Safe Folding/Unfolding with Conditional Narrowing*

M. Alpuente[1] and M. Falaschi[2] and G. Moreno[3] and G. Vidal[1]

[1] DSIC, Universidad Politécnica de Valencia, Camino de Vera s/n, Apdo. 22012, 46020 Valencia, Spain. e.mail:{alpuente,gvidal}@dsic.upv.es.
[2] Dipartimento di Matematica e Informatica, Università di Udine, Via delle Scienze 206, 33100 Udine, Italy. e.mail:falaschi@dimi.uniud.it.
[3] Departamento de Informática, Universidad de Castilla-La Mancha, Campus Universitario s/n, 02071 Albacete, Spain. e.mail:Gmoreno@info-ab.uclm.es.

Abstract. Functional logic languages with a complete operational semantics are based on narrowing, a generalization of term rewriting where unification replaces matching. In this paper, we study the semantic properties of a general transformation technique called *unfolding* in the context of functional logic languages. Unfolding a program is defined as the application of narrowing steps to the calls in the program rules in some appropriate form. We show that, unlike the case of pure logic or pure functional programs, where unfolding is correct w.r.t. practically all available semantics, unrestricted unfolding using narrowing does not preserve program meaning, even when we consider the weakest notion of semantics the program can be given. We single out the conditions which guarantee that an equivalent program w.r.t. the semantics of computed answers is produced. Then, we study the combination of this technique with a *folding* transformation rule in the case of innermost conditional narrowing, and prove that the resulting transformation still preserves the computed answer semantics of the initial program, under the usual conditions for the completeness of innermost conditional narrowing. We also discuss a relationship between unfold/fold transformations and partial evaluation of functional logic programs.

1 Introduction

The problem of integration of functional and logic programming is an important challenge for research in declarative programming (see [15] for a recent survey). A functional logic program can be seen as a Conditional Term Rewriting System (CTRS for short), i.e. a set of conditional equations where the equation in the conclusion is implicitly oriented from left to right. Functional logic languages obtain the power of logic variables, automatic search and constraint solving from logic programming. From functional programming, they obtain the expressivity of functions and types, and a more efficient evaluation mechanism thanks to

* This work has been partially supported by CICYT under grant TIC 95-0433-C03-03 and by HCM project CONSOLE.

the deterministic reduction of functional expressions [14, 15]. The operational semantics is usually based on some variant of narrowing, an execution mechanism which consists of the instantiation of goal variables followed by a reduction step on the instantiated goal. The standard declarative semantics of a program $\mathcal{E}$ is given by the least Herbrand $\mathcal{E}$-model of the program, i.e. the set of all ground equations which hold in the underlying theory [17].

The folding and unfolding transformations, which were first introduced by Burstall and Darlington in [7] for functional programs, are the most basic and powerful techniques for a framework to transform programs. Unfolding is essentially the replacement of a call by its body, with appropriate substitutions. Folding is the inverse transformation, the replacement of some piece of code by an equivalent function call. For functional programs, folding and unfolding steps involve only pattern matching. The fold/unfold transformation approach was first adapted to logic programs by Tamaki and Sato [28] by replacing matching with unification in the transformation rules. A lot of literature has been devoted to proving the correctness of unfold/fold systems w.r.t. the various semantics proposed for functional programs [7, 20], logic programs [18, 25, 27, 28], and constraint logic programs [12]. However, to the best of our knowledge, these techniques have not been studied for functional logic programs so far.

The purpose of this paper is to consider unfold/fold transformations which preserve the semantics of computed answer substitutions of functional logic programs. This type of program behavior is naturally observed by the programmers. However, for the sake of simplicity, most often logic program transformation techniques are only proved correct w.r.t. the declarative semantics of ground logical consequences. We first show what are the problems with naïve extensions of these transformation rules to functional logic programs, considering unrestricted narrowing as the language operational semantics.

Then we show a non standard and extremely useful relationship of partial evaluation with unfolding. We show that a slightly modified transformation (*generalized unfolding*) can be formulated in terms of partial evaluation. As a consequence, the conditions to ensure completeness of the partial evaluation process (defined in [4]) can be used to formalize a sufficient condition for the completeness of unfolding w.r.t. the computed answers for unrestricted narrowing. Note that this is different from the case of pure logic programming, where no applicability condition is required to produce an equivalent program.

The definition of a folding transformation for unrestricted narrowing requires conditions which are too strong to preserve computed answers. For this reason and in order to study the typical properties of a more efficient narrowing strategy, we have defined a folding rule directly for innermost narrowing and have instantiated the general unfolding definition to this case. We have then proved that the unfolding/folding transformation preserve the computed answers under the usual conditions for the completeness of innermost conditional narrowing. In our formulation, unfolding allows the expansion of a single innermost call of a program rule at each step, and hence can be 'selectively' applied. This allows us to see the techniques of unfolding/folding as a base for the definition of a framework for the transformation of programs, in which heuristics or automatic transformation processes might be combined. Finally, as an example application of the unfolding technique we have defined a semantics modelling computed answers which consists of a (possibly infinite) set of unconditional rules, computed as the limit of the unfolding expansions of the initial program.

In the literature, we found only three explicit formulations of fold/unfold rules for functional logic programs, which are based on some form of narrowing. In [8], Darlington and Pull showed how instantiation (an operation of the Burstall/Darlington framework which introduces an instance of an existing equation) can be embedded into unfolding steps to get the ability (of narrowing) to deal with logical variables by means of unification. Similarly, folding steps are regarded as narrowing steps against the reversed equations. However, in this paper we show that folding steps require the ability to generalize (or "deinstantiate") calls rather than instantiating them, which is similar to the case of logic programming and unlike what is done by narrowing or SLD–resolution steps. No claim is made in [8] for any sort of completeness of the transformations and, indeed, some restrictions for the application of the rules are necessary to obtain it [11, 20]. Another closely related approach is that of [11], which formulates a rewrite-based technique for the synthesis of functional programs which makes use of the rule of instantiation. However, there, the manipulations induced to allow folding/unfolding are often more complex than simple instantiation and involve auxiliary function definition and induction. Finally, the forward closures of [9] produce a kind of unfolding of program rules which is used to formulate conditions for the termination of the program.

