Decidable Higher-Order Unification Problems

Christian Prehofer[*]

Technische Universität München[**]

Abstract. Second-order unification is undecidable in general. Miller showed that unification of so-called higher-order patterns is decidable and unitary. We show that the unification of a linear higher-order pattern s with an arbitrary second-order term that shares no variables with s is decidable and finitary. A few extensions of this unification problem are still decidable: unifying two second-order terms, where one term is linear, is undecidable if the terms contain bound variables but decidable if they don't.

1 Introduction

Higher-order unification is currently used in theorem provers such as Isabelle [25], TPS [1], Nuprl[1] [4] and others. The success of λ-Prolog [22] has shown the utility of higher-order constructs for programming. Other applications of higher-order unification include program synthesis [13] and machine learning [14, 6]. In this paper we consider the unification of a linear λ-term with an arbitrary second-order λ-term and develop several classes where this unification problem is decidable.

We start with an overview of the existing decidability results for higher-order unification problems in Figure 1. The column labeled Monadic refers to the unification of terms with unary function symbols only. A simply typed λ-term is a higher-order pattern, if all its free variables only have distinct bound variables as arguments. Dale Miller, as indicated in the column labeled Patterns, recently showed that unification of higher-order patterns is decidable and unitary.[2]

Section 3 reviews a set of transformation rules for full higher-order unification. Then we show in Section 4 that unification of linear higher-order patterns with an arbitrary second-order term is decidable and finitary, if the two terms share no variables. In particular, we do not have to resort to pre-unification, as equations with variables as outermost symbols on both sides (flex-flex) pairs can be finitely solved in this case. Further extensions are discussed in Section 5. The most general extension, unifying two second-order terms where one term is linear, is undecidable if the terms contain bound variables and decidable otherwise.

[*] Research supported by the DFG under grant Br 887/4-2, *Deduktive Programmentwicklung* and by ESPRIT WG 6028, *CCL*.

[**] Full Address: Fakultät für Informatik, Technische Universität München, 80290 München, Germany. E-mail: prehofer@informatik.tu-muenchen.de

[1] Nuprl uses only second-order pattern matching.

[2] We will adopt a relaxed notion of patterns, where unification is only finitary.

Order	Unification Problem			
	Unification	Patterns	Monadic	Matching
1	decidable			
2	undecidable Goldfarb ’81 [12] Farmer ’91 [10]	⋮	decidable Farmer ’88 [9]	decidable Huet ’73 [17, 16]
3	undecidable Huet ’73 [17, 16] Lucchesi ’72 [20]	⋮	undecidable Narendran ’90 [23]	decidable G. Dowek ’92 [7]
∞	⋮	decidable D. Miller ’91[21]	⋮	? Wolfram ’92 [31]

Fig. 1. Decidability of Higher-Order Unification

2 Notation and Basic Definitions

The following notation of λ-calculus are used in the sequel. For the standard theory of λ-calculus we refer to [15, 2]. 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,
- α, β types.

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 $\overline{s_n}$. We will use n-fold abstraction and application, i.e.

$$\lambda \overline{x_m}.f(\overline{s_n}) = \lambda x_1 \ldots \lambda x_m.((\cdots(f\; s_1)\cdots)\; s_n)$$

Substitutions are finite mappings from variables to terms and are denoted by $\{\overline{X_n \mapsto t_n}\}$. We assume the following standard conversions in λ-calculus:

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 $\lambda x.(tx) =_\eta t$

A term in β-normal form is in long $\beta\eta$-normal form if it is η-expanded [30]. For our proofs we assume that terms are in long $\beta\eta$-normal form, for brevity we sometimes use η-normal form, which is denoted by $t{\downarrow}_\eta$. We assume that this transformation into long $\beta\eta$-normal form is an implicit operation, e.g. occurs when applying a substitution to a term. The **head** of a term $\lambda\overline{x_k}.v(\overline{t_n})$ is defined as $Head(\lambda\overline{x_k}.v(\overline{t_n})) = v$. **Free** and **bound variables** of a term t will be denoted as $\mathcal{FV}(t)$ and $\mathcal{BV}(t)$, respectively. We describe the subterm at a **position** p in a λ-term t by $t|_p$. A (sub-)term $t|_p$ is **ground**, if no free variables of t occur in $t|_p$. A variable is **isolated** if it occurs only once (in a term or in a system of equations). A term is **linear** if no free variable occurs repeatedly.

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

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

A **language of order** n is restricted to function constants of order $\leq n+1$ and variables of order $\leq n$.

