Higher-Order Narrowing

Christian Prehofer*
Institut für Informatik,
TU München, 80290 München, Germany
prehofer@informatik.tu-muenchen.de

Abstract

We introduce several approaches for solving higher-order equational problems by higher-order narrowing and give first completeness results. The results apply to higher-order functional-logic programming languages and to higher-order unification modulo a higher-order equational theory.

We lift the general notion of first-order narrowing to so-called higher-order patterns and argue that the full higher-order case is problematic. Integrating narrowing into unification, called lazy narrowing, can avoid these problems and can be adapted to the full higher-order case. For the second-order case, we develop a version where the needed second-order unification remains decidable. Finally we discuss a method that combines both approaches by using narrowing on higher-order patterns with full higher-order constraints.

1 Introduction

In recent years many results for first-order term rewriting have been lifted to the higher-order case. Starting with the work of Klop [12], there exist several notions of higher-order term rewriting [21, 32]. Among results for these systems are a critical pair lemma for higher-order term rewriting systems (HRS) [21], confluence of orthogonal HRS [22, 32] and termination criteria [31].

This interest in higher-order rewriting follows the progress in its applications, for instance functional languages and theorem provers, where higher-order concepts are of growing interest. In this paper we lift yet another first-order concept to the higher-order case: solving equations by narrowing. We introduce several versions of higher-order narrowing for solving higher-order equations modulo a higher-order equational theory and give first completeness results.

The well developed theory of first-order narrowing serves as a general method for R-unification, where R is a theory given by a convergent term rewriting system. For an overview see [17]. Narrowing also forms the underlying computation rule for functional-logic programming languages [26, 6]. For several of these there exist higher-order extensions [2, 3, 10, 13, 27], but to our knowledge completeness results for the higher-order case are still missing.

Recently, Qian [25] lifted the completeness of first-order narrowing strategies to higher-order patterns for first-order rules. Higher-order patterns are an important subclass of λ-terms, discovered by Miller [19], that include bound variables, but behave almost as first-order terms in most respects. However patterns are often too restrictive for many applications (see e.g. [16]). For instance, a standard example for functional programs, the function

$$map(F, cons(X, Y)) = cons(F(X), map(F, Y))$$

has the non-pattern $F(X)$ on the right-hand side. Hence rewriting with this rule may yield non-pattern terms. When narrowing a first-order term with such higher-order rules, some restrictions on the rewrite rules can guarantee that the resulting term is still first-order, as developed in [14].[1] Examples from other areas, e.g. formalizing logics, can be found in [21].

The structure of the work is as follows. The first approach we consider is the general notion of narrowing, for which many refinements exist, e.g. basic narrowing [9]. For this, Section 3 presents an abstract view of higher-order narrowing, where a problem with locally bound variables in the solutions becomes apparent. We show in Section 3.1 that the first-order notion of narrowing can be lifted to higher-order patterns and argue that it is problematic when going beyond higher-order patterns.

*Research supported by DFG grant Br 887/4-2 and by ESPRIT WG 6028, *CCL*.

[1] The permitted terms are slightly more general than first-order terms (see Sec. 3.1).

An alternative approach is lazy narrowing, discussed in Section 4, where many of the problems encountered in the first approach can be avoided, as narrowing is embedded into unification. Using new results on second-order unification [24], we present a refinement of lazy narrowing for second-order equational matching with left-linear rules, where syntactic solvability of the system remains decidable, as it is in the first-order case.

The main problems of the first approach with general narrowing come from the fact that narrowing at variable positions is needed. Section 5 shows that we can factor out this complicated case by flattening the terms to patterns plus adding some constraints. Then narrowing on the pattern part proceeds almost as in the first-order case and it remains to solve the constraints, which can be done by lazy narrowing. In that way we have a modular structure, and higher-order lazy narrowing or an equally powerful method is used only where it is really needed.

2 Preliminaries

We follow the standard notation of term rewriting, see e.g. [5]. For the standard theory of λ-calculus we refer to [8, 1] and for higher-order unification we refer to [29].

We assume the following variable conventions:

- F, G, H, P, X, Y free variables,
- a, b, c, f, g (function) constants,
- x, y, z bound variables and
- α, β, τ type variables.

Type judgements are written as $t : \tau$. Further, s and t stand for terms and u, v, w for constants or bound variables. The following grammar defines the syntax for **λ-terms**:

$$t \;=\; F \mid x \mid c \mid \lambda x.t \mid (t_1\ t_2)$$

A list of syntactic objects $s_1, \ldots, s_n$ where $n \geq 0$ is abbreviated by $\overline{s_n}$. For instance, n-fold abstraction and application are written as $\lambda\overline{x_n}.s = \lambda x_1 \ldots \lambda x_n.s$ and $a(\overline{s_n}) = ((\cdots(a\ s_1)\cdots)\ s_n)$, respectively.

Substitutions are finite mappings from variables to terms and are denoted by $\{\overline{X_n \mapsto t_n}\}$. Free and bound variables of a term t will be denoted as $\mathcal{FV}(t)$ and $\mathcal{BV}(t)$, respectively.

The **conversions in λ-calculus** are defined as:

- α-conversion:$\lambda x.t =_\alpha \lambda y.(\{x \mapsto y\}t)$
- β-conversion:$(\lambda x.s)t =_\beta \{x \mapsto t\}s$
- η-conversion: if $x \notin \mathcal{FV}(t)$, then $\lambda x.(tx) =_\eta t$

We will in general assume that terms are in long $\beta\eta$-normal form [21]. For brevity, we may write variables in η-normal form, e.g. X instead of $\lambda\overline{x_n}.X(\overline{x_n})$. We assume that the transformation into long $\beta\eta$-normal form is an implicit operation, e.g. when applying a substitution to a term.

The set of types $\mathcal{T}$ for the simply typed λ-terms is generated by a set $\mathcal{T}_0$ of base types (e.g. int, bool) and the function type constructor $\rightarrow$. Notice that $\rightarrow$ is right associative, i.e. $\alpha \rightarrow \beta \rightarrow \gamma = \alpha \rightarrow (\beta \rightarrow \gamma)$.