This paper is organized as follows. Section 2 formalizes the conditional narrowing semantics we focus on. In Section 3, we formalize the notion of unfolding for functional logic programs using conditional narrowing, give the conditions for the soundness and completeness of the transformation w.r.t. the set of ground equational consequences and show the relationship with partial evaluation. Then, we state and prove the soundness and completeness properties for a generalized unfolding technique w.r.t. computed answers. Section 4 introduces a transformation method which combines folding and unfolding for an efficient (call-by-value) evaluation strategy: innermost narrowing. As an application of the innermost unfolding transformation, in Section 5, we define a semantics based on unfolding which is able to characterize the answer substitutions computed by innermost narrowing syntactically. Section 6 concludes the paper and outlines some directions for future research. More details and missing proofs can be found in [1].

2 Semantics of Functional Logic Programs

An equational Horn theory $\mathcal{E}$ consists of a finite set of equational Horn clauses of the form $(\lambda = \rho) \Leftarrow C$, where the condition C is a (possibly empty) sequence $e_1, \ldots, e_n$, $n \geq 0$, of equations. A Conditional Term Rewriting System (CTRS for short) is a pair $(\Sigma, \mathcal{R})$, where $\mathcal{R}$ is a finite set of reduction (or rewrite) rule schemes of the form $(\lambda \to \rho \Leftarrow C)$, $\lambda, \rho \in \tau(\Sigma \cup V)$, $\lambda \notin V$, and

$Var(\rho) \cup Var(C) \subseteq Var(\lambda)$. We will often write just $\mathcal{R}$ instead of $(\Sigma, \mathcal{R})$. If a rewrite rule has no condition we usually write $\lambda \rightarrow \rho$. A Horn equational theory $\mathcal{E}$ which satisfies the above assumptions can be viewed as a CTRS $\mathcal{R}$, where the rules are the heads (implicitly oriented from left to right) and the conditions are the respective bodies. We assume that these assumptions hold for all theories we consider in this paper.

The computation mechanism of functional logic languages is based on narrowing, an evaluation mechanism that uses unification for parameter passing [26]. Narrowing solves equations by computing unifiers with respect to a given CTRS (which we call 'program'). $O(t)$ and $\bar{O}(t)$ denote the set of occurrences and the set of nonvariable occurrences of a term t, respectively. $t_{|u}$ is the subterm at the occurrence u of t. $t[r]_u$ is the term t with the subterm at the occurrence u replaced with r. These notions extend to sequences of equations in a natural way. We denote by $\theta_{\restriction W}$ the substitution obtained from θ by restricting its domain to W. $\widehat{\theta}$ denotes the equational representation of a substitution θ. A function symbol $f/n \in \Sigma$ is irreducible iff there is no rule $(\lambda \rightarrow \rho \Leftarrow C) \in \mathcal{R}$ such that f occurs as the outermost function symbol in λ, otherwise it is a defined function symbol. In theories where the above distinction is made, the signature Σ is partitioned as $\Sigma = \mathcal{C} \biguplus \mathcal{F}$, where $\mathcal{C}$ is the set of irreducible function symbols (or constructors) and $\mathcal{F}$ is the set of defined function symbols. For CTRS $\mathcal{R}$, $r \ll \mathcal{R}$ denotes that r is a new variant of a rule in $\mathcal{R}$. For more details on term rewriting and functional logic programming consult [10, 15, 17, 19].

Given a program $\mathcal{R}$, an equational goal g conditionally narrows into a goal clause g' (in symbols[4] $g \overset{\theta}{\leadsto} g'$), iff:

1. there exists $u \in \bar{O}(g)$, a standardised apart variant $(\lambda \rightarrow \rho \Leftarrow C) \ll \mathcal{R}$ and a substitution θ such that $\theta = mgu(\{g_{|u} = \lambda\})$ and $g' = (C, g[\rho]_u)\theta$, or
2. $\theta = mgu(g)$ and $g' = true$.

A *narrowing derivation* for g in $\mathcal{R}$ is defined by $g \overset{\theta}{\leadsto}{}^{*} g'$ iff $\exists \theta_1, \ldots, \exists \theta_n .\ g \overset{\theta_1}{\leadsto} \ldots \overset{\theta_n}{\leadsto} g'$ and $\theta = \theta_1 \ldots \theta_n$. We say that the derivation has length n. If $n = 0$, then $\theta = \epsilon$. A successful derivation (or refutation) for g in $\mathcal{R}$ is a narrowing derivation $g \overset{\theta}{\leadsto}{}^{*} true$, and $\theta_{\restriction Var(g)}$ is called a computed answer substitution (c.a.s.) for g in $\mathcal{R}$. We define the *success set* of an equational goal g in the program $\mathcal{R}$ as: $\mathcal{O}_{\mathcal{R}}(g) = \{\theta_{\restriction Var(g)} \mid g \overset{\theta}{\leadsto}{}^{*} true, \text{ and } \theta_{\restriction Var(g)} \text{ is normalized}\}$.

Since unrestricted narrowing has quite a large search space, several strategies for controlling the selection of redexes have been devised to improve the efficiency of narrowing by getting rid of some useless derivations. A *narrowing strategy* (or *position constraint*) is any well-defined criterion which obtains a smaller search space by permitting narrowing to reduce only some chosen positions, e.g. *basic*, *innermost*, *innermost basic*, or *lazy* narrowing (see, e.g., [15]). Formally, a narrowing strategy φ is a mapping that assigns to every goal g (different from *true*) a subset $\varphi(g)$ of $\bar{O}(g)$ such that for all $u \in \varphi(g)$ the goal g is

[4] We sometimes write $g \overset{[u,r,\theta]}{\leadsto} g'$ or $g \overset{[u,\theta]}{\leadsto} g'$ to make the occurrence or the rule used to prove the narrowing step explicit.

narrowable at occurrence u. An important property of a narrowing strategy φ is completeness, meaning that the narrowing constrained by φ is still complete. In this context, completeness means that for every solution σ to a given set of equations g, a more general $\mathcal{E}$-unifier θ can be found by narrowing (i.e., a $\mathcal{E}$-unifier θ s.t. $\theta \leq_{\mathcal{E}} \sigma\ [Var(g)]$). It is well-known that the subscript $\mathcal{E}$ in $\theta \leq_{\mathcal{E}} \sigma$ can be dropped if we only consider completeness w.r.t. normalized substitutions. A survey of results about the completeness of narrowing strategies can be found in [15]. Unrestricted narrowing is complete, e.g., for confluent programs w.r.t. normalized substitutions.