Definition 1. A simply typed λ-term s is a **relaxed higher-order pattern**, if all free variables in s only have bound variables as arguments, i.e. if $X(\overline{t_n})$ is a subterm of s, then all $t_i{\downarrow_\eta}$ are bound variables.

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

In most of the existing literature [21, 24], patterns are required to have distinct bound variables as arguments to a free variable. This restriction is necessary for unitary unification, but for our purpose this is not relevant and we will henceforth work with relaxed higher-order patterns and call these patterns for brevity.

We identify α-equivalent terms and assume that free and bound variables are kept disjoint [2]. Furthermore, we assume that bound variables with different binders have different names.

3 Pre-unification by Transformations

We present in the following a version of the transformation system $\mathcal{PT}$ for higher-order unification of Snyder and Gallier [30]. More precisely, we use the primed transformations for pre-unification of Section 5 in [30], where also the omitted type constraints can be found. These transformation rules in Figure 2 work on sets of pairs of terms to be unified, written as $\{u = v, \ldots\}$. Pre-unification differs from unification by the handling of so-called flex-flex pairs. These are equations of the form $\lambda\overline{x_k}.P(\ldots) = \lambda\overline{x_k}.P'(\ldots)$, which permit in general an infinite number of incomparable unifiers but are guaranteed to have at least one unifier, e.g. $\{P \mapsto \lambda\overline{x_m}.a, P' \mapsto \lambda\overline{x_n}.a\}$. The idea of pre-unification goes back to Huet [17]. It means to handle flex-flex pairs as constraints and not to attempt to solve them explicitly.

The only place where the restriction to second-order terms simplifies the system is the last rule, projection, where x_i must be a first-order object. Hence the binding to F in this case is of the simpler form $F = \lambda\overline{x_n}.x_i$, which will be important for our results.

We have restricted the application of rule (3) slightly compared to [30]: the rule of Snyder et al. can also be applied to flex-flex equations. We have excluded

(1) **Delete**

$$\{t = t\} \cup S \Rightarrow S$$

(2) **Decompose**

$$\{\lambda\overline{x_k}.f(\overline{t_n}) = \lambda\overline{x_k}.f(\overline{t'_n})\} \cup S \Rightarrow \bigcup_{i=1,\ldots,n}\{\lambda\overline{x_k}.t_i = \lambda\overline{x_k}.t'_i\} \cup S$$

(3) **Eliminate**

$$\{F = \lambda\overline{x_k}.t\} \cup S \Rightarrow \{F = \lambda\overline{x_k}.t\} \cup \{F \mapsto \lambda\overline{x_k}.t\}S$$
if $F \notin \mathcal{FV}(\lambda\overline{x_k}.t)$ and
$Head(t)$ is not a free variable

(4′a) **Imitate**

$$\{\lambda\overline{x_k}.F(\overline{t_n}) = \lambda\overline{x_k}.f(\overline{t'_m})\} \cup S \Rightarrow \{F = \lambda\overline{x_n}.f(\overline{H_m(\overline{x_n})})\} \cup \{\lambda\overline{x_k}.F(\overline{t_n}) = \lambda\overline{x_k}.f(\overline{t'_m})\} \cup S$$

(4′b) **Project**

$$\{\lambda\overline{x_k}.F(\overline{t_n}) = \lambda\overline{x_k}.v(\overline{t'_m})\} \cup S \Rightarrow \{F = \lambda\overline{x_n}.x_i(\overline{H_m(\overline{x_n})})\} \cup \{\lambda\overline{x_k}.F(\overline{t_n}) = \lambda\overline{x_k}.v(\overline{t'_m})\} \cup S$$
where v is a constant or bound variable

Fig. 2. System $\mathcal{PT}$ for Higher-order Pre-unification

this case as it is not necessary for completeness (the same is done in the algorithm presented by Snyder et al.).

Theorem 2 (Snyder-Gallier). *System $\mathcal{PT}$ is a sound and complete transformation system for higher-order pre-unification.*

When applying the rules of system $\mathcal{PT}$ to a set of equations, the completeness does not depend on how the equations are selected. The only branching occurs when both immitation and projection apply to some equation. This was shown by Huet [17].

Example 1. Consider the unification problem at the root of the search tree in Figure 3, which is obtained by the transformations $\mathcal{PT}$ in Figure 2. Notice that in this example all projection substitutions are of the form $\lambda x.x$. The failure cases are caused by a clash of distinct symbols and are abbreviated. Putting the substitutions of the only successful path together gives the only solution $\{F \mapsto \lambda x.g(x,x), G_1 \mapsto G, G_2 \mapsto G\}$.

