, some of the rules can be joined in a single one as de£ned by the following transformation. The latter improvement is crucial for our narrowing strategy.

[6] This is not an accidental property of Viry's transformation but a consequence of the result that CTRSs are more compact than TRSs, as mentioned at the beginning of this section and shown in [9].

[7] In [11] it is also shown that Viry's transformation can be extended to preserve con¤uence in general. However, we omit this result since it is not needed in the context of this paper.

DEFINITION 13. **(Transformation $\mathcal{J}$)** *Let $\mathfrak{R}$ be a constructor-based CTRS whose rules are either of the form*

$$\rho_{f,i} : f(l_1,\ldots,l_n) \to r_{f,i} \text{ if } u_{f,i} \stackrel{*}{\to} v_{f,i} \quad \text{or}$$
$$\rho_{f,i} : f(l_1,\ldots,l_n) \to r_{f,i}$$

We de£ne $\mathcal{R}_{\mathcal{V}'}$ as the unconditional TRS that contains the following two rules for each conditional rule of the previous form, where m is the number of conditional rules for f in $\mathfrak{R}$:

$$\begin{array}{rl}\pi_{f,i} : & f(l_1,\ldots,l_n \mid y_1,\ldots,\bot_i,\ldots,y_m) \\ & \to f(l_1,\ldots,l_n \mid y_1,\ldots,u^{\bot}_{f,i},\ldots,y_m) \\ \pi''_{f,i} : & f(l_1,\ldots,l_n \mid y_1,\ldots,v_{f,i},\ldots,y_m) \to r^{\bot}_{f,i}.\end{array}$$

For each unconditional rule of $\mathfrak{R}$, $\mathcal{R}_{\mathcal{V}'}$ contains

$$\pi''_{f,i} : f(l_1,\ldots,l_n \mid y_1,\ldots,y_m) \to r^{\bot}_{f,i}.$$

The set of rules of $\mathcal{R}_{\mathcal{J}}$ is de£ned by joining all π-rules of $\mathcal{R}_{\mathcal{V}'}$ with identical left-hand sides. Let $I = \{i_1,\ldots,i_k\}$ be the set of all rules in $\mathcal{R}_{\mathcal{V}'}$ of the form

$$\begin{array}{rl}\pi_{f,i_1} : & f(l_1,\ldots,l_n \mid \ldots,\ \bot_{i_1},\ldots,y_{i_k},\ldots) \\ & \to f(l_1,\ldots,l_n \mid \ldots,u_{f,i_1},\ldots,y_{i_k},\ldots) \\ & \vdots \qquad\qquad \vdots \qquad\quad \vdots \\ \pi_{f,i_k} : & f(l_1,\ldots,l_n \mid \ldots,\ y_{i_1},\ldots,\bot_{i_k},\ldots) \\ & \to f(l_1,\ldots,l_n \mid \ldots,\ y_{i_1},\ldots,u_{f,i_k},\ldots).\end{array}$$

Instead of these rules, $\mathcal{R}_{\mathcal{J}}$ contains a single rule

$$\begin{array}{rl}\pi'_{f,I} : & f(l_1,\ldots,l_n \mid \ldots,\ \bot_{i_1},\ldots,\bot_{i_k},\ldots) \\ & \to f(l_1,\ldots,l_n \mid \ldots,u_{f,i_1},\ldots,u_{f,i_k},\ldots).\end{array}$$

The mappings t^X, $t^{\bot}$ and $\overline{t}$ as well as the notions of instantiating *and* reducing rule, bottom term, *and the de£nition of the* terms of interest *carry over from $\mathcal{R}_{\mathcal{V}}$ to $\mathcal{R}_{\mathcal{J}}$ without dif£culty.* ◇

Since the above de£nition applies to constructor-based CTRSs, by contrast to De£nition 11 it is not necessary to transform the subterms in the left-hand sides of the source systems. For constructor terms, $l_i^X = l_i$.

EXAMPLE 5. *Applying transformation $\mathcal{J}$ to the rules of Example 1, we obtain the following system:*

$$\begin{array}{lcl} abs(x \mid \bot,\bot) & \to & abs(x \mid x \geq 0, x < 0) \\ abs(x \mid true, y_2) & \to & x \\ abs(x \mid y_1, true) & \to & -x \end{array}$$

However, the transformation is not sound for arbitrary CTRSs.

EXAMPLE 6. *Consider the following (non-con¤uent) set of conditional rules:*

$$\begin{array}{lcl} f(g(x)) & \to & x \quad \text{if } x \stackrel{*}{\to} s(0) \\ g(s(x)) & \to & g(x) \end{array}$$

The system is transformed to

$$\begin{array}{lcl} f(g(x) \mid \bot) & \to & f(g(x) \mid x)) \\ f(g(x) \mid s(0)) & \to & x \\ g(s(x)) & \to & g(x). \end{array}$$

The derivation $\underline{f(g(s(0)) \mid \bot)} \to f(\underline{g(s(0))} \mid s(0)) \to \underline{f(g(0) \mid s(0))} \to 0$ is not sound, since, in the source system, $f(g(s(0)))$ is reducible to both the normal forms $s(0)$ and $f(g(0))$ but not to 0.

It is easy to see that the problem can not occur with Viry's transformation. Using the variable vector $\vec{x}$, the last step would have been $f(g(0) \mid [s(0),(s(0))]) \to s(0)$. As mentioned above, we will prove that there is a simple reason why this system and the one of Example 4 are problematic: both systems are not constructor-based.