3 Unfolding of Functional Logic Programs

In logic programming, unfolding is usually defined as the application of a resolution step to a subgoal in the body of a program clause in all possible ways. Transformation typically proceeds in a 'step-by-step' fashion: a call is unfolded, then the clause is deleted from the program and replaced by the unfolded clauses [6, 18, 25, 27, 28]. This technique is safe for the least Herbrand model semantics [28], for the semantics of computed answer substitutions [6, 18], and also preserves the finite failure [25, 27]. In this section we first introduce, mimicking the case of logic programming, a naïve unfolding transformation based on conditional narrowing.

Definition 1 Unfolding of a rule in a program. Let $\mathcal{R}$ be a program and $r \equiv (\lambda \rightarrow \rho \Leftarrow C) \ll \mathcal{R}$ be a program rule. Let $\{g \overset{\theta_i}{\leadsto} (C'_i, \rho'_i = y)\}_{i=1}^{n}$ be the set of all one-step narrowing derivations that perform an effective narrowing step for the goal $g \equiv (C, \rho = y)$ in $\mathcal{R}$. Then, $Unf_{\mathcal{R}}(r) = \{(\lambda\theta_i \rightarrow \rho'_i \Leftarrow C'_i) \mid i = 1 \dots n\}$.

Note that the unfolding of a rule in a program never gives back the original, unfolded rule, since we have purposely left out the one-step narrowing derivations $(C, \rho = y) \overset{\theta_i}{\leadsto} true$ that could be proved if the goal $(C, \rho = y)$ syntactically unifies. This corresponds to the naïve, intuitive idea that one has in mind about how to define the unfolding operation.

Definition 2 Unfolding of a program w.r.t. a rule. Let $\mathcal{R}$ be a program and $r \in \mathcal{R}$ be a program rule. The unfolding of $\mathcal{R}$ w.r.t. r is the program:

$$Unfold(\mathcal{R}, r) = \begin{cases} (\mathcal{R} - \{r\}) \cup Unf_{\mathcal{R}}(r) & \text{if } Unf_{\mathcal{R}}(r) \neq \emptyset \\ \mathcal{R} & \text{otherwise.} \end{cases}$$

Note that, with this definition of unfolding, even the (weaker) semantics of (ground) equational consequences of the original program is not preserved by the transformation:

Example 1. Let us consider the following program $\mathcal{R} = \{\ f(c(x)) \rightarrow f(x)\ \}$. The rhs of the rule, $f(x)$, can only be narrowed to $f(y)$ with substitution $\{x/c(y)\}$. Then, we obtain the following unfolded program $\mathcal{R}' = \{\ f(c(c(y))) \rightarrow f(y)\ \}$. Now, the equation $f(c(a)) = f(a)$ is only true in the original program.

The results in this paper show that there is a close connection between the conditions for the correctness of the partial evaluation of functional logic programs (see [4]) and the correctness of the unfolding transformation. By exploiting this relation, we identify sufficient conditions that guarantee the correctness of the unfolding transformation. Namely, the transformation is always strongly sound but the properties of *confluence, decreasingness* [15], and a sort of *closedness* are necessary for completeness. The notion of closedness was introduced in [4] for the correctness of the Partial Evaluation (PE) of functional logic programs. PE is a transformation technique which, given a program $\mathcal{R}$ and a goal g, returns a partially evaluated program $\mathcal{R}'$ which gives exactly the same answers for g (and for any goal which satisfies some specific requirements, including the closedness condition) as $\mathcal{R}$ does. Roughly speaking, a term t is S-closed if 1) t is a variable, 2) t is a constructor term, or 3) t is an instance of a term $s \in S$, with $t = s\theta$, and the terms in θ are also S-closed. Intuitively, the closedness condition guarantees that all calls that might occur during the execution of g are "covered" by some program rule of $\mathcal{R}'$.

Theorem 3 Strong soundness. *Let $\mathcal{R}$ be a program and $\mathcal{R}' = Unfold(\mathcal{R}, r)$, $r \in \mathcal{R}$. Then, we have that $\mathcal{O}_{\mathcal{R}'}(g) \subseteq \mathcal{O}_{\mathcal{R}}(g)$, for any goal g.*

Theorem 4 Completeness. *Let $\mathcal{R} \equiv \{\lambda_i \rightarrow \rho_i \Leftarrow C_i\}_{i=1}^n$ be a confluent, left-linear and decreasing program and let $L = \{\lambda_1, \ldots, \lambda_n\}$. Let $\mathcal{R}' = Unfold(\mathcal{R}, r)$, $r \in \mathcal{R}$, and g be a goal. If $\mathcal{R}$ is L-closed, then $\mathcal{R}'$ is decreasing and L-closed, and for all $\theta \in \mathcal{O}_{\mathcal{R}}(g)$, there exists $\theta' \in \mathcal{O}_{\mathcal{R}'}(g)$ such that $\theta' \leq \theta$ $[Var(g)]$.*

Roughly speaking, the condition that $\mathcal{R}$ is L-closed requires that the calls in the rhs's and in the conditions of program rules whose outermost functor is a defined function symbol not be in normal form. The extra requirement for decreasingness ensures that these calls can be finitely normalized. These conditions suffice for ensuring that the considered rule can be safely dropped from the original program in exchange for the rules that result from unfolding, without losing completeness.

In the following section, we introduce a generalized definition of unfolding which preserves completeness under less demanding conditions.

3.1 Generalized-unfolding Operation

Definition 5. Let $\mathcal{R}$ be a program and $r \equiv (\lambda \rightarrow \rho \Leftarrow C) \ll \mathcal{R}$ be a program rule. We define the generalized unfolding of r in $\mathcal{R}$ by:

$$Gen\text{-}Unf_{\mathcal{R}}(r) = Unf_{\mathcal{R}}(r) \cup \{(\lambda \rightarrow y)\theta \mid \theta = mgu(C \cup \{\rho = y\}) \not\equiv fail\}.$$

We note that the unfolding and generalized unfolding transformations coincide for the case when $mgu(C \cup \{\rho = y\}) \equiv fail$. We also note that, for unconditional rules, $mgu(\{true, \rho = y\})$ is never $fail$, which implies that unconditional rules are always reproduced in the derived program. Finally, for $r \equiv (\lambda \rightarrow \rho \Leftarrow C)$, note that if $Gen\text{-}Unf_{\mathcal{R}}(r) = \emptyset$ then there is no successful derivation starting from $(C, \rho = y)$ in $\mathcal{R}$.

The following example illustrates the previous definition and points out the difference w.r.t. Definition 2.