4 Unifying Linear Patterns with Second-Order Terms

In this section we show that unification of second-order λ-terms with linear patterns is decidable and finitary. Let us first use system $\mathcal{PT}$ to solve the pre-unification problem.

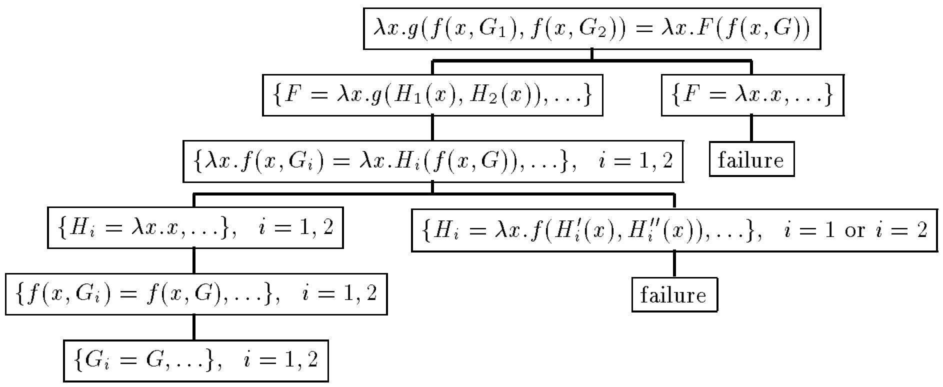
$\lambda x.g(f(x,G_1),f(x,G_2)) = \lambda x.F(f(x,G))$
$\{F = \lambda x.g(H_1(x),H_2(x)),\ldots\}$
$\{F = \lambda x.x,\ldots\}$
$\{\lambda x.f(x,G_i) = \lambda x.H_i(f(x,G)),\ldots\}, \quad i = 1,2$
failure
$\{H_i = \lambda x.x,\ldots\}, \quad i = 1,2$
$\{H_i = \lambda x.f(H_i'(x),H_i''(x)),\ldots\}, \quad i = 1$ or $i = 2$
$\{f(x,G_i) = f(x,G),\ldots\}, \quad i = 1,2$
failure
$\{G_i = G,\ldots\}, \quad i = 1,2$

Fig. 3. Search Tree with System $\mathcal{PT}$

Lemma 3. *System $\mathcal{PT}$ terminates for two variable-disjoint terms $s = t$ if s is a linear pattern and t is second-order. Furthermore, $\mathcal{PT}$ terminates with a set of flex-flex pairs of the form $\lambda\overline{x_k}.P(\overline{y_l}) = \lambda\overline{x_k}.P'(\overline{u_l})$ where all y_i are bound variables and P is isolated.*

Proof We show that system $\mathcal{PT}$ terminates for this unification problem. We start with the equation $s = t$ and apply the transformations modulo commutativity of $=$ in Figure 2. By this we achieve that after any sequence of transformations, all free variables on the left hand sides (lhs) are isolated in the system of equations, as all newly introduced variables on the lhs are linear also. The latter can be easily seen be examining the cases $(4'a)$ or $(4'b)$, the other rules are trivial. Another important invariant is that the lhs's remain patterns, which is easy to verify.

In addition, we ignore "solved" equations of the form $F = t$ or $t = F$ which are created by transformation (3). This is necessary for the termination ordering. Since transformations (1), (2) and (3) preserve the set of solutions, as shown in [30], we can assume that variable elimination (3) is applied eagerly; in particular, after a transformation $(4'a)$ or $(4'b)$, we assume that (3) is applied (with implicit β-normalization). In addition, we assume that transformation (2) is applied after $(4'a)$ and after applying $(4'b)$ to a lhs.

We use the following lexicographic termination ordering on the multiset of equations (ignoring all solved equations):

A: Compare the number of constant symbols on all lhs's, if equal
B: compare the number of occurrences of bound variables on all lhs's that are not below a free variable, if equal
C: compare the multiset of the sizes of the right-hand sides (rhs).

Now we show that the transformations reduce the above ordering:

(1) trivial
(2) A or B is reduced.
(3) Although (3) eliminates one equation, it is not trivial that it also reduces the above ordering. In particular, we do not apply (3) to flex-flex pairs, which could increase the size of some rhs if a bound variable occurs repeatedly on the lhs. Consider the possible equations (3) is applied to:
   - $\lambda\overline{x_k}.F(\overline{x_k}) = \lambda\overline{x_k}.t$: As the free variable F is isolated, A and B remain constant and C is reduced.
   - $\lambda\overline{x_k}.a(\ldots) = \lambda\overline{x_k}.F(\overline{x_k})$: The elimination of an equation with a constant a reduces A.
   - $\lambda\overline{x_k}.x_i(\ldots) = \lambda\overline{x_k}.F(\overline{x_k})$: Here B is reduced (and possibly A).