THEOREM 3. *For any constructor-based, left-linear CTRS $\mathfrak{R}$ the transformation $\mathcal{J}$ is complete and sound, i.e.,*

1. *for each derivation $t \stackrel{*}{\to}_{\mathfrak{R}} t'$ there is a derivation $t^{\bot} \stackrel{*}{\to}_{\mathcal{R}_{\mathcal{J}}} t'^{\bot}$ (completeness),*
2. *for each derivation $t^{\bot} \stackrel{*}{\to}_{\mathcal{R}_{\mathcal{J}}} t'$ there is a derivation $t \stackrel{*}{\to}_{\mathfrak{R}} \overline{t'}$ (soundness).*

PROOF. (Completeness)
We show completeness by induction over the *actual length* $\mathfrak{L}(t \stackrel{*}{\to} t')$ of the derivations $t \stackrel{*}{\to}_{\mathfrak{R}} t'$ which is de£ned as follows:

$$\begin{array}{lcl} \mathfrak{L}(t \to_{l\to r} t') & = & 1 \\ \mathfrak{L}(t \to_{l\to r \text{ if } u \stackrel{*}{\to} v} t') & = & 1 + \mathfrak{L}(u \stackrel{*}{\to} v) \\ \mathfrak{L}(t_0 \to t_1 \to \ldots \to t_n) & = & \sum_{i=1}^{n} \mathfrak{L}(t_{i-1} \to t_i) \end{array}$$

Base case ($\mathfrak{L}(t \stackrel{*}{\to} t') = 0$): The claim trivially holds for all derivations with an actual length of zero, since $t = t'$ implies $t^{\bot} = t'^{\bot}$ and, thus, $t^{\bot} \stackrel{*}{\to} t'^{\bot}$.
Inductive case: Suppose that the claim holds for all derivations of the actual length $\leq \mathfrak{l}$. We have to show that it holds for all derivations with length $\mathfrak{l}+1$. Consider the derivation $t \stackrel{*}{\to} s \to_{\rho,p} t'$. By the induction hypothesis, there exists a derivation $t^{\bot} \stackrel{*}{\to} s^{\bot}$. We now distinguish whether ρ is an unconditional or a conditional rule. For an unconditional rule $\rho = f(l_1 \ldots l_n) \to r \in \mathfrak{R}$, there exists a reducing rule $\rho'' : f(l_1 \ldots l_n \mid y_1 \ldots y_m) \to r^{\bot} \in \mathcal{R}_{\mathcal{J}}$. For this rule the reduction $s^{\bot} \to_{\rho'',p} t'^{\bot}$ yields the last step we need.[8] If $\rho = f(l_1,\ldots,l_n) \to r \text{ if } u \stackrel{*}{\to} v$ is the i-th conditional rule de£ning f in $\mathfrak{R}$, then, by de£nition of $\mathfrak{L}$, the actual length of $u \stackrel{*}{\to} v$ is $\leq \mathfrak{l}$ and, by induction hypothesis, there exists a derivation $\alpha : u^{\bot} \stackrel{*}{\to}_{\mathcal{R}_{\mathcal{J}}} v (= v^{\bot})$. Using the instantiating rule ρ' and the reducing rule ρ'' corresponding to ρ in $\mathcal{R}_{\mathcal{J}}$, we can construct the derivation $s^{\bot} \to_{\rho',p} s' \stackrel{*}{\to}_{p\cdot(n+i)\cdot\alpha} s'' \to_{\rho'',p} t'^{\bot}$, where $p\cdot(n+i)\cdot\alpha$ means that the derivation α is inserted and performed at the position $p\cdot(n+i)$.
(Soundness)
We de£ne a mapping $\mu : \mathcal{T}_{\mathcal{R}_{\mathcal{J}}} \to \mathcal{T}_{\mathcal{R}_{\mathcal{V}}}$ between the terms of interest of $\mathcal{R}_{\mathcal{J}}$ and $\mathcal{R}_{\mathcal{V}}$:

$$\mu(t) = \begin{cases} t & \text{if } t = \overline{t}^{\bot}, \\ f(\mu(t_1),\ldots,\mu(t_n) \mid s'_1,\ldots,s'_m) & \\ \qquad \text{if } t = f(t_1,\ldots,t_n \mid s_1,\ldots,s_m), & \end{cases}$$

where $s'_i = \bot$ if $s_i = \bot$. Otherwise, $s'_i = [\mu(s_i), \sigma(\vec{x})]$ where σ is the matching of the term $f(t_1,\ldots,t_n \mid \bot,\ldots,\bot)$ and the left-hand side of the rule $\pi'_{f,i}$ of $\mathcal{R}_{\mathcal{J}}$.[9]
We will prove that any derivation $t^{\bot} \stackrel{*}{\to}_{\mathcal{R}_{\mathcal{J}}} t'$ implies the existence of a derivation $t^{\bot} \stackrel{*}{\to}_{\mathcal{R}_{\mathcal{V}}} \mu(t')$ by induction over the length l of the derivation $t^{\bot} \stackrel{*}{\to}_{\mathcal{R}_{\mathcal{J}}} t'$.

[8] Note that any position in $t \in \mathcal{T}(\Sigma_{\mathfrak{R}}, X)$ is also a position in $t^{\bot}$ since the new conditional positions are added after the original unconditional positions.
[9] This matching must (still) exist since in constructor-based and left-linear systems the matching cannot be destroyed by reductions below patterns.