The **order of a type** $\varphi = \alpha_1 \rightarrow \ldots \rightarrow \alpha_n \rightarrow \beta,\ \ \beta \in \mathcal{T}_0$ is defined as

$$Ord(\varphi) = \begin{cases} 1 & \text{if } n = 0, \text{i.e. } \varphi = \beta \in \mathcal{T}_0 \\ 1 + k & \text{otherwise, where} \\ & k = max(Ord(\alpha_1), \ldots, Ord(\alpha_n)) \end{cases}$$

A **language of order** n is restricted to

- function constants of order $\leq n + 1$ and
- variables of order $\leq n$.

A substitution θ is in long $\beta\eta$-normal form if all terms in the image of θ are in long $\beta\eta$-normal form. The convention that α-equivalent terms are identified and that free and bound variables are kept disjoint (see also [1]) is used in the following. Furthermore, we assume that bound variables with different binders have different names. Define $\mathcal{D}om(\theta) = \{X \mid \theta X \neq X\}$ and $\mathcal{R}ng(\theta) = \bigcup_{X \in \mathcal{D}om(\theta)} \mathcal{FV}(\theta X)$. Two **substitutions are equal on a set of variables W,** written as $\theta =_W \theta'$, if $\theta\alpha = \theta'\alpha$ for all $\alpha \in W$. The restriction of a substitution to a set of variables W is defined as $\theta|_W\alpha = \theta\alpha$ if $\alpha \in W$ and $\theta|_W\alpha = \alpha$ otherwise. A substitution θ is **idempotent** iff $\theta = \theta\theta$. We will in general assume that substitutions are idempotent. A substitution θ' is more general than θ, written as $\theta' \leq \theta$, if $\theta = \sigma\theta'$ for some substitution σ.

We describe positions in λ-terms by sequences over natural numbers. The subterm at a **position** p in a λ-term t is denoted by $t|_p$. A term t with the subterm at position p replaced by s is written as $t[s]_p$.

The following subclass of λ-terms was introduced originally by Dale Miller [18].

Definition 2.1 A term t in β-normal form is called a **(higher-order) pattern** if every free occurrence of a variable F is in a subterm $F(\overline{u_n})$ of t such that the $\overline{u_n}$ are η-equivalent to a list of distinct bound variables.

Unification of patterns is decidable and a most general unifier exists if they are unifiable [18]. Also, the unification of a linear pattern with a second-order term is decidable and finitary, if they are variable-disjoint [24].

Examples of higher-order patterns, or patterns for short, are $\lambda x, y.F(x, y), \lambda x.f(G(\lambda z.x(z)))$, where the latter is at least third-order. Non-patterns are $\lambda x, y.F(a, y)$ and $\lambda x.G(H(x))$.

If p is a position in s then let $bv(s, p)$ be the set of all λ-abstracted variables on the path from the root of s to p. Such a path is called **rigid** if it contains no free variables. An $\overline{x_k}$**-lifter** of a term t **away from** W is a substitution $\sigma = \{F \mapsto (\rho F)(\overline{x_k}) \mid F \in \mathcal{FV}(t)\}$ where ρ is a renaming such that $\mathcal{D}om(\rho) = \mathcal{FV}(t)$, $\mathcal{R}ng(\rho) \cap W = \{\}$ and $\rho F : \tau_1 \to \cdots \to \tau_k \to \tau$ if $x_1 : \tau_1, \ldots, x_k : \tau_k$ and $F : \tau$. A term t is $\overline{x_k}$-lifted if an $\overline{x_k}$-lifter has been applied to t. For example $\{G \mapsto G'(x)\}$ is an x-lifter of $g(G)$ away from any W not containing G'.

The following definitions for Higher-Order Rewrite Systems are slightly less restrictive than the ones by Nipkow [21, 22], but are an instance of the definitions by Wolfram [33].

Definition 2.2 A **rewrite rule** is a pair $l \to r$ such that l is not η-equivalent to a free variable, l and r are long $\beta\eta$-normal forms of the same base type, and $\mathcal{FV}(l) \supseteq \mathcal{FV}(r)$. A **Higher-Order Rewrite System** (for short: **HRS**) is a set of rewrite rules. The letter R always denotes an HRS. Assuming a rule $(l \to r) \in R$ and a position p in a term s in long $\beta\eta$-normal form, a rewrite step from s to t is defined as

$$s \longrightarrow^{l \to r}_{p,\theta} t \quad \Leftrightarrow \quad s|_p = \theta l \ \wedge \ t = s[\theta r]_p .$$

In the first-order case, $t \rightsquigarrow^{l \to r}_{p,\theta} s$ is a **narrowing** step if θ is a most general unifier of $t|_p$ and l and $s = \theta t[r]_p$. For a rewrite or narrowing step we often omit some of the parameters $l \to r, p$ and θ.

3 Full higher-order narrowing

The idea of first-order narrowing is, roughly speaking, to find an instance of a term such that some subterm can be rewritten. Repeating this yields a complete method for matching modulo a theory given by a convergent rewrite system R, and for instance R-unification can easily be embedded.[2]

Since λ-calculus can express a notion of subterm, we can model narrowing in a very abstract way. Even in this very general setting we will identify a problem with locally bound variables in solutions. To handle bound variables correctly within λ-calculus, it will be necessary to guess these variables beforehand, which is clearly unsatisfactory.

We simulate a context where reduction takes place by an appropriate higher-order variable C, i.e. instead of $s \longrightarrow^{l \to r} t$ we can write $s = \theta C(l) \longrightarrow \theta C(r) = t$ for an appropriate substitution θ. This yields the following generalization of first-order narrowing, where most of the real problems are hidden in the unification.

Definition 3.1 A λ-term s **narrows** to t with the rule $l \to r$ and with the substitution θ, written as $s \rightsquigarrow^{l \to r}_{\theta} t$, if

- τ is a $\overline{y_k}$-lifter of l,
- θ is a unifier of $s =^? C(\lambda\overline{y_k}.\tau l)$, where C is a new variable of appropriate type, and
- $t = \theta C(\lambda\overline{y_k}.\tau r)$.