(4′a) We have two cases:
   - $\lambda\overline{x_k}.F(\overline{y_n}) = \lambda\overline{x_k}.f(\overline{t_m})$: The imitation binding for F is of the form $F = \lambda\overline{x_n}.f(\overline{H_m(\overline{x_n})})$. Now, after applying (3) and (2), we replace the above equation by a set of equations of the form $\lambda\overline{x_j}.H_i(\overline{y_n}) = \lambda\overline{x_j}.t_i$, where $i = 1,\ldots,m$. Notice that the number of constants on the lhs (A) does not increase, as all y_m are bound variables. Also, B remains unchanged. As F is isolated and hence does not occur on any right hand side, C decreases after transformation (2) is applied.
   - $\lambda\overline{x_k}.f(\overline{t_n}) = \lambda\overline{x_k}.F(\overline{u_m})$: We obtain an imitation binding as above and can apply (3) and (2). Then the number of constant symbols on the lhs's decreases, since F may not occur on the lhs's.

(4′b) We again have two cases:
   - $\lambda\overline{x_k}.F(\overline{y_n}) = \lambda\overline{x_k}.y_i(\overline{t_m})$: As $\overline{y_n}$ are bound variables, this rule applies only if the head of the rhs is a bound variables as well, say y_i. Then the case is similar to the Imitation case above, as after (4′b), transformations (3) and (2) apply.
   - $\lambda\overline{x_k}.v(\overline{t_n}) = \lambda\overline{x_k}.F(\overline{u_m})$: As we have second-order variables on the rhs, we only have projection bindings of the form $F = \lambda\overline{x_k}.x_i$. Hence the lhs's (i.e. A and B) are unchanged, whereas C decreases, as we assume terms in long $\beta\eta$-normal form.

□

So far, we have shown that pre-unification is decidable. To solve the remaining flex-flex pairs, notice that all of these are of the form

$$\lambda\overline{x_k}.P(\overline{y_m}) = \lambda\overline{x_k}.P'(\overline{u_n}),$$

where P is isolated and $\{\overline{y_m}\}$ are bound variables. Now $\lambda\overline{x_k}.P'(\overline{u_n})$ is almost an instance of the lhs, we only have to eliminate all occurrences of bound variables that are not in $\{\overline{y_m}\}$.

Example 2. Consider the pair $\lambda x,y.F(x) = \lambda x,y.F'(F''(x),F''(y))$. There are two ways to eliminate y on the rhs, i.e. $\theta_1 = \{F' \mapsto \lambda z_1,z_2.F_1'(z_1)\}$ and $\theta_2 = \{F'' \mapsto \lambda z_1.F_1''\}$, where F_1' and F_1'' are new variables.

Eliminate

$(\theta, [\lambda\overline{x_k}.P(\overline{t_n})|R], W) \Rightarrow_{el} (\tau_{P,i}\theta, \tau_{P,i}[\lambda\overline{x_k}.P(\overline{t_n})|R], W)$
if $\exists x \in W \cap \mathcal{BV}(\lambda\overline{x_k}.t_i)$

Proceed

$(\theta, [\lambda\overline{x_k}.v(\overline{t_n})|R], W) \Rightarrow_{el} (\theta, [\overline{\lambda\overline{x_k}.t_n}|R], W)$
unless v is a bound variable in W

Fig. 4. System $\mathcal{EL}$ for Eliminating Bound Variables

We first define some notation to formalize this idea. We use square brackets to denote lists, i.e. appending a list R to an element t is written as $[t|R]$. The **application of a substitution to a list,** written as $\theta[\overline{t_n}]$ is defined as $[\overline{\theta t_n}]$. For a variable F of type $\overline{\alpha_n} \rightarrow \alpha_0$ we define the **i-th parameter eliminating substitution** $\tau_{F,i}$ as

$$\tau_{F,i} = \{F \mapsto \lambda\overline{x_n}.F'(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)\},$$

where F' is a new variable of appropriate type.

The transformation rules $\Rightarrow_{el}$ in Figure 4 transform triples of the form (θ, l, W), where θ is the computed substitution, l is the list of remaining terms, and W is the set of bound variables to be eliminated.