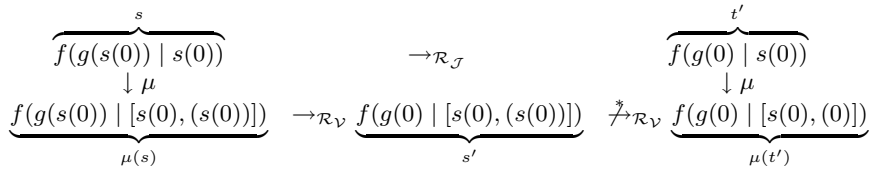


Figure 1: Sample derivations from Example 6

$l = 0$: The claim holds since $\mu(t^\perp) = t^\perp$.
Inductive case:
Consider the last step of the derivation $t^\perp \xrightarrow{*} s \rightarrow_{p,\pi} t'$.
a) If π is an instantiating rule π', then $\mu(s)$ can be reduced to $\mu(t')$ by applying all of the corresponding ρ'-rules of $\mathcal{R}_\mathcal{V}$ joined in π' in an arbitrary order.
b) The remaining case is that π is a reducing rule π''. By construction of $\mathcal{R}_\mathcal{J}$, there is a corresponding rule ρ'' in $\mathcal{R}_\mathcal{V}$ which can be applied at a position p' similar to p in $\mu(s)$.[10] We consider the result s' of the reduction $\mu(s) \rightarrow_{p',\rho''} s'$. How far can $\mu(t')$ and s' differ? The only possible difference is that the mapping μ might instantiate variables in the vectors $\vec{x}$ differently in s and t'. This causes only problems (i.e., leads to terms out of the terms of interest) if $\mathfrak{R}$ is not constructor-based, see, for example, the derivations in Figure 1.
Since $\mathfrak{R}$ is constructor-based, any reduction at an unconditional position must be possible within the corresponding variable vectors as well, because the redexes copied when the vectors are instantiated cannot be destroyed.[11] Therefore, the same rule π'' can be applied below all the variables in the vector in arbitrary order, leading to $\mu(s') \xrightarrow{*} \mu(t')$.
Hence, we have shown that $t^\perp \xrightarrow{*}_{\mathcal{R}_\mathcal{J}} t'$ implies the existence of a derivation $t^\perp \xrightarrow{*}_{\mathcal{R}_\mathcal{V}} \mu(t')$. By de£nition of μ, $\overline{\mu(t)} = \overline{t}$ holds for all t. The soundness of $\mathcal{V}$ implies the existence of a derivation $t \xrightarrow{*}_{\mathfrak{R}} \overline{t}$ and, thus, the transformation $\mathcal{J}$ is sound. □

4. TRANSFORMATION OF FUNCTIONAL LOGIC PROGRAMS

As stated in Section 1, our main aim is to apply parallel narrowing to conditional TRSs. The remaining task to reach this aim is to examine under which circumstances the property of weak orthogonality of the target system can be guaranteed, since this is a basic requirement for the completeness of parallel narrowing. Therefore, we will extend the notion of (weak) orthogonality to CTRSs. The basic idea of orthogonality is as follows. If t is an arbitrary term that can be reduced to two terms t_1, t_2, then these reductions do not overlap. In the case of weak orthogonality, any two overlapping rules must reduce the terms in the same way, i.e., $t_1 = t_2$.

For constructor-based TRSs, overlaps can only occur between rules de£ning the same operation and at root positions. Thus, it is suf£cient to examine rules $l_i \rightarrow r_i, i = 1, 2$ where l_1 and l_2 are uni£able.

[10] p and p' differ slightly because terms of $\mathcal{R}_\mathcal{V}$ contain variable vectors and terms of $\mathcal{R}_\mathcal{J}$ do not. Thus, whenever a pre£x pp of p denotes a conditional position with $s|_{pp} = u$, the corresponding pp' has to denote the $\mu(u)$ in $\mu(s)|_{qq'} = [\mu(u), (\vec{x})]$ (where qq' is directly above pp').

[11] In the case of Example 6, the existence of a rule $f(g(s(0)) \mid \perp) \rightarrow f(g(s(0)) \mid s(0))$ in a constructor-based system $\mathcal{R}$ implies that there cannot be a rule in $\mathcal{R}$ rooted by the symbol g.

Considering constructor-based CTRSs, the left-hand sides of two different rules might unify but, nevertheless, there may not exist any term that is contracted by both rules. This can happen because the conditions of the two rules may not be satis£able by a single term. Thus, we de£ne (weak) orthogonality for conditional term rewriting systems as follows.

DEFINITION 14. **((Weakly) Orthogonal CTRS)** *Let $\mathfrak{R}$ be a left-linear and constructor-based CTRS in which all unconditional rules $l \rightarrow r$ are considered to be of the form $l \rightarrow r$ if $true$.* $\mathfrak{R}$ *is* orthogonal *if for all distinct conditional rules $l_i \rightarrow r_i$ if $u_i \xrightarrow{*} v_i, i \in \{1, 2\}$, chosen with no variables in common, and for all uni£ers σ with $\sigma(l_1) = \sigma(l_2)$ at most one derivation among $\sigma(u_1) \xrightarrow{*} v_1$ and $\sigma(u_2) \xrightarrow{*} v_2$ is possible.*
$\mathfrak{R}$ *is* weakly orthogonal *if for all distinct rules $l_i \rightarrow r_i$ if $u_i \xrightarrow{*} v_i, i \in \{1, 2\}$, chosen with no variables in common, and for all uni£ers σ with $\sigma(l_1) = \sigma(l_2)$ it holds: either $\sigma(r_1) = \sigma(r_2)$ or at most one derivation among $\sigma(u_1) \xrightarrow{*} v_1$ and $\sigma(u_2) \xrightarrow{*} v_2$ is possible.* ◇

This de£nition captures the idea of orthogonality, as described above. It generalizes a similar notion in [8] in which multiple rules with overlapping left-hand sides are not allowed. Various examples in this paper show that this generalization supports a more expressive programming style.

Note that there is an important difference between orthogonality for TRSs and the extended notion for CTRSs. Orthogonality for a TRS is effectively decidable by a simple analysis of the syntactic structure of the system. For CTRSs orthogonality is not decidable, since the de£nition considers a semantic property by testing the simultaneous satis£ability of the conditions. If in practical applications the property of (weak) orthogonality has to be effectively tested, e.g., by the compiler of a functional logic program, orthogonality can be syntactically approximated, see, for instance, [30].

Since orthogonality for CTRSs is a non-trivial semantic property and orthogonality for TRSs is a syntactic property, no computation can transform a (weakly) orthogonal CTRS into a (weakly) orthogonal TRS. The key idea is to limit the notion of orthogonality to that part of the target system $\mathcal{R}_\mathcal{J}$ which represents the semantics of $\mathfrak{R}$: the terms of interest $\mathcal{T}_{\mathcal{R}_\mathcal{J}}$. It turns out that $\mathcal{R}_\mathcal{J}$ is (weakly) orthogonal if $\mathfrak{R}$ is (weakly) orthogonal provided that we consider only the terms of interest $\mathcal{T}_{\mathcal{R}_\mathcal{J}}$. To show this, we £rst de£ne what orthogonality means when restricted to a set of terms:

DEFINITION 15. **((Weak) Orthogonality for $\mathcal{T}$)** *Let $\mathcal{R}$ be a program and $\mathcal{T}$ a set of terms which is closed under $\xrightarrow{*}_\mathcal{R}$.* $\mathcal{R}$ *is* orthogonal for $\mathcal{T}$ *if, for all distinct rules ρ_1 and ρ_2 in $\mathcal{R}$, there exist no terms $t, t_1, t_2 \in \mathcal{T}$ and a position p in t such that $t \rightarrow_{p,\rho_1} t_1$ and $t \rightarrow_{p,\rho_2} t_2$.*
$\mathcal{R}$ *is* weakly orthogonal for $\mathcal{T}$ *if, for all rules ρ_1, ρ_2 in $\mathcal{R}$, there exist no terms $t, t_1, t_2 \in \mathcal{T}$ and a position p in t such that $t \rightarrow_{p,\rho_1} t_1$ and $t \rightarrow_{p,\rho_2} t_2$ and $t_1 \neq t_2$.* ◇

Analogously to the de£nition of ordinary orthogonality (De£nition 10), we make use of the constructor discipline of the supposed system, since we consider overlapping at a same position only. Note that, if $\mathcal{T} = \mathcal{T}(\Sigma, \mathcal{X})$, then (weak) orthogonality for $\mathcal{T}$ is equivalent to ordinary (weak) orthogonality.

We can now de£ne the set of CTRSs on which our transformation $\mathcal{J}$ preserves orthogonality for the terms of interest:

DEFINITION 16. **(Conditional Program)** *We call a* conditional program *any constructor-based weakly orthogonal CTRS in which overlapping rules have identical left-hand sides.* ◇

Limiting conditional programs to overlapping rules with identical left-hand sides is not a restriction. We show by an example the idea of transforming every constructor-based left-linear CTRS into a form where all overlapping rules have identical left-hand sides. This is done by moving the pattern matching into the conditions and adding explicit rules for these matchings for the needed constructors. This example can be easily generalized to all constructor-based weakly orthogonal CTRSs. A formal proof can be found in [11].

EXAMPLE 7. *The system de£ned by the rules*

$$\begin{array}{lll} \mathit{fib}(x) & \to 1 & \mathit{if}\ x < 2 \xrightarrow{*} \mathit{true} \\ \mathit{fib}(s(s(x))) & \to \mathit{fib}(s(x)) + \mathit{fib}(x) & \end{array}$$

can also be written as

$$\begin{array}{lll} \mathit{fib}(x) & \to 1 & \mathit{if}\ x < 2 \xrightarrow{*} \mathit{true} \\ \mathit{fib}(x) & \to \mathit{fib}(s(\mathit{select}(x))) + \mathit{fib}(\mathit{select}(x)) & \\ & & \mathit{if}\ \mathit{match}(x) \xrightarrow{*} \mathit{true} \\ \mathit{match}(s(s(x))) & \to \mathit{true} & \\ \mathit{select}(s(s(x))) & \to x & \end{array}$$

Operations select and match are speci£cally generated wherever needed. A compiler implementing this transformation would probably bene£t from more general purpose operations, e.g., a select_n operation that selects the n-th argument of a constructor-rooted term. However, our specialized operations ease the understanding of the example. Note that it may be necessary to rename variables in other rules in order to get identical left-hand sides. Derivations in the new system will be longer because of the explicit matching and the selection steps, but it is easy to see that there exists a bisimulation between the two systems with respect to starting terms of the universe of the original system. The advantage of this transformation over similar approaches, like the "sequentialization" given in [2, section 4.3], is that it preserves (weak) orthogonality, which is crucial for our purpose.

From now on we will assume that overlapping rules in the given CTRSs have identical left-hand sides, as the only considered systems are constructor-based and left linear.

THEOREM 4. *Let $\mathfrak{R}$ be a conditional program. Then $\mathcal{R}_\mathcal{J}$ is weakly orthogonal for the set of terms of interest $\mathcal{T}_{\mathcal{R}_\mathcal{J}}$. Moreover, if $\mathfrak{R}$ is orthogonal, then $\mathcal{R}_\mathcal{J}$ is orthogonal for the set of terms of interest $\mathcal{T}_{\mathcal{R}_\mathcal{J}}$.*

PROOF. Suppose that there exists a t in $\mathcal{T}_{\mathcal{R}_\mathcal{J}}$ and two distinct rules $\rho : l \to r$ and $\rho' : l' \to r'$ such that

$$t \to_{\Lambda,\rho} w \quad \text{and} \quad t \to_{\Lambda,\rho'} w'$$

Without loss of generality, we consider only root positions, since in the constructor-based setting one can always remove the context of a redex. Let π and π' be the rules of the CTRS $\mathfrak{R}$, from which ρ and ρ' originate. The transformation replaces each conditional rule by two unconditional ones. The instantiating rule introduces the conditions by replacing $\bot$ symbols. The reducing rule contracts the redex analogously to the unconditional part of the original rule. We do not have overlapping instantiating rules because they are joined by the transformation. Thus, all overlapping rules and, in particular, ρ and ρ' must be reducing rules. Hence, because all overlapping rules have identical left-hand-sides, the two original rules of $\mathfrak{R}$ have the form $\pi : \overline{l} \to \overline{r}\ \mathit{if}\ v \xrightarrow{*} u$ and $\pi' : \overline{l} \to \overline{r'}\ \mathit{if}\ v' \xrightarrow{*} u'$. Furthermore, the pattern l must match both terms $\overline{t}$ and $\overline{t'}$. Since the transformation is sound, this implies the existence of the two reductions $\overline{t} \to_{\Lambda,\pi} \overline{w}$ and $\overline{t} \to_{\Lambda,\pi'} \overline{w'}$ and, therefore, $\mathfrak{R}$ cannot have been orthogonal.
For weak orthogonality, we have to consider the additional case that w and w' are different. Since the right-hand sides of the reducing rules ρ and ρ' can only be different if the right-hand sides of their original rules, namely $\overline{r}$ and $\overline{r'}$, differ, $\overline{w} \neq \overline{w'}$ implies that $\mathfrak{R}$ cannot have been weakly orthogonal as well. □

In the next section we will show that for a conditional program $\mathfrak{R}$ the resulting system $\mathcal{R}_\mathcal{J}$ can be evaluated with parallel narrowing if we limit ourselves to the terms of interest. For this it is crucial to see that the Parallel Moves Lemma (originally in [23], here according to the more general formulation [7, Lemma 6.4.4]) can be extended to the rewrite relation on $\mathcal{T}_{\mathcal{R}_\mathcal{J}}$. For this the notion of a *parallel rewriting step* is required: $t \rightrightarrows_P t'$ if $P = \{p_1, \ldots, p_n\}$ is a set of pairwise disjoint positions of t such that and $t \to_{p_1} \cdots \to_{p_n} t'$.