Instead of explicitly replacing a subterm at position p, we use β-reduction for this purpose. It would be possible to make the subterm explicit where the replacement takes place, but this considerably complicates the completeness proof. Note that l may occur repeatedly or not at all in $\theta C(l)$, i.e. $\theta s = \theta t$ is possible.

Lemma 3.2 (One Step Lifting) *Let R be an HRS and let $l \to r \in R$. Suppose we have two terms s and t with $t = \theta s$ for a substitution θ and a set of variables V such that $\mathcal{FV}(s) \cup \mathcal{D}om(\theta) \subseteq V$. If $t \longrightarrow^{l \to r}_{p} t'$ with $\mathcal{FV}(l) \cap V = \{\}$, then there exist a term s' and substitutions δ and σ such that*

- $s \rightsquigarrow^{l \to r}_{\sigma} s'$,
- $\delta s' = t'$,
- $\delta\sigma =_V \theta$ *and δ is R-normalized if θ is R-normalized,*
- $\mathcal{FV}(s') \cup \mathcal{D}om(\delta) \subseteq V - \mathcal{D}om(\sigma) \cup \mathcal{R}ng(\sigma)$.

With Lemma 3.2, completeness of narrowing can be shown similarly to the first-order case, as e.g. in [17]. For the proof of the above lemma it is important that the rewrite rule $l \to r$ has been lifted over the right number of bound variables.

Let us see by an example that the number of variables over which a rule has to be lifted cannot be determined beforehand. The problem occurs when a solution θ for a variable X contains a local λy and a rewrite step in a subterm below where y occurs has to be lifted. When narrowing the replaced subterm is made explicit in $\sigma C(l) \longrightarrow \sigma C(r)$, but y is not visible yet. With the lifting of $l \to r$ it is possible to rename bound variables in r later.

[2] To R-unify s and t, it suffices to add both a new symbol $=$ and a rule $X{=}X \ \to \ true$ and then to solve $s = t \to^? true$.

Example 3.3 Assume $R = \{h(P,a) \to g(P,a)\}$ and consider the matching problem $H(a) \to^? u(\lambda y.g(y,a))$ with the solution $\{H \mapsto \lambda x.u(\lambda y.h(y,x))\}$. When narrowing without lifting, we obtain $H(a) \leadsto^R H''(g(P',a))$, which matches $u(\lambda y.g(y,a))$, but does not subsume the above solution, as $g(P',a)$ cannot be instantiated to $g(y,a)$.

The solution is obtained here by lifting the rule over one parameter. First, one solution to the unification problem $H(a) =^? C(\lambda y.h(P(y),a))$, which is needed for the narrowing step, is

$$\{H \mapsto \lambda x.H'(\lambda y.h(P(y),x)), C \mapsto \lambda x.H'(\lambda y.x(y)).$$

Then we have $H(a) \leadsto^R H'(\lambda y.g(P(y),a))$ and the matching problem can be solved with the substitution $\{H' \mapsto \lambda x.u(x), P \mapsto \lambda x.x\}$. In the general case, the solution to H may contain an arbitrary number of locally bound variables, such as y here, but the need to lift over these variables is not visible when looking at $H(a)$. To obtain completeness for this definition of narrowing, we thus have to guess locally bound variables, at least in our framework.

The above notion of narrowing is not of great computational interest. For instance, there is little hope to find cases where even the application of narrowing is decidable.

3.1 Narrowing on patterns

In this section we show that the first-order notion of narrowing can be adapted to a restricted set of λ-terms, higher-order patterns. Then, as in the first-order case, narrowing at variable positions implies that the used substitution is reducible, thus this step is redundant. We assume in this section that the rules $l \to r \in R$ are **pattern rules,** i.e. both l and r are patterns.

Definition 3.4 A **narrowing** step from a pattern s to t with a pattern rule $l \to r$ at position q with substitution θ, is defined as $s \overset{p}{\leadsto}_{q,\theta}^{l \to r} t$, where

- τ is a $\overline{y_k}$-lifter of l, where $\overline{y_k} = bv(s,q)$ and
- θ is a most general unifier of $\lambda\overline{y_k}.s|_q$ and $\lambda\overline{y_k}.\tau l$, and $t = \theta(s[\tau r]_q)$.

Here, in contrast to the last result, we only have to lift the rule $l \to r$ to the context at position q. The problem in Section 3 with locally bound variables occurs only when narrowing at variable positions, which is not needed here. When working with first-order equations, as done by Qian [25] and by Snyder [28], this lifting is not strictly needed, as the bound variables in $s|_q$ can be treated as new constants and/or ignored. This enables Qian to lift completeness of first-order narrowing strategies to patterns for first-order equations. We conjecture that most first-order narrowing strategies can also be lifted to our setting, yet not as in [25].

Completeness of narrowing follows from Lemma 3.5 as in the first-order case (e.g. [17]) and is omitted here.

Lemma 3.5 (One Step Lifting) *Let R be an HRS with pattern rules and let $l \to r \in R$. Suppose we have two patterns s and t with $t = \theta s$ for a R-normalized substitution θ, and a set of variables V such that $\mathcal{FV}(s) \cup \mathcal{D}om(\theta) \subseteq V$. If $t \longrightarrow_\varphi^{l \to r} t'$ with $\mathcal{FV}(l) \cap V = \{\}$, then there exist a term s' and substitutions δ, σ such that*

- $s \overset{p}{\leadsto}_\sigma^{l \to r} s'$ *and* $\delta s' = t'$
- $\delta\sigma =_V \theta$ *and* δ *is R-normalized*
- $\mathcal{FV}(s') \cup \mathcal{D}om(\delta) \subseteq V - \mathcal{D}om(\sigma) \cup \mathcal{R}ng(\sigma)$

A similar result has been developed independently in [14] for conditional rules. In this work, rules with pattern left-hand sides are used for narrowing on so called quasi first-order terms. These are slightly more general than first-order terms. This guarantees that the resulting term is still quasi first-order. Although this property is desirable, the restrictions in this approach appear rather ad-hoc, e.g. higher-order variables in the left-hand sides of rules may occur only directly below the outermost symbol. For instance, the example in the introduction, the function $map(F, cons(X,Y)) = \ldots$, fulfills this requirement if X and Y are first-order. Roughly speaking, when narrowing with such a rule, narrowing and rewriting coincide for these higher-order variables as they occur only at depth one on the left-hand side.