We say system $\mathcal{EL}$ succeeds if it reduces a triple to $(\theta, [], W)$. For the flex-flex pair in Example 2 system $\mathcal{EL}$ works as follows, starting with the triple $(\{\}, [\lambda x, y.F'(F''(x), F''(y))], \{y\})$. Then $\mathcal{EL}$ can either eliminate the second argument of F' or it can proceed until the triple $(\{\}, [\lambda x, y.F''(y)], \{y\})$ is reached and then eliminate y. In these two cases, $\mathcal{EL}$ succeeds with θ_1 and θ_2, respectively, as in Example 2. All other cases fail.

Observe that system $\mathcal{EL}$ is not optimal, as it can produce the same solution twice. For instance, consider the pair $\lambda x.F = \lambda x.F'(F'(x))$. There are two different transformation sequences that yield the unifier with $\{F' \mapsto \lambda s.F''\}$. More precisely, this happens only if a bound variable occurs below nested occurrences of a variable at subtrees with the same index. Let us first show the correctness of $\mathcal{EL}$.

Lemma 4 Correctness of $\mathcal{EL}$. *Let $\lambda\overline{x_k}.P(\overline{y_m}) = \lambda\overline{x_k}.P'(\overline{u_n})$ be a flex-flex pair where $\{\overline{y_m}\} \subseteq \{\overline{x_k}\}$ and P does not occur in $P'(\overline{u_n})$. Assume further $W = \{\overline{x_k}\} - \{\overline{y_m}\}$. If $(\{\}, [P'(\overline{u_n})], W) \Rightarrow^*_{el} (\theta, [], W)$ then $\theta \cup \{P \mapsto \theta\lambda\overline{y_m}.P'(\overline{u_n})\}$ is a unifier of $\lambda\overline{x_k}.P(\overline{y_m}) = \lambda\overline{x_k}.P'(\overline{u_n})$.*

Proof We show that $\{P \mapsto \theta\lambda\overline{y_m}.P'(\overline{u_n})\}$ is a well defined substitution, i.e. all bound variables in $\theta P'(\overline{u_n})$ are locally bound or are in $\overline{y_m}$. As any successful sequence of $\mathcal{EL}$ reductions must traverse the whole term $\lambda\overline{x_k}.P'(\overline{u_n})$ to succeed, only bound variables in $\{\overline{y_m}\}$ can remain; occurrences of $\{\overline{x_k}\} - \{\overline{y_m}\}$ are either eliminated by some substitution $\tau_{P,i}$ in rule Eliminate, or the algorithm fails as the rule Proceed does not permit these bound variables. □

The next lemma states that if θ eliminates all occurrences of variables in W from $\overline{t_n}$, then there is a sequence of $\mathcal{EL}$ reductions that approximates θ.

Lemma 5. *If $\tau[\overline{t_n}] = [\overline{t_n}]$, $\mathcal{BV}(\overline{\theta t_n}) \cap W = \emptyset$, $\theta = \delta\tau$ for some substitution δ, and $\overline{t_n}$ are second-order terms, then there exist a reduction $(\tau, [\overline{t_n}], W) \Rightarrow_{el}^{*} (\theta', [], W)$ and a substitution δ' such that $\theta = \delta'\theta'$.*

Proof by induction on the sum of the sizes of the terms in $[\overline{t_n}]$. Clearly, each $\Rightarrow_{el}$ reduction reduces this sum. The base case, where $n = 0$, is trivial. We show that for each such problem some $\mathcal{EL}$ step applies and that the induction hypothesis can be applied. Depending on the form of t_1 and the conditions of the rules of $\mathcal{EL}$, we apply different rules. Assume t_1 is of the form $\lambda\overline{x_k}.P(\overline{u_m})$ and $\theta P = \lambda\overline{y_m}.t$. By our variable conventions, we can assume that $W \cap \mathcal{BV}(\theta P) = \emptyset$. As $\lambda\overline{x_k}.P(\overline{u_m})$ is a second-order term, some bound variable from W appears in $\theta\lambda\overline{x_k}.P(\overline{u_m})$ if and only if it appears in some $\theta\lambda\overline{x_k}.u_i$ where $y_i \in \mathcal{BV}(\lambda\overline{y_m}.t) = \mathcal{BV}(\theta P)$. Then let

$$i = Min\{j \mid \exists x \in \mathcal{BV}(\theta\lambda\overline{x_k}.u_j) \cap W\}.$$

Thus this set describes the indices of bound variables that may not occur in $\theta P = \lambda\overline{y_m}.t$ by assumption on θ, e.g. $y_i \notin \mathcal{BV}(\lambda\overline{y_m}.t)$. If the above set is empty and no j exists, we apply the second rule and can then safely apply the induction hypothesis.