LEMMA 2. **Parallel Moves Lemma [7, Lemma 6.4.4]** *Let $\mathcal{R}$ be a TRS and $l \to r \in \mathcal{R}$ a left-linear rule. If $\sigma(l) \rightrightarrows_P t$ for a set of positions P and all elements of P are below variable positions of l, then there exists a substitution σ' such that $\sigma(r) \rightrightarrows \sigma'(r) \leftarrow t$.*

The important point is that on the terms of interests $\mathcal{T}_{\mathcal{R}_\mathcal{J}}$ any parallel rewriting step meets the conditions of the Parallel Moves Lemma:

LEMMA 3. *Let $\mathfrak{R}$ be a conditional program and $l \to r$ a rule in $\mathcal{R}_\mathcal{J}$. If $\sigma(l) \in \mathcal{T}_{\mathcal{R}_\mathcal{J}} \rightrightarrows_P t$ for a set of (disjoint) positions P, then there exists a substitution σ' such that $\sigma(r) \rightrightarrows \sigma'(r) \leftarrow t$ or $\sigma(r) = \sigma'(r) = t$.*

PROOF. By de£nition of a conditional program and construction of $\mathcal{J}$, all rules of $\mathcal{R}_\mathcal{J}$ are left linear. As conditional programs are also constructor-based, all positions in P are either below variable positions of l or $P = \{\Lambda\}$. In the £rst case, the Parallel Moves Lemma yields the result. In the second case, because $\mathcal{R}_\mathcal{J}$ is weakly orthogonal for $\mathcal{T}_{\mathcal{R}_\mathcal{J}}$, $\sigma(r)$ must be equal to t no matter which rule was applied. □

Analogously to the discussion in [7, p.152f], this leads to the proof that $\rightrightarrows_{\mathcal{R}_\mathcal{J}}$ has the diamond property if it is restricted to the terms of interest:

LEMMA 4. *Let $\mathfrak{R}$ be a conditional program. Then for the terms of interest, $\rightrightarrows$ has the diamond property, i.e., for all $s \in \mathcal{T}_{\mathcal{R}_\mathcal{J}}$ with $s \rightrightarrows_{P_0} t_0$ and $s \rightrightarrows_{P_1} t_1$ there exists an s' with $t_0 \rightrightarrows s'$ and $t_1 \rightrightarrows s'$.*

PROOF. (Analogously to the proof in [7, p.152]) We have to distinguish how the positions of the parallel reductions are relative to each other and partition the sets P_0 and P_1 accordingly:

$$\begin{array}{l} P^{\not\leq} := \{p \in P_i \mid i \in \{0,1\} \wedge \forall q \in P_{1-i} : q \not\leq p\} \\ P^{<} := \{p \in P_i \mid i \in \{0,1\} \wedge \exists q \in P_{1-i} : q < p\} \\ P^{=} := P_0 \cap P_1 \end{array}$$

It is easy to see that we can obtain t_0 and t_1 from s by replacing all subterms of s at positions in $P^{\not\trianglelefteq} \cup P^{=}$ with the corresponding subterms of t_i: $t_i = s[t_i|_p]_{p \in P^{\not\trianglelefteq} \cup P^{=}}$. As $\mathcal{R}_{\mathcal{J}}$ is weakly orthogonal for $\mathcal{T}_{\mathcal{R}_{\mathcal{J}}}$, $t_0|_p = t_1|_p$ holds for all $p \in P^{=}$. Hence it suffices to construct, for each $p \in P^{\not\trianglelefteq}$, a term u_p such that $t_i|_p \rightrightarrows u_p$ $(i = 1, 2)$. For each $p \in P^{\not\trianglelefteq}$, either t_0 or t_1 is rewritten at p. Calling this subterm t_p, there must exist a rule $l \to r$ in $\mathcal{R}_{\mathcal{J}}$ and a substitution σ such that $s|_p = \sigma(l)$ and $t_p = \sigma(r)$. As $s|_p \in \mathcal{T}_{\mathcal{R}_{\mathcal{J}}}$, our adapted Parallel Moves Lemma (Lemma 3) ensures the existence of a substitution σ' with $t_i|_p \rightrightarrows \sigma'(r)$ or $t_i|_p = \sigma'(r)$ and, thus, we define $u_p := \sigma'(r)$. □

These considerations enable us to draw some interesting conclusions. We use [7, Corollary 2.7.7] which states: If $\to_1 \subseteq \to_2 \subseteq \overset{*}{\to}_1$ and $\to_2$ has the diamond property, then $\to_1$ is confluent. With $\to_1 = \to_{\mathcal{R}_{\mathcal{J}}}$ and $\to_2 = \rightrightarrows_{\mathcal{R}_{\mathcal{J}}}$, both restricted to the terms of interest, we obtain:

COROLLARY 1. *For any conditional program $\mathfrak{R}$ and the corresponding TRS $\mathcal{R}_{\mathcal{J}}$, the following properties hold:*

- *$\mathcal{R}_{\mathcal{J}}$ is confluent for $\mathcal{T}_{\mathcal{R}_{\mathcal{J}}}$.*
- *Each $\bot$-term has at most one normal form.*
- *$\mathfrak{R}$ is confluent (by soundness of the transformation).*

Note how the soundness of the transformation enables us to draw conclusions backwards, from the target to the source system. By using this method, results about classic TRSs can be extended to the conditional case. This result can be generalized to constructor-based weakly orthogonal CTRSs using the transformation that ensures that the left-hand sides of overlapping rules are identical, see Example 7.