3.2 Beyond patterns

We argue in the following that it is difficult to adapt the above notion of narrowing for patterns to full λ-terms. In Example 3.3 we identified a problem with locally bound variables. This and several other problems stem from the fact that narrowing at variable positions is required, since the rewrite step we lift might have been at a redex created by β-reduction.

Example 3.6 Assuming the rewrite system

$$R_0 = \{f(f(X)) \to g(X)\},$$

narrowing at a variable position is required to find the solution $\{H \mapsto \lambda x.f(x)\}$ to the problem

$\lambda x.H(f(x)) \to^? \lambda x.g(x)$:

$$\lambda x.H(f(x)) \leadsto^{R_0}_{H \mapsto \lambda x.f(x)} \lambda x.g(x)$$

Now the question is how to define narrowing at variable positions. For instance, consider the solution $\theta = \{H \mapsto \lambda x.h(f(x), x)\}$ to the problem $\lambda x.H(f(x)) \to^? \lambda x.h(g(x), f(x))$, wrt. the R_0-reduction

$$\lambda x.h(f(f(x)), f(x)) \longrightarrow^{R_0} \lambda x.h(g(x), f(x)).$$

The naive approach, to instantiate H as little as possible, as in

$$\lambda x.H(f(x)) \leadsto^{R_0}_{H \mapsto \lambda x.H'(f(x))} \lambda x.H'(g(x)),$$

fails. The problem is that the subterm $f(x)$ is duplicated by θ and the reduction does not occur inside $f(x)$. A solution is to create a local context at this variable. Hence, we instantiate H first with $\{H \mapsto \lambda x.H''(H'(x), x)\}$. Then, after β-reduction, the subterm $H'(f(x))$ can be unified by $\{H' \mapsto \lambda x.f(x)\}$ with the left-hand side $f(f(x))$ and can be rewritten. Thus we have

$$\lambda x.H(f(x)) \leadsto^{R_0}_{H \mapsto \lambda x.H'(f(x),x)} \lambda x, y.H'(g(x), x)$$

and the solution, here $\{H' \mapsto \lambda x, y.h(x, y)\}$, is then obtained by unification.

Extending this approach to second-order narrowing and the completeness proof are highly technical and reveal further problems not discussed here. Further development is not pursued, as we believe that the approaches in the following sections are more promising.

4 Lazy narrowing

Another, more goal-directed method to solve equational problems in a top-down manner is lazy narrowing. The main idea is to integrate narrowing into unification. That is, when R-matching s with t, we start with a goal $s \to^? t$ that may be simplified to smaller goals. Then narrowing steps are performed at the root only, where the unification of the left-hand side of the rule with s again has to be done modulo R. Generalizing this to lazy higher-order narrowing yields system LN, shown in Figure 1. It should be noted that our notion of lazy narrowing is also called lazy unification [7, 15] in the first-order case.

System LN essentially consists of the rules for higher-order unification [29] plus the two narrowing rules. For instance, reconsider from Example 3.6 the R_0-matching problem

$$\lambda x.H(f(x)) \to^? \lambda x.h(g(x), f(x)),$$

where lazy narrowing yields

$$\{\lambda x.H_1(f(x)) \to^? \lambda x.g(x), \lambda x.H_2(f(x)) \to^? \lambda x.f(x)\}$$

by the imitation $\{H \mapsto \lambda y.h(H_1(y), H_2(y))\}$. Then the second goal can be solved by projection, and the first by Lazy Narrowing at Variable with $\{H_1 \mapsto \lambda y.f(H_{11}(y))\}$ to $\{\lambda x.H_{11}(f(x)) \to^? \lambda x.f(x), \ldots\}$, followed by several higher-order unification steps.

Let $s \overset{?}{\leftrightarrow} t$ stand for $s \to^? t$ or $t \to^? s$. Observe that the first five rules in Figure 1 apply symmetrically as well, in contrast to the two narrowing rules. For a sequence $\Rightarrow^{\theta_1} \ldots \Rightarrow^{\theta_n}$ of LN steps, we write $\overset{*}{\Rightarrow}^{\theta}$, where $\theta = \theta_n \ldots \theta_1$. A goal is called **flex-flex** if it is of the form $\lambda\overline{x_k}.Y(\overline{t_j}) \to^? \lambda\overline{x_k}.Y'(\overline{t'_l})$.

The completeness proof of system LN is built upon the completeness proof of higher-order unification.

Theorem 4.1 (Completeness of LN) *If $s \to^? t$ has solution θ, i.e. $\theta s \overset{*}{\longrightarrow}^R \theta t$, then $\{s \to^? t\} \overset{*}{\Rightarrow}^{\delta}_{LN} \{F\}$ such that δ is more general [3] than θ and F is a set of flex-flex goals.*

Compared to the approach in Section 3, many problems are now taken care of by higher-order unification. For instance, locally bound variables in a solution are computed in a outside-in manner before the inner Lazy Narrowing step needs to lift over these. Furthermore, flex-flex pairs can express a possibly infinite number of solutions. This is already very useful for higher-order unification, but even more for higher-order equational unification. In contrast, with general narrowing it is hard to control narrowing at variable positions. The corresponding goals in lazy narrowing can often be delayed as flex-flex pairs. For instance, consider the goal $\lambda x.c(F(f(x))) \to^? \lambda x.c(G(x))$ wrt. R_0, where lazy narrowing stops after one decomposition step, whereas general narrowing may blindly narrow at $F(\ldots)$.