Example 2. Consider again the program $\mathcal{R}$ of Example 1. The generalized unfolding of the first rule of $\mathcal{R}$ is $\{f(c(x)) \to f(x),\ f(c(c(y))) \to f(y)\}$, which contains the original rule (and thus the semantics is preserved).

An unfolding transformation which would always reproduce the original unfolded rule would be trivially complete, and practically useless. This does not happen, in general, with our generalized unfolding operation. On the contrary, the use of other, seemingly equivalent, definitions of conditional narrowing, like the one that substitutes single equations by *true* as soon as they syntactically unify (instead of requiring the unification of the whole sequence [23]), would lead to generalized unfolding definitions that always reproduce the unfolded rule.

Definition 6 Generalized unfolding of a program w.r.t. a rule.
The generalized unfolding of a program $\mathcal{R}$ w.r.t. a rule is defined as:

$Gen\text{-}Unfold(\mathcal{R}, r) = (\mathcal{R} - \{r\}) \cup Gen\text{-}Unf_{\mathcal{R}}(r)$.

Given a program $\mathcal{R}$ and a rule $r \in \mathcal{R}$, $Unfold(\mathcal{R}, r)$ and $Gen\text{-}Unfold(\mathcal{R}, r)$ do not coincide for all narrowing strategies. For unrestricted conditional narrowing, e.g., $Unfold(\mathcal{R}, r) \subseteq Gen\text{-}Unfold(\mathcal{R}, r)$. Therefore, $Gen\text{-}Unfold(\mathcal{R}, r)$ is complete whenever $Unfold(\mathcal{R}, r)$ is.

In the following, we describe and prove the strong soundness and completeness of the generalized unfolding operation under easier, weaker conditions, which rely on the properties of the PE transformation of [4]. We start by formalizing an alternative characterization of $Gen\text{-}Unfold(\mathcal{R}, r)$ in terms of partial evaluation. Roughly speaking, the definition of the PE-based, generalized unfolding operation is based on the idea of partially evaluating the program w.r.t. the lhs's of the heads of program rules. The inspiration for this definition comes from [22].

In [4], a general framework for partial evaluation of functional logic programs is defined which is based on building partial narrowing trees for the goal and extracting the specialized definition —the resultants— from the non-failing root-to-leaf branches. Roughly speaking, a PE of a term s is obtained by building a finite narrowing tree for the goal $s = y$ (where $y \notin Var(s)$), and then constructing a resultant $(s\theta_i \to t_i \Leftarrow C_i)$ for each narrowing derivation $(s = y \overset{\theta_i}{\leadsto}{}^{*}\ C_i, t_i = y)$ of the tree. See [4] for a detailed definition.

Now we provide a generalized definition of unfolding, following [22].

Definition 7 Generalized unfolding of a rule using PE. Let $\mathcal{R}$ be a program and $r \equiv (\lambda \to \rho \Leftarrow C) \ll \mathcal{R}$ be a program rule. Let (f/n) be the outermost function symbol of λ. Let $s \equiv f(x_1, \ldots, x_n)$, with $x_i \neq x_j$, for all $i \neq j$. The PE-based, generalized unfolding of r in $\mathcal{R}$, $PE\text{-}Unf_{\mathcal{R}}(r)$, is a PE of the term s in $\mathcal{R}$ obtained by stopping the branches of the narrowing tree at the end of the first edge for all rules except r, and continuing the branches which use r one more level.

The following proposition is our key point for proving the correctness of the generalized unfolding transformation.

Proposition 8. *Let $\mathcal{R}$ be a program, and $r \equiv (f(t_1,\ldots,t_n) \rightarrow \rho \Leftarrow C) \ll \mathcal{R}$ be a program rule. Then, Gen-Unf$_\mathcal{R}(r) \cup (\mathcal{R}_r - \{r\})=$ PE-Unf$_\mathcal{R}(r)$, where $\mathcal{R}_r$ denotes the subset of rules $(f(t'_1,\ldots,t'_n) \rightarrow \rho' \Leftarrow C') \ll \mathcal{R}$. Also, Gen-Unfold$(\mathcal{R}, r) = (\mathcal{R} - \{r\}) \cup$ Gen-Unf$_\mathcal{R}(r) = (\mathcal{R} - \{r\}) \cup$ PE-Unf$_\mathcal{R}(r)$.*

Now, the correctness results of the unfolding transformation directly follow from the results in [4] since the closedness, linearity and independence conditions required in [4] for the correctness of PE are automatically satisfied for the set of terms partially evaluated when producing the generalized unfolding of the program $\mathcal{R}$.

Theorem 9 Completeness. *Let $\mathcal{R}$ be a canonical program, $r \in \mathcal{R}$ be a rule, $\mathcal{R}' =$ Gen-Unfold$(\mathcal{R}, r)$, and g be a goal. Then, for all $\theta \in \mathcal{O}_\mathcal{R}(g)$, there exists $\theta' \in \mathcal{O}_{\mathcal{R}'}(g)$ s.t. $\theta' \leq \theta\, [Var(g)]$.*

As a consequence of Theorem 9, generalized unfolding is complete for the semantics of the least Herbrand $\mathcal{E}$-model in canonical programs.

The strong correctness is formulated using the ultra-linearity condition [4], which means that no variable appears twice in the rhs and the condition of the rules. Ultra-linearity is quite usual in the case of unconditional programs (where it is known as *right-linearity*). For conditional programs, we plan to study whether the (weaker) conditions for the completeness of a sharing-based implementation of narrowing [16], where terms are represented by graphs and all occurrences of the same variable are shared, are sufficient to get rid of the ultra-linearity condition.

Theorem 10 Strong correctness. *Let $\mathcal{R}$ be a confluent program, $\mathcal{R}' =$ Gen-Unfold$(\mathcal{R}, r)$, $r \in \mathcal{R}$, and g be a goal. Then,*

1. (STRONG SOUNDNESS) $\mathcal{O}_{\mathcal{R}'}(g) \subseteq \mathcal{O}_\mathcal{R}(g)$.
2. (STRONG COMPLETENESS) $\mathcal{O}_\mathcal{R}(g) \subseteq \mathcal{O}_{\mathcal{R}'}(g)$, *if $\mathcal{R}$ is ultralinear.*

Partial evaluation and the unfold/fold transformational approaches have been developed rather independently. Recently, their relation has been the subject of some discussion [21, 25, 27]. In essence, PE is a *strict subset* of the unfold/fold transformation in which unfolding is the only basic transformation rule. Only a limited form of implicit folding is obtained by imposing the closedness condition [25]. In return for this, lower complexity and a more detailed understanding of control are gained using the PE approach.