In case i exists, we know that $\mathcal{BV}(\theta\overline{\lambda\overline{x_k}.u_n}) \cap W \subseteq \mathcal{BV}(\overline{\lambda\overline{x_k}.u_n}) \cap W$. Hence the Eliminate rule applies with $\tau_{P,i} = \{P \mapsto \lambda\overline{x_m}.P_0(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_m)\}$. Then we can apply the induction hypothesis to $(\tau_{P,i}\tau, \tau_{P,i}[\overline{t_n}], W)$: define δ' such that $\delta' X = \delta X$ if $X \neq P$ and $\delta' P_0 = \lambda y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_m.t$. Notice that δ' is well formed, as $y_i \notin \mathcal{BV}(\lambda\overline{y_m}.t)$. Clearly the premises for the induction hypothesis are fulfilled, as $\theta = \delta'\tau_{P,i}\tau$ follows from $\tau P = P$. Then the induction hypothesis assures that both $\mathcal{EL}$ succeeds with a substitution θ' and that a substitution δ'' exists such that $\theta = \delta''\theta'$.

The remaining cases of t_1 are trivial as the Proceed rule does not compute substitutions. □

Now we can show that $\mathcal{EL}$ captures all unifiers.

Lemma 6 (Completeness of $\mathcal{EL}$). *Assume θ is a unifier of a flex-flex pair of the form $\lambda\overline{x_k}.P(\overline{y_m}) = \lambda\overline{x_k}.P'(\overline{u_n})$, where $\{\overline{y_m}\} \subseteq \{\overline{x_k}\}$ and all y_m are distinct. Assume further $\lambda\overline{x_k}.P'(\overline{u_n})$ is second-order and does not contain P. Let $W = \{\overline{x_k}\} - \{\overline{y_m}\}$. Then there exist a substitution $\theta'' = \theta' \cup \{P \mapsto \theta'\lambda\overline{y_m}.P'(\overline{u_n})\}$ and a reduction $(\{\}, [\lambda\overline{x_k}.P'(\overline{u_n})], W) \Rightarrow_{el}^{*} (\theta', [], W)$ such that θ'' is more general than θ.*

Proof It is clear that any unifier must eliminate all bound variables from W on the right hand side. Then the proof follows easily from Lemma 5. □

To state the above lemma in a simple form, we did not allow repeated bound variables on the left hand side. In the next lemma we extend this result to repeated variables, which causes some technical overhead. Repeated variables may cause an additional number of distinct unifiers in each case, as there can be different permutations if a repeated variable occurs in the common instance. Consider for example the pair $\lambda x.F(x,x) = \lambda x.F'(x)$. There are the two solutions $\{F \mapsto \lambda y,z.F'(y)\}$ and $\{F \mapsto \lambda y,z.F'(z)\}$.

Theorem 7. *Assume t is a second-order λ-term and s is a linear pattern such that s shares no variables with t. Then the unification problem $s = t$ is decidable and finitary.*

Proof We first extend Lemma 6 to the case of repeated bound variables. Formally, consider the pair $\lambda\overline{x_k}.P(\overline{y_m}) = \lambda\overline{x_k}.v$ and assume some bound variables occur several times in $P(\overline{y_m})$. Assume $\mathcal{EL}$ succeeds with $(\theta, [], \{\overline{x_k} - \overline{y_m}\})$. Let $p(i,j)$ be the position of the j-th occurrence of x_i in θv. For this solution of $\mathcal{EL}$, all solutions for P are of the form $\{P \mapsto \lambda\overline{z_m}.v'\}$, where $Head(v'|_{p(i,j)}) = z_i$ and $y_i = x_i$ for all positions $p(i,j)$ of some x_i in θv and $Head(v'|_q) = Head(\theta v|_q)$ otherwise. Here the last equation allows for many permutations, as some x_j may occur repeatedly in $\overline{y_m}$. All these permutations are clearly independent from the remaining parts of the computed unifier, as P does not occur elsewhere, and can hence be easily computed.