5. NARROWING ON TRANSFORMED FUNCTIONAL LOGIC PROGRAMS

This paper is concerned with the use of CTRSs in declarative programming. Functional logic programs modeled by TRSs are usually executed by narrowing because narrowing amalgamates computational ideas from both functional and logic programming. Consequently, a pertinent question is the choice of a narrowing strategy appropriate to a given class of programs. The primary motivation of this work is the transformation of a CTRS into an ordinary TRS so that a wealth of existing results about narrowing strategies for ordinary TRSs can be applied to CTRSs.

In general, if we only consider the syntactic structure of the rules of $\mathcal{R}_{\mathcal{J}}$, we will not find any of the properties that support efficient, sound, and complete strategies, e.g., inductive sequentiality or weak orthogonality. However, we have shown that some strong properties are preserved for the rewrite relation over the set of the terms of interest $\mathcal{T}_{\mathcal{R}_{\mathcal{J}}}$. These terms capture all and only the computations that one can perform with $\mathfrak{R}$. Hence, there is no loss in restricting our attention to these terms and considering terms outside this set does not conform with the semantics of $\mathfrak{R}$. Moreover, by construction, the set of terms of interest of $\mathcal{R}_{\mathcal{J}}$ is closed under both *substitution of bottom terms* and *rewriting*. Consequently, it is closed under narrowing for bottom terms, too.

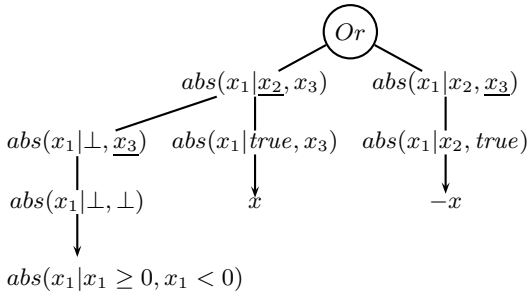
Weakly needed and parallel narrowing [4] are strategies defined for those TRSs $\mathcal{R}$ in which any operation defined in $\mathcal{R}$ has a parallel definitional tree [1]. For a defined operation f, the *definitional tree for f* is a data structure defined by the patterns, i.e., the left-hand sides, of the rules defining f in $\mathcal{R}$. A *parallel* definitional tree may have so-called *or*-nodes in addition to those defined by pattern matching. These *or*-nodes allow the specification of overlapping rules and, thus, lead to parallel tests for pattern matching. This is necessary to cover the class of weakly orthogonal TRSs. A consequence of allowing overlapping rules is that the notion of a *needed redex position*, i.e., the position of a redex in a given term t which has to be reduced in order to yield a normal form for t, has to be generalized to a *necessary set* of redex positions. From this set the redex of *at least one* position has to be contracted to yield a normal form. This leads to the basic idea of *weakly needed narrowing*: (a) compute such a necessary set of positions for a given term using the parallel definitional trees for the defined operations of $\mathcal{R}$; (b) non-deterministically choose some of the positions; (c) instantiate the terms at the chosen positions so that they become redexes and reduce them. In comparison to weakly needed narrowing, the strategy called *parallel narrowing* is more sophisticated: a *minimal* set of substitutions, i.e., the instantiations of variables, is computed by examining *the whole necessary set* of positions. For each substitution in this minimal set all resulting weakly needed redexes are reduced in parallel. Consequently, parallel narrowing is, in contrast to weakly needed narrowing, deterministic on ground terms since there is only the empty substitution in this case. Due to lack of space, we can not present all precise definitions but they are available in [4].

In order to apply weakly or parallel narrowing, we have to prove that any operation defined in $\mathcal{R}_{\mathcal{J}}$ has a parallel definitional tree:

LEMMA 5. *Let $\mathcal{R}_{\mathcal{J}}$ be a TRS obtained by transformation $\mathcal{J}$ from a conditional program $\mathfrak{R}$. If f is a defined operation of $\mathcal{R}_{\mathcal{J}}$, there exists a parallel definitional tree for f.*

PROOF. If there is no left-hand side of a rule subsumed by the left-hand side of another rule in $\mathcal{R}_{\mathcal{J}}$, [1, Theorem 19] proves the existence of a parallel definitional tree. Note that [1, Theorem 19] is proved under the assumption that the TRS is *weakly* orthogonal, but the proof considers only the left-hand sides of the rules. Thus, the triviality of possible critical pairs is irrelevant. Now we prove that if the left-hand side of a rule is subsumed by some other left-hand side in $\mathcal{R}_{\mathcal{J}}$, then one of the rules is *useless* (see [1]) and can be discarded. By construction, subsuming left-hand sides of $\mathcal{R}_{\mathcal{J}}$ may originate only from subsuming left-hand sides of $\mathfrak{R}$. In the original program $\mathfrak{R}$, the only case of subsumption of left-hand sides occurs when there are several conditional rules and possibly one unconditional rule, all with the same left-hand side. If there is one such unconditional rule, we can discard all the conditional rules with the same left-hand side. So, we are left only with conditional rules. In this case, the rules of $\mathcal{R}_{\mathcal{J}}$ that originate from these conditional rules do not have subsuming left-hand sides according to the definition of the transformation $\mathcal{J}$. □

EXAMPLE 8. *The function abs of Example 5 has the following parallel definitional tree:*



Hence, one can apply both the weakly needed and parallel narrowing strategies to $\mathcal{R}_{\mathcal{J}}$. However, the narrowing relation computed

by these strategies might not be complete for $\mathcal{R}_\mathcal{J}$ w.r.t. arbitrary terms, because these strategies £re only one rule (after variable instantiation) when multiple overlapping rules can be applied to a same redex. This problem does not occur for the set of the terms of interest $\mathcal{T}_{\mathcal{R}_\mathcal{J}}$ obtained from a conditional program $\mathfrak{R}$, because either a single rule can be £red for a redex (when $\mathfrak{R}$ is orthogonal and, hence, $\mathcal{R}_\mathcal{J}$ is orthogonal for $\mathcal{T}_{\mathcal{R}_\mathcal{J}}$) or the choice of multiple rules does not matter (when $\mathfrak{R}$ is weakly orthogonal and, hence, $\mathcal{R}_\mathcal{J}$ is weakly orthogonal for $\mathcal{T}_{\mathcal{R}_\mathcal{J}}$). This leads us to our main result:[12]