4 Unfolding/Folding via Innermost Narrowing

The use of efficient forms of narrowing can significantly improve the accuracy of the specialization method and increase the efficiency of the resulting program. In this section, we formalize and study the properties of a highly efficient unfold/fold transformation based on innermost conditional narrowing.

4.1 Innermost Conditional Narrowing

An *innermost* term t is an operation applied to constructor terms, i.e. $t = f(t_1, \ldots, t_k)$, where $f \in \mathcal{F}$ and, for all $i = 1, \ldots, k$, $t_i \in \tau(\mathcal{C} \cup V)$. A CTRS is *constructor-based* (CB), if the left-hand side of each rule is an innermost term. A *constructor goal* is a goal which consists of a sequence of equations $s_i = t_i$, with $s_i, t_i \in \tau(C \cup V)$. A substitution σ is *(ground) constructor*, if $x\sigma$ is a (ground) constructor term for all $x \in Dom(\sigma)$.

A function symbol is *completely-defined* (everywhere defined), if it does not occur in any ground term in normal form, that is to say that functions are reducible on all ground terms (of an appropriate sort). $\mathcal{R}$ is said to be completely-defined (CD), if each defined function symbol is completely-defined. In a CD CTRS, the set of ground normal terms is the set of ground constructor terms $\tau(\mathcal{C})$ over $\mathcal{C}$.

Let $\varphi_{\blacktriangleleft}(g)$ be an innermost selection function, a narrowing strategy which assigns the occurrence u of an innermost subterm of g to the goal g. We formulate innermost conditional narrowing $\leadsto_{\blacktriangleleft}$ as the smallest relation satisfying:

$$(1)\quad \frac{\sigma = mgu(g) \ \wedge \ g \text{ is a constructor goal}}{g \overset{\sigma}{\leadsto}_{\blacktriangleleft} \ true}$$

$$(2)\quad \frac{u = \varphi_{\blacktriangleleft}(g) \ \wedge \ (\lambda \rightarrow \rho \Leftarrow C) \ll \mathcal{R} \ \wedge \ \sigma = mgu(\{g_{|u} = \lambda\})}{g \overset{\sigma}{\leadsto}_{\blacktriangleleft} (C, g[\rho]_u)\sigma}$$

We let $\mathcal{O}^{\blacktriangleleft}_{\mathcal{R}}(g)$ denote the *innermost success set* of g in $\mathcal{R}$, i.e., the set of all computed answer substitutions corresponding to the successful innermost conditional narrowing derivations for g in $\mathcal{R}$. Note that all answers computed by innermost conditional narrowing are normalized since they are constructor [13].

For a goal g and CB-CD, canonical program $\mathcal{R}$, innermost conditional narrowing is complete w.r.t. ground constructor solutions σ satisfying that $Var(g) \subseteq Dom(\sigma)$. This means that, given σ, there is a c.a.s. θ for $\mathcal{R} \cup \{g\}$ using $\leadsto_{\blacktriangleleft}$ such that $\theta \leq \sigma[Var(g)]$ [13].

4.2 Innermost Unfolding Transformation

Let us now formally introduce the unfolding of a program rule at one of its innermost function calls. By abuse, for a rule $r \equiv (\lambda \rightarrow \rho \Leftarrow C)$, we define $O(r) = O(C, \rho = y)$ and use it to unequivocally refer to the positions of r. We also use $r_{|u}$ and $r[t]_u$, $u \in O(r)$, with the obvious meaning.

Definition 11 Innermost unfold. Let $\mathcal{R}$ be a program and $r \equiv (\lambda \rightarrow \rho \Leftarrow C) \ll \mathcal{R}$ be a program rule. Let $u \in \bar{O}(r)$ be the occurrence of an innermost subterm of the rule r. We define the unfolding of $\mathcal{R}$ w.r.t. u and r as follows:

$Unfold^{\blacktriangleleft}(\mathcal{R}, u, r) = (\mathcal{R} - \{r\}) \cup Unf^{\blacktriangleleft}_{\mathcal{R}}(u, r)$

where $Unf^{\blacktriangleleft}_{\mathcal{R}}(u, r)$ is the set of unfolded rules which result from the one-step innermost narrowing derivations $\{(C, \rho = y) \overset{[u,\theta_i]}{\leadsto}_{\blacktriangleleft} g_i \mid i = 1, \ldots, n\}$ for $(C, \rho = y)$ in $\mathcal{R}$. By abuse, we omit the parameter u when it is fixed by $\varphi_{\blacktriangleleft}((C, \rho = y)) = u$.

The only difference w.r.t. the general definition is in the use of innermost narrowing instead of unrestricted narrowing to build the derived rules. For innermost narrowing, the selection of the innermost position to be narrowed is don't-care nondeterministic, hence we prefer to leave the unfolding position as a parameter of the innermost unfolding definition. Note that, for the innermost strategy, unfolding and general unfolding coincide.

The following theorem establishes the computational equivalence of a program and any of its innermost unfoldings, under the simple, standard requirements for the completeness of innermost conditional narrowing.

Theorem 12. *Let $\mathcal{R}$ be a CB-CD canonical program, $r \in \mathcal{R}$ a program rule, and $u \in \bar{O}(r)$ the occurrence of an innermost subterm of r. Let $\mathcal{R}' = Unfold^{\blacktriangleleft}(\mathcal{R}, u, r)$. Then, we have that $\mathcal{O}^{\blacktriangleleft}_{\mathcal{R}}(g) = \mathcal{O}^{\blacktriangleleft}_{\mathcal{R}'}(g)$, for any goal g.*

4.3 Innermost Folding Transformation

Now we introduce a folding transformation, which is intended to be the inverse of the unfolding operation, that is, an unfolding step followed by the corresponding folding step (and viceversa) is expected to give back the initial program. Roughly speaking, the folding operation consists of substituting a function call (*folding call*) for a definitionally equivalent set of calls (*folded calls*) together with a set of equational conditions. This operation is generally used in all transformation techniques in order to pack back unfolded rules and to detect implicitly recursive definitions. It is also used when partial evaluation techniques are recast in terms of unfold/fold transformations [25].