From Lemma 3 we know that $\mathcal{PT}$ terminates with a set of flex-flex pairs, where the lhs is a pattern. Then by the extended Lemma 6 we can use $\mathcal{EL}$ to compute a complete and finite set of unifiers for some flex-flex pair, as $\mathcal{EL}$ terminates and is finitely branching. This unifier is applied to the remaining equations. Repeat this for all flex-flex pairs. This procedure terminates and works correctly as all lhs's are patterns and only have isolated variables. Notice that a flex-flex pair remains flex-flex when applying a unifier computed by $\mathcal{EL}$. $\square$

It can be shown that $\mathcal{EL}$ computes at most a quadratic number of different substitutions. Let n be the number of occurrences of variables to be eliminated and let m be the maximal number of nested free variables. Then there can be at most m distinct ways to eliminate some particular variable. As m and n are both linear in the size, the maximal number is solutions, i.e. mn, is quadratic.

However, repeated bound variables on the lhs may cause an exponential number of different solutions. Consider for instance $\lambda x.F(x,x) = \lambda x.v$, where x occurs in v exactly n times. Then there are 2^n different solutions.

It would be interesting to examine whether the computed set of unifiers is minimal, in particular for $\mathcal{EL}$. However, the most concise representation of all unifiers is still a flex-flex pair. Which representation is best clearly depends on the application. For instance, flex-flex pairs may not be satisfactory for programming languages where explicit solutions are desired. For automated theorem proving, flex-flex pairs can be advantageous as they can reduce the search space in some cases.

Observe that $\mathcal{EL}$ is not complete for the third-order case. Here, if a free variable has two arguments, one can be a function. If in some solution this function is applied to the other argument, then this function could eliminate, in the above sense, the other argument. For instance, consider the third-order pair $\lambda x,y.F(x) = \lambda x,y.F'(\lambda z.F''(z),y)$. Here $\mathcal{EL}$ would not uncover the solution $\{F' \mapsto \lambda y,z.F_0'(y(z)), F'' \mapsto \lambda x.a, F \mapsto \lambda y,z.F_0'(a)\}$.

5 Extensions

In the following sections, we will examine extensions of the above decidability result. First, notice that the linearity restriction is essential; otherwise full second-order unification can easily be embedded. But even with one linear term, this embedding still works:

Example 3. Consider the unification problem

$$\lambda x.F(f(x,G)) = \lambda x.g(f(x,t_1), f(x,t_2)),$$

where t_1 and t_2 are arbitrary second-order terms. By applying the transformations $\mathcal{PT}$ it is easy to see (compare to Example 1) that in all solutions of the above problem $F = \lambda x.g(x,x)$ and $t_1 = t_2$ must be solved, which is clearly undecidable.

Motivated by this example, we consider the following two extensions. First, we assume that arguments of free variables are either bound variables or second-order ground terms. Secondly, we consider the case where an argument of a free variable contains no bound variables. These two cases can be combined in a straightforward way, as shown in Section 5.3. Thus arguments of free variables may either be ground second-order terms or terms with no bound variables. The general case where one term is linear follows easily from Example 3:

Corollary 8. *It is undecidable to determine if two second-order terms unify, even if one is linear.*

Pre-unification of two linear second-order terms is however decidable and finitary, as shown by Dowek [8].

5.1 Ground Second-Order Arguments to Free Variables

We now loosen the restriction that one term must be a linear pattern. As long as all arguments of free variables are either bound variables or ground second-order terms, we can still solve the pre-unification problem. In particular, for the second-order case, this can be rephrased as disallowing nested free variables. However, we only solve the pre-unification problem, as the resulting flex-flex pairs are more intricate than in the last section.

Similar to the above, we present a termination ordering for a particular strategy of the $\mathcal{PT}$ transformations. We will see that in essence only one new case results from these ground second-order terms. This can be handled separately by second-order matching, which is decidable and finitary. (It is also an instance of Theorem 7.) That is, whenever such a matching problem occurs, this is solved immediately (considering all its solutions). Hence we first need a lemma about matching. A **substitution is ground** if it maps variables to ground terms only.

Lemma 9. *Solving a second-order matching problem with system $\mathcal{PT}$ yields only solutions that are ground substitutions.*

This result does not hold for the higher-order case, as noted by Dowek [8]: e.g. $\{F \mapsto \lambda x.x(Y)\}$ is a solution to $F(\lambda x.a) = a$, but no complete set of ground matchers exists. Now we can show the desired theorem:

Theorem 10. *Assume s, t are λ-terms such that t is second-order, s is linear and s shares no variables with t. Furthermore, all arguments of free variables in s are either*

- *bound variables of arbitrary type or*
- *second-order ground terms of base type.*

Then the pre-unification problem $s = t$ is decidable and finitary.