4.1 Decidability of goal systems

In this section we show how to balance a system of goals such that the syntactic solvability (wrt. the conversions of λ-calculus) remains decidable.

Assumption. We assume in this section a second-order HRS R where all left-hand sides are linear patterns. Furthermore, we assume that all bound variables are of base type. In case a goal contains a

[3] Modulo the newly added variables.

Delete

$$\{t \to^? t\} \cup S \quad \Rightarrow \quad S$$

Decompose

$$\{\lambda\overline{x_k}.f(\overline{t_n}) \to^? \lambda\overline{x_k}.f(\overline{t'_n})\} \cup S \quad \Rightarrow \quad \{\overline{\lambda\overline{x_k}.t_n \to^? \lambda\overline{x_k}.t'_n}\} \cup S$$

Eliminate

$$\{F \stackrel{?}{\leftrightarrow} \lambda\overline{x_k}.t\} \cup S \quad \Rightarrow^\theta \quad \theta S \text{ if } F \notin \mathcal{FV}(\lambda\overline{x_k}.t) \text{ and where } \theta = \{F \mapsto \lambda\overline{x_k}.t\}$$

Imitate

$$\{\lambda\overline{x_k}.F(\overline{t_n}) \stackrel{?}{\leftrightarrow} \lambda\overline{x_k}.f(\overline{t'_m})\} \cup S \quad \Rightarrow^\theta \quad \overline{\{\lambda\overline{x_k}.H_m(\overline{\theta t_n}) \stackrel{?}{\leftrightarrow} \lambda\overline{x_k}.\theta t'_m)\}} \cup \theta S$$
where $\theta = \{F \mapsto \lambda\overline{x_n}.f(\overline{H_m(\overline{x_n})})\}$
and $\overline{H_m}$ are new variables

Project

$$\{\lambda\overline{x_k}.F(\overline{t_n}) \stackrel{?}{\leftrightarrow} \lambda\overline{x_k}.v(\overline{t'_m})\} \cup S \quad \Rightarrow^\theta \quad \{\lambda\overline{x_k}.\theta t_i(\overline{H_j(\overline{t_n})}) \stackrel{?}{\leftrightarrow} \lambda\overline{x_k}.v(\overline{\theta t'_m})\} \cup \theta S$$
where $\theta = \{F \mapsto \lambda\overline{x_n}.x_i(\overline{H_j(\overline{x_n})})\}$,
$t_i : \overline{\tau_j} \to \tau_0$ and $\overline{H_j}$ are new variables

Lazy Narrowing

$$\{\lambda\overline{x_k}.f(\overline{t_n}) \to^? \lambda\overline{x_k}.t\} \cup S \quad \Rightarrow \quad \{\overline{\lambda\overline{x_k}.t_n \to^? \lambda\overline{x_k}.l_n}\} \cup \{\lambda\overline{x_k}.r \to^? \lambda\overline{x_k}.t\} \cup S$$
where $f(\overline{l_n}) \to r$ is an $\overline{x_k}$-lifted rule from R

Lazy Narrowing at Variable

$$\{\lambda\overline{x_k}.H(\overline{t_n}) \to^? \lambda\overline{x_k}.t\} \cup S \quad \Rightarrow \quad \{\lambda\overline{x_k}.H(\overline{t_n}) \to^? \lambda\overline{x_k}.l\} \cup \{\lambda\overline{x_k}.r \to^? \lambda\overline{x_k}.t\} \cup S$$
where $l \to r$ is a $\overline{x_k}$-lifted rule from R

Figure 1: System LN for Lazy Narrowing

second-order bound variable, lifting over this variable may yield a third-order term, which we avoid here for simplicity.

We first define an ordering on goals:

Definition 4.2 We write $s \to^? s' \ll t \to^? t'$, if $\exists X.X \in \mathcal{FV}(s') \wedge X \in \mathcal{FV}(t)$.

The next goal is to achieve the following invariant:

Definition 4.3 A system of goals $\overline{G_n} = \{\overline{s_n \to^? t_n}\}$ is called **cycle free** if the transitive closure of $\ll$ is a strict partial ordering on $\overline{G_n}$ and **right isolated** if every variable occurs at most once on the right-hand sides of $\overline{G_n}$. Furthermore, $\overline{G_n}$ is called **simple** if all right-hand sides $\overline{t_n}$ are patterns and $\overline{G_n}$ is both cycle free and right isolated.

For instance, to solve a matching problem $\{s \to^? t\}$ we may wlog. assume that t is ground, thus the system is simple. Solving a single goal $l \to^? r$ of a simple system by pure unification is decidable [24], since r is a linear pattern and l and r share no variables. Furthermore, in a simple system, no occurs check is needed, e.g. $P \to^? c(P)$ cannot occur. This extends to the full system of goals since no cycles are allowed.

The next theorem shows that simple systems are closed under the rules LN. For the Decomposition rule and the two narrowing rules, the proof follows easily from the form of the goals in a simple system and from the restriction on the rules. The Imitation and Projection substitutions introduce new variables and hence do not create cycles. The elimination rule requires a few case distinctions. For instance, when eliminating a goal of the form $t \to^? P$, the variable P may not occur in any other goal on the right hand side.

Theorem 4.4 *Assume an HRS R where all left-hand sides are linear patterns. If G is a simple system of goals then applying LN with R preserves this property.*

In order to check if a second-order simple system is solvable by unification only, we iteratively solve maximal (wrt. $\ll$) goals with LN. That is, if $s \to^? t$ is a maximal goal, then t is a linear pattern and the free variables in t may not occur elsewhere. Then solving this goal with LN (without the narrowing rules) terminates with a set of flex-flex pairs, all of which are of the form

$$\lambda\overline{x_n}.t = \lambda\overline{x_n}.G(\overline{y_j}),$$

where G does not occur elsewhere. Such pairs can be finitely solved as shown in [24]. It remains to be seen that this solution preserves the property that the remaining system is simple: all solutions for $F \in \mathcal{FV}(t)$ are of the form $\{F \mapsto \lambda\overline{x_k}.F'(\overline{z_j})\}$, where $\{\overline{z_j}\} \subseteq \{\overline{x_k}\}$ and F' is a new variable of appropriate type. Hence when applying this solution to the remaining equations, the system remains simple, as G does not occur elsewhere.