In the following, we introduce a folding transformation that can be seen as an extension to functional logic programs of the reversible folding of [25] for logic programs. We have chosen this form of folding since it exhibits the useful, pursued property that the answer substitutions computed by innermost narrowing are preserved through the transformation. We consider the investigation of more general definitions of folding as a matter of further research.

Let us now introduce the innermost folding operation. Note that it has two sources of nondeterminism. The first is in the choice of the folded calls; the second, is in the choice of a generalization (folding call) of the heads of the instantiated function definitions which are used to substitute the folded calls.

Definition 13 Innermost fold. Let $\mathcal{R}$ be a program. Let $\{r_1, \ldots, r_n\} \ll \mathcal{R}$ (the "folded rules") and $R_{def} \equiv \{r'_1, \ldots, r'_n\} \ll \mathcal{R}$ (the "folding rules") be two disjoint subsets of program rules (up to renaming), with $r'_i \equiv (\lambda'_i \to \rho'_i \Leftarrow C'_i)$, $i = 1, \ldots, n$. Let r be a rule[5], $u \in O(r)$ be a position of the rule r, and t be an *innermost* term such that, for all $i = 1, \ldots, n$:

1. $\theta_i = mgu(\{\lambda'_i = t\}) \not\equiv fail$,

[5] Roughly speaking, r is the "common skeleton" of the rules that are folded in the folding step. The occurrence u in r acts as the pointer to the "hole" where the folding call is let fall.

2. $r_i \equiv (\lambda \to \rho_i \Leftarrow C_i', C_i)\theta_i$ and $r[\rho_i']_u \equiv (\lambda \to \rho_i \Leftarrow C_i)$, and
3. for any rule $r' \equiv (\lambda' \to \rho' \Leftarrow C') \ll \mathcal{R}$ not in R_{def}, $mgu(\{\lambda' = t\}) \equiv fail$.

Then, we define the folding of $\{r_1, \ldots, r_n\}$ in $\mathcal{R}$ using R_{def} as follows:

$$Fold^{\blacktriangleleft}(\mathcal{R}, \{r_1, \ldots, r_n\}, R_{def}) = (\mathcal{R} - \{r_1, \ldots, r_n\}) \cup \{r_{fold}\}$$

where $r_{fold} \equiv r[t]_u$.

Intuitively, the folding operation proceeds in a contrary direction to the narrowing steps. In narrowing steps, for a given unifier of the redex and the lhs of the applied rule, a reduction step is performed on the instantiated redex, then the conditions of the unfolding rule are added to the unfolded one, and finally the narrowing substitution is applied. Here, first folded rules are "deinstantiated" (generalized). Next, one gets rid of the conditions of the applied folding rules, and, finally, a reduction step is performed against the reversed heads of the folding rules. The following example illustrates our notion of innermost folding.

Example 3. Let us consider the following CB-CD canonical program $\mathcal{R}$:

$$f(x) \to s(x) \Leftarrow h(s(x)) = 0 \ (r_1) \qquad num(y) \to y \Leftarrow h(y) = 0 \quad (r_3)$$
$$f(s(z)) \to s(s(0))) \Leftarrow z = 0 \quad (r_2) \quad num(s(s(z))) \to s(s(0)) \Leftarrow z = 0 \ (r_4)$$

Now, we can fold the rules $\{r_1, r_2\}$ of $\mathcal{R}$ w.r.t. $R_{def} \equiv \{r_3, r_4\}$ using $r \equiv (f(x) \to \square)$ and $t \equiv num(s(x))$, obtaining the resulting program:

$$\mathcal{R}' = \{ \quad f(x) \to num(s(x)) \quad (r_{fold})$$
$$num(y) \to y \Leftarrow h(y) = 0 \quad (r_3)$$
$$num(s(s(z))) \to s(s(0)) \Leftarrow z = 0 \ (r_4) \quad \}.$$

The above definition requires two applicability conditions for a folding step: (1) the set of folded rules and the set of folding rules are disjoint (up to renaming) and (2) the term t which replaces the folded calls is innermost. Note that the latter requirement is novel as it arises for the first time in the functional logic context, and it is a key point for proving the *reversibility* property that a folded program can be unfolded back by an innermost step.

The following lemma formalizes the reversibility condition, by showing how folding steps can be undone by appropriate innermost unfolding steps. This allows us to prove, in Theorem 15, the total correctness of the transformation.

Lemma 14 Reversibility. *Let $\mathcal{R}$ be a CB-CD, canonical program. If $\mathcal{R}' = Fold^{\blacktriangleleft}(\mathcal{R}, \{r_1, \ldots, r_n\}, R_{def})$, then there exists an occurrence $u \in \bar{O}(r_{fold})$ of an innermost term s.t. $\mathcal{R} = Unfold^{\blacktriangleleft}(\mathcal{R}', u, r_{fold})$ (up to renaming), where r_{fold} is the new rule introduced in $\mathcal{R}'$ by the innermost folding step.*

Example 4. Consider again the folded program $\mathcal{R}'$ of Example 3. If we unfold the rule r_{fold} of $\mathcal{R}'$ w.r.t. the occurrence of the innermost function call $num(s(x))$, then we get back the initial program $\mathcal{R}$.

Theorem 15 Strong correctness. *Let $\mathcal{R}$ be a CB-CD, canonical program and $\mathcal{R}' = Fold^{\blacktriangleleft}(\mathcal{R}, \{r_1, \ldots, r_n\}, R_{def})$ be a folding of $\{r_1, \ldots, r_n\}$ in $\mathcal{R}$ using R_{def}. Then, we have that $\mathcal{O}^{\blacktriangleleft}_{\mathcal{R}}(g) = \mathcal{O}^{\blacktriangleleft}_{\mathcal{R}'}(g)$, for any equational goal g.*

Theorem 12 and Theorem 15 show that it is possible to define a program transformation strategy based on our innermost unfold/fold rules preserving computed answer substitutions, which is outside the scope of this paper.

As an application of the innermost unfolding transformation, in the following section, we define a semantics based on unfolding which is able to characterize the answer substitutions computed by innermost narrowing syntactically.

5 A Semantics Modelling Computed Answers

The operational semantics of a program is a mapping from the set of programs to a set of program denotations which, given a program $\mathcal{R}$, returns a set of 'results' of the computations in $\mathcal{R}$. In this section, we formalize a *nonground* operational semantics for functional logic programs which is defined in terms of the set of all 'values' that functional expressions can compute. This semantics fully characterizes the c.a.s.'s computed by innermost conditional narrowing and it admits an alternative characterization in terms of innermost unfolding.