THEOREM 5. *Let $\mathfrak{R}$ be a conditional program. Then both weakly needed narrowing and parallel narrowing are sound and complete with respect to the terms of interest $\mathcal{T}_{\mathcal{R}_\mathcal{J}}$, i.e.:*

1. *For each $t \in \mathcal{T}_{\mathcal{R}_\mathcal{J}}$ and every constructor term $s \in \mathcal{T}(\mathcal{C}, X)$ holds: If $t \leadsto_{\sigma_1} \ldots \leadsto_{\sigma_n} s$ ($t \overset{\sim}{\sim}_{\sigma_1} \ldots \overset{\sim}{\sim}_{\sigma_n} s$) is a weakly needed (parallel) narrowing derivation, then $\sigma_n(\ldots\sigma_1(t)\ldots) \overset{*}{\to} s$ (soundness).*
2. *For each term $t \in \mathcal{T}_{\mathcal{R}_\mathcal{J}}$ and each constructor substitution σ such that $\sigma(t)$ is also in $\mathcal{T}_{\mathcal{R}_\mathcal{J}}$ and reduces to a constructor term $s \in \mathcal{T}(\mathcal{C}, X)$, there exists a weakly needed (parallel) narrowing derivation $t \leadsto_{\sigma_1} \ldots \leadsto_{\sigma_n} s'$ ($t \overset{\sim}{\sim}_{\sigma_1} \ldots \overset{\sim}{\sim}_{\sigma_n} s'$) such that $\sigma_n \circ \cdots \circ \sigma_1$ is more general than σ and s is an instance of s' (completeness).*

PROOF. Due to Lemma 5, both strategies can be applied to $\mathcal{R}_\mathcal{J}$.
(Soundness)
The soundness of both strategies is straightforward because a narrowing step is de£ned as an instantiation followed by a rewriting step.
(Completeness)
In [3] the completeness of weakly needed and parallel narrowing are proven using commutative diagrams. The existence and commutativity of these diagrams is ensured by con¤uence and the Parallel Moves Lemma. Consequently the proofs of completeness for weakly needed narrowing [3, Theorem 4] and parallel needed narrowing [3, Theorems 5] carry over without dif£culty. □

Weakly needed and parallel narrowing are relatively ef£cient for the class of systems we are discussing. The main advantage of using parallel narrowing for the class of transformed conditional programs is that, on ground terms, the computation is deterministic, i.e., at each computation step there are no *don't know choices*. This ensures, among other advantages, that non-terminating evaluations of a condition do not preclude the application of other conditional rules. To see this, refer to the CTRS from Example 2 in Section 1. In that system the evaluation of the expression $gmult(\infty, 0)$ does not terminate if we try to apply the £rst rule for $gmult$. Similarly, $gmult(0, \infty)$ does not terminate for the second rule and the third rule leads to an in£nite computation for both terms. However, consider the system obtained by our transformation.

EXAMPLE 9. *Transforming the CTRS from Example 2, we obtain:*

$$
\begin{array}{l}
gmult(x, y \mid \bot, \bot, \bot) \to \\
\qquad gmult(x, y \mid g(x), g(y), g(x) > 0 \wedge g(y) > 0) \\
gmult(x, y \mid 0, y_2, y_3) \to 0 \\
gmult(x, y \mid y_1, 0, y_3) \to 0 \\
gmult(x, y \mid y_1, y_2, true) \to g(x) * g(y) \\
g(0) \to 0 \\
g(\infty) \to g(\infty)
\end{array}
$$

[12]As usual in functional logic programming, we consider only constructor substitutions and constructor terms as the intended results of a computation, cf. [5]

In this system both expressions $gmult(\infty, 0 \mid \bot, \bot, \bot)$ and $gmult(0, \infty \mid \bot, \bot, \bot)$ are evaluated with a £nite number of parallel narrowing steps:

$$
\begin{array}{l}
\underline{gmult(\infty, 0 | \bot, \bot, \bot)} \\
\overset{\sim}{\sim} \underline{gmult(\infty, 0 | g(\infty), g(0), g(\infty) > 0 \wedge g(0) > 0)} \\
\overset{\sim}{\sim} \underline{gmult(\infty, 0 | g(\infty), 0, g(\infty) > 0 \wedge 0 > 0)} \\
\overset{\sim}{\sim} 0
\end{array}
$$

and

$$
\begin{array}{l}
\underline{gmult(0, \infty | \bot, \bot, \bot)} \\
\overset{\sim}{\sim} \underline{gmult(0, \infty | g(0), g(\infty), g(0) > 0 \wedge g(\infty) > 0)} \\
\overset{\sim}{\sim} \underline{gmult(0, \infty | 0, g(\infty), 0 > 0 \wedge g(\infty) > 0)} \\
\overset{\sim}{\sim} 0
\end{array}
$$

6. CONCLUSIONS AND RELATED WORK

We have presented a new evaluation strategy for functional logic programs where functions are de£ned by conditional rewrite rules. In order to cover a large part of such programs, we consider weakly orthogonal constructor-based CTRSs, i.e., the overlaps are either trivial or the conditions of overlapping rules are exclusive. We have presented a transformation for this class of programs that maps such programs into unconditional TRSs. Although the resulting TRSs are not weakly orthogonal in general, they have this property for all terms of interest, i.e., terms obtained by this transformation. This property enables us to apply appropriate narrowing strategies for unconditional TRSs to evaluate such programs. In particular, both the non-strict evaluation strategies weakly needed and parallel narrowing can be applied to the transformed programs. Since parallel narrowing is deterministic on ground terms, our work offers the £rst known narrowing strategy that deterministically evaluates ground terms in weakly orthogonal constructor-based CTRSs.

We want to emphasize that our new strategy is a conservative extension of narrowing strategies known to be appropriate for functional logic programs. If the source program contains no conditional rules, our transformation is the identity. Thus, if the source program is inductively sequential, our strategy specializes to needed narrowing that is known to be optimal w.r.t. the length of successful computations and the number of computed solutions. Furthermore, if the left-hand sides of a conditional program $\mathfrak{R}$ are inductively sequential, then weakly needed narrowing as well as parallel narrowing computes a set of disjoint solutions [4, 5].