Theorem 4.5 *Solving a simple second-order goal system $\overline{G_n}$ by unification is decidable and yields only a finite number of solutions.*

Thus we have achieved that divergence of simple systems only stems from the lazy narrowing rules, as in the first-order case. This is important for practical applications, and may be a good starting point for functional-logic programming languages.

Simple systems also have the advantage that it is easy to see if a system is in solved form:

Theorem 4.6 *If a simple system of goals is of the form $X_1 \stackrel{?}{\leftrightarrow} t_1, \ldots, X_n \stackrel{?}{\leftrightarrow} t_n$, where all $\overline{X_n}$ are distinct, then it is solvable.*

It can be shown that the above solved form is equivalent to dag-solved form [11] in the first-order case, but notice that the $\ll$-ordering does not correspond to the ordering needed for dag-solved form.

With the above completeness result for LN it remains for future research to develop more refined and incremental strategies, e.g. that behave like in the first-order case as much as possible. For instance, if a goal of the form $\lambda\overline{x_k}.Y(\overline{t_j}) \rightarrow^? \lambda\overline{x_k}.t$ is solvable by pure unification, application of Lazy Narrowing at the variable Y could be delayed, and similarly for goals with a variable on the right. Interestingly, if $Y(\overline{t_j})$ is a pattern, $Y \notin \mathcal{FV}(t)$ and the goal is solvable by unification, then it is solvable for all instances of $\lambda\overline{x_k}.t$, thus generalizing the first-order case. This case is particularly easy to detect in simple systems.

5 Narrowing on patterns with constraints

We have seen in Section 3 that the well-developed first-order notion of (general) narrowing is problematic when we go beyond higher-order patterns. Although lazy narrowing solves most of these problems, it would be nice to integrate some of the ideas of the former approach. For instance, lazy narrowing has similar disadvantages as lazy evaluation, which can be less efficient than eager evaluation.

An alternative approach that allows to use the general version of first-order narrowing is presented in this section. The idea is to factor out the complicated case, narrowing at variable positions, into constraints and work with the simpler pattern part as shown in Section 3.1. Compared to [23], where non-pattern unification problems are delayed in a higher-order logic programming language, we also have to solve the constraints modulo R.

The rules NC in Figure 2 work on a pair (t, C), where t is a term whose subterms with variable heads can be shifted to the goals C with rule Flatten. These can be solved with lazy narrowing as in NC or any comparable method. Then on t, narrowing at or below variable positions is not needed. The assumption is that in many applications, most (sub-)terms are patterns, such that the pattern part performs the large part of the computation.

For instance, to solve a goal $f(F(f(a))) \rightarrow^? g(a)$ wrt. R_0 as in Example 3.6, we may flatten the left-hand side to $(f(F') \rightarrow^? g(a), \{F(f(a)) \rightarrow^? F'\})$. Then the flattened term can be handled with first-order techniques, possibly yielding $\{F' \mapsto f(a)\}$. Solving the remaining constraint $F(f(a)) \rightarrow^? f(a)$ is simple, and it may not even be desirable to compute all its solutions.

It is sufficient to apply the rule Narrow only at subterms that have been flattened to patterns, as done in the completeness proof. Hence the unification needed in rule Narrow is pattern unification if the the left-hand sides are assumed to be patterns.

Notice that rule Flatten may also apply at a subterm $X(\overline{x_n})$ that is a pattern. This is clearly not needed in many cases, but it is difficult to guarantee that all substitutions involved are R-normalized. For lazy narrowing, it may even be considered an advantage (for lazy evaluation) that not only normalized substitutions are computed. This is a major problem when integrating the two approaches to narrowing. There are however simple criteria when this copying is unnecessary for normalized substitutions. For instance, if a variable Y in a tuple (t, C) occurs only once in t and once in C, no flattening on a pattern subterm $Y(\ldots)$ is needed.

Theorem 5.1 (Completeness of NC) *Assume an HRS R where all left-hand sides are higher-order patterns. If $s \rightarrow^? t$ has the solution $\theta s \stackrel{*}{\longrightarrow}^R t$, then $(s \rightarrow^? t, \{\}) \stackrel{*}{\Rightarrow}^{\delta}_{NC} (t \rightarrow^? t, C)$ such that $\delta s = t$ and δ is more general[4] than θ and the goals in C are flex-flex equations.*

[4] Modulo the newly added variables.

Solve
$(t \to^? t', C) \quad \Rightarrow^\theta \quad (t' \to^? t', \theta C)$ if $\theta t = t'$

Flatten Non-pattern
$(t \to^? t', C) \quad \Rightarrow \quad (t[X'(\overline{x_k})]_p \to^? t', \{\lambda\overline{x_k}.X(\overline{t_n}) \to^? \lambda\overline{x_k}.X'(\overline{x_k})\} \cup C)$
if p is a rigid path in t such that $t|_p = X(\overline{t_n})$
and $\overline{x_k} = bv(t,p)$

Pattern Narrow
$(t \to^? t', C) \quad \Rightarrow^\theta \quad (\theta t[r]_p \to^? t', \theta C)$ if p is a rigid path in t,
$\theta t|_p = \theta l$ where $l \to r$ is a $\overline{x_k}$-lifted rule
and $\overline{x_k} = bv(t,p)$

Lazy Narrowing
$(t \to^? t', C) \quad \Rightarrow^\theta \quad \theta(t \to^? t', C')$ if $C \Rightarrow^\theta_{LN} C'$

Figure 2: System NC for Narrowing with Constraints

Examining the completeness proof shows that rewrite steps in $\theta s \xrightarrow{*}{}^R t$ are modeled either in the pattern term or in the constraints. If the reduction from θs has certain properties, such as left-most or innermost, these also hold for the pattern, as the reductions there are part of the full reduction. Hence we conjecture that many narrowing strategies for first-order rewrite systems can be lifted to the pattern part.