5.1 Operational Semantics

The following definitions are auxiliary. An equation of the form $x = y$, $x, y \in V$ is called a *trivial* equation. A *flat* equation is an equation of the form $f(x_1, \ldots, x_n) = x_{n+1}$ or $x_n = x_{n+1}$, where $x_i \neq x_j$ for all $i \neq j$. Any goal g can be transformed into an equivalent one, *flat*(g), which is flat [5].

Definition 16. Let $\mathcal{R}$ be a program. Then,

$$\begin{aligned} \mathcal{O}^{\blacktriangleleft}(\mathcal{R}) = \{(f(x_1, \ldots, x_n) = x_{n+1})\theta \mid\; & f(x_1, \ldots, x_n) = x_{n+1} \overset{\theta\;*}{\rightsquigarrow}_{\blacktriangleleft} \mathit{true} \text{ in } \mathcal{R}, \\ & \text{and } (f/n) \in \Sigma \;\}. \end{aligned}$$

The following theorem asserts that the computed answer substitutions of any (possibly conjunctive) goal g can be derived from $\mathcal{O}^{\blacktriangleleft}(\mathcal{R})$ (i.e. from the observable behaviour of single equations), by unification of the equations in the goal with the equations in the denotation. We note that this property is a kind of AND-compositionality which does not hold for unrestricted conditional narrowing [3]. We assume that the equations in the denotation are renamed apart. Equations in the goal have to be flattened first, i.e. subterms have to be unnested so that the term structure is directly accessible to unification.

Definition 17. Let g be a goal. We define the function $\mathit{split} : \mathit{Goal} \longrightarrow \mathit{Goal} \times \mathit{Goal}$ by $\mathit{split}(g) = (g_1, g_2)$, where all trivial equations of g are in g_2, and g_1 contains the other, non-trivial, equations of g.

Theorem 18. *Let $\mathcal{R}$ be a CB-CD canonical program and let g be a goal. Let $\mathit{split}(\mathit{flat}(g)) = (g_1, g_2)$. Then θ is a computed answer substitution for g in $\mathcal{R}$ iff there exists $C \equiv (e_1, \ldots, e_m) \ll \mathcal{O}^{\blacktriangleleft}(\mathcal{R})$ such that $\theta' = \mathit{mgu}(g_1, C)$ and $\theta = (\theta' \Uparrow \mathit{mgu}(g_2))\; [\mathit{Var}(g)]$.*

Theorem 18 shows that $\mathcal{O}^{\blacktriangleleft}(\mathcal{R})$ is a fully abstract semantics w.r.t. computed answer substitutions, i.e. two programs $\mathcal{R}_1$ and $\mathcal{R}_2$ with $\mathcal{O}^{\blacktriangleleft}(\mathcal{R}_1) = \mathcal{O}^{\blacktriangleleft}(\mathcal{R}_2)$ (up to renaming) cannot produce different computed answers. Moreover, $\mathcal{O}^{\blacktriangleleft}(\mathcal{R})$ can be viewed as a (possibly infinite) set of 'unit' clauses, and the computed answer substitutions for g in $\mathcal{R}$ can be determined by 'executing' $\mathit{flat}(g)$ in the program $\mathcal{O}^{\blacktriangleleft}(\mathcal{R})$ by syntactic unification, as if the equality symbol were an ordinary predicate. We note that in [2] a similar operational semantics was defined for basic conditional narrowing.

5.2 Unfolding Semantics

Now we introduce an unfolding semantics for functional logic programs, based on the unfolding transformation we have defined in Section 4.2. First, we define the unfolding of a program as follows.

Definition 19 Unfolding of a program. The unfolding of a program $\mathcal{R}$ is the program obtained by unfolding the rules of $\mathcal{R}$ w.r.t. $\mathcal{R}$. Formally,

$$\mathit{Unfold}^{\blacktriangleleft}(\mathcal{R}) = \bigcup_{r \in \mathcal{R}} \{\mathit{Unf}^{\blacktriangleleft}_{\mathcal{R}}(r) \mid r \in \mathcal{R}\}.$$

Now, the repeated application of unfolding leads to a sequence of equivalent programs which is inductively defined as follows.

Definition 20. The sequence:

$$\begin{aligned} \mathcal{R}^0 &= \mathcal{R} \\ \mathcal{R}^{i+1} &= \mathit{Unfold}^{\blacktriangleleft}(\mathcal{R}^i),\ i \geq 0 \end{aligned}$$

is called the innermost unfolding sequence starting from $\mathcal{R}$.

We notice that, as an immediate consequence of Theorem 12, we have that $\mathcal{O}^{\blacktriangleleft}(\mathcal{R}^i) = \mathcal{O}^{\blacktriangleleft}(\mathcal{R}^{i+1})$, $i \geq 0$, for CB-CD, canonical programs.

The unfolding semantics of a program is defined as the limit of the (top-down) unfolding process described in Definition 19. Let us now formally define the *unfolding semantics* $\mathcal{U}^{\blacktriangleleft}(\mathcal{R})$ of a program $\mathcal{R}$. The main point of this definition is in compelling the rhs's of the equations in the denotation to be constructor terms. Let $\Phi_{\mathcal{C}}$ be the set of identical equations $c(x_1, \ldots, x_n) = c(x_1, \ldots, x_n)$, for each $c/n \in \mathcal{C}$.

Definition 21. Let $\mathcal{R}$ be a program. Then,

$$\mathcal{U}^{\blacktriangleleft}(\mathcal{R}) = \Phi_{\mathcal{C}} \cup \bigcup_{i \in \omega} \{(s = d) \mid (s \to d \Leftarrow) \in \mathcal{R}^i \text{ and } d \in \tau(C \cup V)\}$$

where $\mathcal{R}^0, \mathcal{R}^1, \ldots$ is the innermost unfolding sequence starting from $\mathcal{R}$.

The following theorem is the main result of this section and it formalizes the intuitive claim that, since the unfolding rule preserves the observable properties, we have found out a useful alternative characterization of the computed answers semantics $\mathcal{O}^{\blacktriangleleft}(\mathcal{R})$ in terms of unfolding.