Our transformation is not an "unraveling" in the sense of [29]. By de£nition, an unraveling leaves the unconditional rules of a CTRS unchanged [29, De£nition 3.1]. Furthermore, a "tidy" unraveling will transform every single conditional rule separately [29, p.109]. This leads to the idea to introduce a new function symbol for each conditional rule. In this case, one has to decide which rule must be applied before knowing which conditions are satis£able. The key idea of Viry's approach is to keep the information of testing all conditions in a single structure. This ensures that the correct rule can be chosen depending on the successful tests of all conditions. This has the immediate consequence that Viry's transformation can be extended to preserve properties like termination and con¤uence (as shown in [11]) which unravelings principally cannot [29, p.120].

Transforming each conditional rule separately is also the approach of [33] so that it leads to the same problem as Marchiori's unravelings. The interesting feature of the transformation presented in [33] is that it can deal with a certain form of extra variables. The extension of our transformation to the systems considered by Ohlebusch could be done by applying lambda-lifting [24] to eliminate this kind of variables.

An interesting topic for future work is the ef£cient implementation of this strategy. A useful starting point could be [6] where an implementation of weakly needed narrowing with a limited form of parallel reductions is proposed.

7. REFERENCES

[1] S. Antoy. De£nitional trees. In *Proc. of the 3rd International Conference on Algebraic and Logic Programming*, pages 143–157. Springer LNCS 632, 1992.

[2] S. Antoy. Constructor-based conditional narrowing. In *Proc. of the 3rd International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP 2001)*, pages 199–206. ACM Press, 2001.

[3] S. Antoy, R. Echahed, and M. Hanus. A parallel narrowing strategy. Technical report tr 96-1, Portland State University, 1996.

[4] S. Antoy, R. Echahed, and M. Hanus. Parallel evaluation strategies for functional logic languages. In *Proc. of the Fourteenth International Conference on Logic Programming (ICLP'97)*, pages 138–152. MIT Press, 1997.

[5] S. Antoy, R. Echahed, and M. Hanus. A needed narrowing strategy. *Journal of the ACM*, 47(4):776–822, 2000.

[6] S. Antoy, M. Hanus, B. Massey, and F. Steiner. An implementation of narrowing strategies. In *Proc. of the 3rd International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP 2001)*, pages 207–217. ACM Press, 2001.

[7] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[8] J.A. Bergstra and J.W. Klop. Conditional rewrite rules: Con¤uence and termination. *Journal of Computer and System Sciences*, 32(3):323–362, 1986.

[9] J.A. Bergstra and J.V. Tucker. Algebraic speci£cations of computable and semicomputable data types. *Theoretical Computer Science*, 75:111–138, 1987.

[10] D. Bert and R. Echahed. Design and implementation of a generic, logic and functional programming language. In *Proc. European Symposium on Programming*, pages 119–132. Springer LNCS 213, 1986.

[11] B. Brassel. Conditional narrowing with lazy evaluation (in german). Master's thesis, RWTH Aachen, 1999.

[12] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 243–320. Elsevier, 1990.

[13] N. Dershowitz and M. Okada. A rationale for conditional equational programming. *Theoretical Computer Science*, 75:111–138, 1990.

[14] 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.

[15] E. Giovannetti, G. Levi, C. Moiso, and C. Palamidessi. Kernel LEAF: A logic plus functional language. *Journal of Computer and System Sciences*, 42(2):139–185, 1991.

[16] E. Giovannetti and C. Moiso. Notes on the elimination of conditions. In *Proc. CTRS'87*, pages 91–97. Springer LNCS 308, 1987.

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

[19] M. Hanus. Reduction strategies for declarative programming. In B. Gramlich and S. Lucas, editors, *Electronic Notes in Theoretical Computer Science*, volume 57. Elsevier Science Publishers, 2001.

[20] M. Hanus, S. Lucas, and A. Middeldorp. Strongly sequential and inductively sequential term rewriting systems. *Information Processing Letters*, 67(1):1–8, 1998.

[21] M. Hanus (ed.). Curry: An integrated functional logic language (vers. 0.8). Available at `http://www.informatik.uni-kiel.de/~curry`, 2003.

[22] C. Hintermeier. How to transform canonical decreasing CTRSs into equivalent canonical TRSs. In *Proc. of the 4th International Workshop on Conditional and Typed Rewriting Systems (CTRS'94)*, pages 186–205. Springer LNCS 968, 1994.

[23] G. Huet and J.-J. Lévy. Computations in orthogonal rewriting systems. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 395–443. MIT Press, 1991.

[24] T. Johnsson. Lambda lifting: Transforming programs to recursive functions. In *Functional Programming Languages and Computer Architecture*, pages 190–203. Springer LNCS 201, 1985.

[25] J.W. Klop. Term rewriting systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume II. Oxford University Press, 1992.

[26] J. Lloyd. Programming in an integrated functional and logic language. *Journal of Functional and Logic Programming*, (3):1–49, 1999.

[27] 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.

[28] F. López-Fraguas and J. Sánchez-Hernández. TOY: A Multiparadigm Declarative System. In *Proc. of RTA'99*, pages 244–247. Springer LNCS 1631, 1999.

[29] M. Marchiori. Unravelings and ultra-properties. In *5th International Conference on Algebraic and Logic Programming (ALP'96)*, pages 107–121. Springer LNCS 1139, 1996.

[30] 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.

[31] 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.

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

[33] E. Ohlebusch. Transforming conditional rewrite system with extra variables into unconditional systems. In *Proc. of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, pages 111–130. Springer LNCS 1705, 1999.

[34] G. Smolka. The Oz programming model. In J. van Leeuwen,

editor, *Computer Science Today: Recent Trends and Developments*, pages 324–343. Springer LNCS 1000, 1995.

[35] Z. Somogyi, F. Henderson, and T. Conway. The execution algorithm of Mercury, an ef£cient purely declarative logic programming language. *Journal of Logic Programming*, 29(1-3):17–64, 1996.

[36] P. Viry. Elimination of conditions. *Journal of Symbolic Computation*, 28:381–400, 1999.