6 Example

In this section we present an example for modeling symbolic differentiation where narrowing is used as a programming language. Symbolic differentiation is a standard example in many text books on Prolog [30]. In contrast to first-order programming, we can easily formalize the rules for differentiating nested functions, e.g. $\lambda x.sin(cos(x))$. This requires a notion of bound variables and is hence excluded in the standard versions for Prolog.

The naive approach to specify differentiation with an equation $\mathrm{diff}(\lambda x.F) = \lambda x.0$ fails, as the equation is not of base type. With rules of higher type, our notion of rewriting does not capture the corresponding equational theory [21]. The idea is a function d such that $d(\lambda x.v, X)$ computes the value of the differential of $\lambda x.v$ at X. When abstracting over this X, we can express the differential of a function again as a function. Although this generalization is slightly less elegant, we can now use our notion of rewriting.

Figure 3 shows how to specify differentiation with second-order equations of base type. Observe that we do not formalize the chain rule explicitly, as this would require nested free variables and our goal is to have patterns as left-hand sides, i.e. the left-hand side of the chain rule would be of the form $\lambda x.d(F(G(x)))$.

Figure 3 also includes a few rules for simple trigonometry. Observe that the right-hand sides are not patterns, hence rewriting a pattern term may yield a non-pattern. Now we can attempt to solve the query

$$\lambda x.d(\lambda y.ln(F(y)), x) \to^? \lambda x.cotan(x).$$

The solution $\{F \mapsto \lambda x.sin(x)\}$ can be found with the narrowing sequence

$$\begin{array}{ll} \lambda x.d(\lambda y.ln(F(y)), x) & \longrightarrow \\ \lambda x.d(\lambda y.F(y), x)/F(x) & \overset{*}{\leadsto} \\ \lambda x.cos(x)/sin(x) & \longrightarrow \\ \lambda x.cotan(x). & \end{array}$$

Lazy narrowing provides a more goal directed search in this example, as unification can be used earlier for simplification:

$$\begin{array}{ll} \{\lambda x.d(\lambda y.ln(F(y)), x) \to^? \lambda x.cotan(x)\} & \Rightarrow \\ \{\lambda x.d(\lambda y.F(y), x)/F(x) \to^? \lambda x.cotan(x)\} & \Rightarrow \\ \{\lambda x.d(\lambda y.F(y), x)/F(x) \to^? \lambda x.cos(x)/sin(x), & \\ \lambda x.cotan(x) \to^? \lambda x.cotan(x)\} & \overset{*}{\Rightarrow} \\ \{\lambda x.d(\lambda y.F(y), x) \to^? \lambda x.cos(x), & \\ \lambda x.F(x) \to^? \lambda x.sin(x)\} & \end{array}$$

Now the solution can be found by first solving the second goal by unification and then by rewriting the first goal.

$$
\begin{array}{lcl}
d(\lambda y.F, X) & \rightarrow & 0 \\
d(\lambda y.y, X) & \rightarrow & 1 \\
d(\lambda y.sin(F(y)), X) & \rightarrow & cos(F(X)) * d(\lambda y.F(y), X) \\
d(\lambda y.cos(F(y)), X) & \rightarrow & -1 * sin(F(X)) * d(\lambda y.F(y), X) \\
d(\lambda y.F(y) + G(y), X) & \rightarrow & d(\lambda y.F(y), X) + d(\lambda y.G(y), X) \\
d(\lambda y.F(y) * G(y), X) & \rightarrow & d(\lambda y.F(y), X) * G(X) + d(\lambda y.G(y), X) * F(X) \\
d(\lambda y.ln(F(y)), X) & \rightarrow & d(\lambda y.F(y), X)/F(X) \\
cos(X)/sin(X) & \rightarrow & cotan(X) \\
X * 1 & \rightarrow & X
\end{array}
$$

Figure 3: Rules for symbolic differentiation

7 Conclusions

This work gives a first framework for solving higher-order equations by comparing several approaches to higher-order narrowing. The general first-order notion of narrowing can be extended to higher-order patterns, but for the full higher-order case it seems less promising. Among several technical problems, handling locally bound variables in the solutions seems to be a principal problem with this approach. The alternative, lazy narrowing, avoids most of these problems by integrating equational reasoning into unification. Another alternative is to divide a higher-order R-matching problem into a pattern part, where narrowing works as in the first-order case, and a part with constraints that are not patterns. This is possible as higher-order patterns on the one side act like first-order terms, but on the other side can express bound variables in a sufficiently powerful way.

This paper prepares the ground for the integration of higher-order functional and logic programming. Whereas all major functional languages support higher-order programming, most existing approaches to functional-logic programming only allow for limited higher-order programming (see e.g. [2, 3, 10, 13, 27]) and mostly use first-order semantics.

For an HRS R with linear patterns as left-hand sides, we have introduced simple goal systems, where solvability by pure unification is decidable in the second-order case. This corresponds to the first-order case. It permits second-order functional-logic programming with decidable unification. Compared to higher-order logic programming, this functional approach lies between λ-Prolog [20], where full higher-order terms are used, and Elf [23], where non-patterns are just delayed as constraints.

Furthermore, this result can be the basis for further investigation into the decidability of second-order R-matching problems. For instance, Curien [4] presents first results on second-order E-matching for first-order theories.

Acknowledgements. The author wishes to thank Tobias Nipkow, Heinrich Hußmann, Konrad Slind and Jaco van de Pol for valuable comments and discussions, in particular Tobias Nipkow for the idea to lift rules into a context. The anonymous referees provided further helpful comments.

References

[1] Hendrik Pieter Barendregt. *The Lambda Calculus, its Syntax and Semantics.* North Holland, 2nd edition, 1984.

[2] P. G. Bosco and E. Giovannetti. IDEAL: An ideal deductive applicative language. In *Symposium on Logic Programming*, pages 89–95. IEEE Computer Society,, The Computer Society Press, September 1986.