Theorem 22. *Let $\mathcal{R}$ be a CB-CD canonical program. Then, $\mathcal{U}^{\blacktriangleleft}(\mathcal{R}) = \mathcal{O}^{\blacktriangleleft}(\mathcal{R})$.*

6 Conclusions and Further Research

In this paper, we have considered the correctness of the unfold/fold transformations in relation to some standard semantics of functional logic programs, namely, unrestricted conditional narrowing and innermost conditional narrowing. We have taken on the systematic study of program transformations for unrestricted narrowing because it brings to light some common problems caused by the basic mechanism and not tied to the intricacies of any particular strategy. We have ascertained and exemplified general conditions that guarantee that the meaning of the program is not modified by the transformation. These conditions cover many practical cases and are easy to check, since they are mostly syntactical and do not depend on the final program, but only on the initial one.

Future investigation concerns the study of unfolding techniques based on more elaborated narrowing strategies. As logic programming unfold/fold suggests, constructor computed answer substitutions are easier to preserve by transformed programs. Since lazy narrowing only computes solutions of this kind, we hope that stronger results may be obtained with this strategy when used as operational semantics in unfold/fold transformations. The definition of a framework for combining folding and unfolding with new, useful transformation techniques is subject of ongoing research.

References

1. M. Alpuente, M. Falaschi, G. Moreno, and G. Vidal. Safe Folding/Unfolding with Conditional Narrowing. Technical Report DSIC-II/3/97, DSIC, UPV, 1997.
2. M. Alpuente, M. Falaschi, M.J. Ramis, and G. Vidal. A Compositional Semantics for Conditional Term Rewriting Systems. In H.E. Bal, editor, *Proc. of 6th Int'l Conf. on Computer Languages, ICCL'94*, pages 171–182. IEEE, New York, 1994.
3. M. Alpuente, M. Falaschi, and G. Vidal. A Compositional Semantic Basis for the Analysis of Equational Horn Programs. *Theoretical Computer Science*, 165(1):97–131, 1996.
4. M. Alpuente, M. Falaschi, and G. Vidal. Partial Evaluation of Functional Logic Programs. Technical Report DSIC-II/33/96, DSIC, UPV, 1996. Short version in *Proc. of ESOP'96*, Springer LNCS 1058, pages 45-61. Also available from URL: http://www.dsic.upv.es/users/elp/papers.html.
5. P. Bosco, E. Giovannetti, and C. Moiso. Narrowing vs. SLD-resolution. *Theoretical Computer Science*, 59:3–23, 1988.
6. A. Bossi and N. Cocco. Basic Transformation Operations which preserve Computed Answer Substitutions of Logic Programs. *Journal of Logic Programming*, 16:47–87, 1993.
7. R.M. Burstall and J. Darlington. A Transformation System for Developing Recursive Programs. *Journal of the ACM*, 24(1):44–67, 1977.
8. J. Darlington and H. Pull. A Program Development Methodology Based on a Unified Approach to Execution and Transformation. In D. Bjørner, A.P. Ershov, and N.D. Jones, editors, *Proc. of the Int'l Workshop on Partial Evaluation and Mixed Computation*, pages 117–131. North-Holland, Amsterdam, 1988.

9. N. Dershowitz. Termination of Rewriting. *Journal of Symbolic Computation*, 3(1&2):69–115, 1987.
10. N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 243–320. Elsevier, Amsterdam, 1990.
11. N. Dershowitz and U. Reddy. Deductive and Inductive Synthesis of Equational Programs. *Journal of Symbolic Computation*, 15:467–494, 1993.
12. S. Etalle and M. Gabbrielli. Modular Transformations of CLP Programs. In *Proc. of 12th Int'l Conf. on Logic Programming*. The MIT Press, 1995.
13. L. Fribourg. SLOG: a logic programming language interpreter based on clausal superposition and rewriting. In *Proc. of Second IEEE Int'l Symp. on Logic Programming*, pages 172–185. IEEE, New York, 1985.
14. M. Hanus. Efficient Implementation of Narrowing and Rewriting. In *Proc. Int'l Workshop on Processing Declarative Knowledge*, pages 344–365. Springer LNAI 567, 1991.
15. M. Hanus. The Integration of Functions into Logic Programming: From Theory to Practice. *Journal of Logic Programming*, 19&20:583–628, 1994.
16. M. Hanus. On Extra Variables in (Equational) Logic Programming. In *Proc. of 20th Int'l Conf. on Logic Programming*, pages 665–678. The MIT Press, 1995.
17. S. Hölldobler. *Foundations of Equational Logic Programming*. Springer LNAI 353, 1989.
18. T. Kawamura and T. Kanamori. Preservation of Stronger Equivalence in Unfold/Fold Logic Programming Transformation. In *Proc. Int'l Conf. on Fifth Generation Computer Systems*, pages 413–422. ICOT, 1988.
19. J.W. Klop. Term Rewriting Systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume I, pages 1–112. Oxford University Press, 1992.
20. L. Kott. Unfold/fold program transformation. In M. Nivat and J.C. Reynolds, editors, *Algebraic methods in semantics*, chapter 12, pages 411–434. Cambridge University Press, 1985.
21. M. Leuschel, D. De Schreye, and A. de Waal. A Conceptual Embedding of Folding into Partial Deduction: Towards a Maximal Integration. In M. Maher, editor, *Proc. of the Joint International Conference and Symposium on Logic Programming JICSLP'96*, pages 319–332. The MIT Press, Cambridge, MA, 1996.
22. J.W. Lloyd and J.C. Shepherdson. Partial Evaluation in Logic Programming. *Journal of Logic Programming*, 11:217–242, 1991.
23. A. Middeldorp and E. Hamoen. Completeness Results for Basic Narrowing. *Applicable Algebra in Engineering, Communication and Computing*, 5:213–253, 1994.
24. A. Middeldorp, S. Okui, and T. Ida. Lazy Narrowing: Strong Completeness and Eager Variable Elimination. *Theoretical Computer Science*, 167(1,2):95–130, 1996.
25. A. Pettorossi and M. Proietti. Transformation of Logic Programs: Foundations and Techniques. *Journal of Logic Programming*, 19,20:261–320, 1994.
26. U.S. Reddy. Narrowing as the Operational Semantics of Functional Languages. In *Proc. of 2nd Int'l Symp. on Logic Programming*, pages 138–151. IEEE, 1985.
27. H. Seki. Unfold/fold Transformation of General Logic Programs for the Well-Founded Semantics. *Journal of Logic Programming*, 16(1&2):5–23, 1993.
28. H. Tamaki and T. Sato. Unfold/Fold Transformations of Logic Programs. In *Proc. of 2nd Int'l Conf. on Logic Programming*, pages 127–139, 1984.