[3] W. Chen, M. Kifer, and D. S. Warren. HiLog: A First-Order Semantics for Higher-Order Logic Programming Constructs. In Ewing L. Lusk and Ross A. Overbeek, editors, *Proceedings of the North American Conference on Logic Programming*, pages 1090–1114, Cleveland, Ohio, USA, 1989.

[4] Régis Curien. Second-order E-matching as a tool for automated theorem proving. In *EPIA '93*. Springer LNCS 725, 1993.

[5] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In Jan Van Leeuwen, editor, *Handbook of Theoretical Computer Science Volume B: Formal Models and Semantics*, pages 243–320. Elsevier, 1990.

[6] M. Hanus. The integration of functions into logic programming: A survey. 1994. To appear in Journal of Logic Programming.

[7] M. Hanus. Lazy unification with simplification. In *Proc. European Symposium on Programming*. Springer LNCS (to appear), 1994.

[8] J.R. Hindley and Jonathan P. Seldin. *Introduction to Combinators and λ-Calculus.* Cambridge University Press, 1986.

[9] Jean-Marie Hullot. Canonical forms and unification. In W. Bibel and R. Kowalski, editors, *Proceedings of 5th Conference on Automated Deduction*, pages 318–334. Springer Verlag, LNCS, 1980.

[10] M.Rodríguez-Artalejo J.C.González-Moreno, M.T.Hortalá-González. On the completeness of narrowing as the operational semantics of functional logic programming. In E.Börger, G.Jäger, H.Kleine Büning, S.Martini, and M.M.Richter, editors, *Computer Science Logic. Selected papers from CSL'92*, LNCS, pages 216–231, San Miniato, Italy, September 1992. Springer.

[11] Jean-Pierre Jouannaud and Claude Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 257–321. MIT Press, 1991.

[12] Jan Willem Klop. *Combinatory Reduction Systems*. Mathematical Centre Tracts 127. Mathematisch Centrum, Amsterdam, 1980.

[13] Hendrik C.R Lock. *The Implementation of Functional Logic Languages*. Oldenbourg Verlag, 1993.

[14] C. A. Loría-Sáenz. *A Theoretical Framework for Reasoning about Program Construction Based on Extensions of Rewrite Systems*. PhD thesis, Univ. Kaiserslautern, December 1993.

[15] A. Martelli, G. F. Rossi, and C. Moiso. Lazy unification algorithms for canonical rewrite systems. In H. Aït-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures, Vol. 2, Rewriting Techniques*. Academic Press, 1989.

[16] Spiro Michaylov and Frank Pfenning. Higher-order logic programming as constraint logic programming. In *Position Papers for the First Workshop on Principles and Practice of Constraint Programming*, pages 221–229, Newport, Rhode Island, April 1993. Brown University.

[17] A. Middeldorp and E. Hamoen. Counterexamples to completeness results for basic narrowing. In H. Kirchner and G. Levi, editors, *Algebraic and Logic Programming: Proc. of the Third International Conference*, pages 244–258. Springer, Berlin, Heidelberg, 1992. Long version to appear in J. of Applicable Algebra in Engineering, Communication and Computing.

[18] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. In Peter Schroeder-Heister, editor, *Extensions of Logic Programming*, pages 253–281. LNCS 475, 1991.

[19] Dale Miller. Unification of simply typed lambda-terms as logic programming. In P.K. Furukawa, editor, *Proc. 1991 Joint Int. Conf. Logic Programming*, pages 253–281. MIT Press, 1991.

[20] Gopalan Nadathur and Dale Miller. An overview of λ-Prolog. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Proc. 5th Int. Logic Programming Conference*, pages 810–827. MIT Press, 1988.

[21] Tobias Nipkow. Higher-order critical pairs. In *Proceedings, Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 342–349, Amsterdam, The Netherlands, 15–18 July 1991. IEEE Computer Society Press.

[22] Tobias Nipkow. Orthogonal higher-order rewrite systems are confluent. In M.A. Bezem and Jan Friso Groote, editors, *Proc. Int. Conf. Typed Lambda Calculi and Applications*, pages 306–317. LNCS 664, 1993.

[23] Frank Pfenning. Logic programming in the LF logical framework. In Gérard Huet and Gordon D. Plotkin, editors, *Logical Frameworks*. Cambridge University Press, 1991.

[24] Christian Prehofer. Decidable higher-order unification problems. In *Automated Deduction: CADE-12 - Proc. of the 12th International Conference on Automated Deduction*, 1994. To appear.

[25] Z. Qian. Higher-order equational logic programming. In *Proc. 21st ACM Symposium on Principles of Programming Languages*, Portland, 1994.

[26] U. S. Reddy. Narrowing as the operational semantics of functional languages. In *Symposium on Logic Programming*, pages 138–151. IEEE Computer Society, Technical Committee on Computer Languages, The Computer Society Press, July 1985.

[27] Yeh-Heng Sheng. HIFUNLOG: Logic programming with higher-order relational functions. In David H. D. Warren and Peter Szeredi, editors, *Proceedings of the Seventh International Conference on Logic Programming*, pages 529–545, Jerusalem, 1990. The MIT Press.

[28] W. Snyder. Higher order E-unification. In M. E. Stickel, editor, *10th International Conference on Automated Deduction*, pages 573–587, Berlin, Heidelberg, 1990. Springer.

[29] Wayne Snyder and Jean Gallier. Higher-order unification revisited: Complete sets of transformations. *J. Symbolic Computation*, 8:101–140, 1989.

[30] Leon Sterling. *The Art of Prolog Advanced Programming Techniques*. MIT Press, 1986.

[31] J. C. van de Pol. Termination proofs for higher order rewrite systems. In *HOA 93*. Springer LNCS, 1994. To appear.

[32] Femke van Raamsdonk. Confluence and superdevelopments. In *Rewriting Techniques an Applications*, pages 168–182. LNCS 690, June 1993.

[33] D. A. Wolfram. *The Clausal Theory of Types*. Cambridge Tracts in Theoretical Computer Science 21. Cambridge University Press, 1993.