Proof We give a termination ordering for system $\mathcal{PT}$ with the same additional assumptions as in the proof of Lemma 3, i.e. eager application of rules (2) and (3). In addition, we consider solving a second-order matching problem an atomic operation, with possibly many solutions. In particular, after a projection on a lhs, this step eliminates one equation and applies a (ground) substitution to the rhs. It is easy to see that the two premises, only isolated variables and no nested free variables on the lhs's, are invariant under the transformations.

We use the following (lexicographic) termination ordering on the multiset of equations (ignoring all solved equations):

A: Compare the number of occurrences of constant symbols and of bound variables that are not below a free variable on a lhs, if equal
B: compare the number of free variables in all rhs's, if equal
C: compare the multiset of the sizes of the rhs's.

The remainder of the proof is similar to Lemma 3 and is left out for lack of space. □

It might seem tempting to apply the same technique to arguments that are third-order ground terms, as third-order matching is known to be decidable. However, there can be an infinite number of matchers and without a concise representation for these the extension of the above method seems difficult.

5.2 No Bound Variables in an Argument of a Free Variable

We say a bound variable y in $\lambda\overline{x_n}.t$ is **outside bound** if $y = x_i$ for some i. The set of all outside bound variables of a term $\lambda\overline{x_n}.t$ is written as $\mathcal{OBV}(\lambda\overline{x_n}.t) = \mathcal{BV}(\lambda\overline{x_n}.t) \cap \{\overline{x_n}\}$.

We show that the remaining case, where an argument of a free variable contains no (outside-)bound variables, can be reduced to a simpler case. This method checks unifiability, but does not give a complete set of unifiers. We use the standard notation of contexts as terms with holes, written as $C[t]$.

Theorem 11. *Assume $s = \lambda\overline{x_n}.C[H(t_1, \ldots, t_i, \ldots)]$ and t are variable-disjoint λ-terms such that s is linear. Assume further t_i contains free but no bound variables, i.e. $\mathcal{OBV}(\lambda\overline{x_n}, \overline{y_m}.t_i) = \emptyset$, where $\overline{y_m}$ are all bound variables on the path to the position of t_i in $\lambda\overline{x_n}.C[H(t_1, \ldots, t_i, \ldots)]$. Then the unification problem*

$s = t$ has a solution, iff $\lambda x_0, \overline{x_n}.C[H(t_1, \ldots, x_0, \ldots)] = \lambda x_0.t$, where x_0 does not occur elsewhere, is solvable.

Proof Consider the unification problem

$$\lambda \overline{x_n}.C[H(t_1, \ldots, t_i, \ldots)] = \lambda \overline{x_n}.u$$

where H occurs only once in $\lambda \overline{x_n}.C[H(t_1, \ldots, t_i, \ldots)]$ and t_i does not contain bound variables. Assume $\{X_1, \ldots, X_m\} = \mathcal{FV}(t_i)$. Let a solution to this problem be of the form $\{H \mapsto \lambda \overline{x_n}.t_0\} \cup \{\overline{X_m \mapsto u_m}\} \cup S$. As H does not occur elsewhere, we can construct a substitution $\theta = \{H \mapsto \lambda \overline{x_n}.\{x_i \mapsto t'\}t_0\} \cup S$, where $t' = \{\overline{X_m \mapsto u_m}\}t_i$, that is a solution to

$$\lambda x_0, \overline{x_n}.C[H(t_1, \ldots, x_0, \ldots)] = \lambda x_0, \overline{x_n}.u$$

Notice that θ is well-formed, as $\lambda \overline{x_n}, \overline{y_m}.t_i$ does not contain (outside) bound variables. The other direction is simple, since x_0 does not occur elsewhere, i.e. not in an instance of $\lambda x_0, \overline{x_n}.u$. $\square$

Notice that the above procedure only helps deciding unification problems but does not imply that pre-unification or even unification is finitary.

5.3 Putting It All Together

Now we can combine the previous results. Recall that the remaining case is undecidable in general.

Theorem 12. *Assume s, t are λ-terms such that t is second-order, s is linear and s shares no variables with t. Furthermore, all arguments of a free variable F in s are either*

- *bound variables of arbitrary type or*
- *second-order ground terms of base type or*
- *second-order terms of base type without variables bound outside of F.*

Then the unification problem $s = t$ is decidable.

Proof First apply Theorem 11 to the unification problem until s has no nested free variables. This argument can be applied repeatedly, as the lhs is linear and hence the substitutions of multiple applications do not overlap. Then Theorem 10 can be applied to decide this problem. $\square$

A special case often considered (e.g. [12]) is terms with second-order variables, but no bound variables. Then we get the following stronger result as an instance of Theorem 12:

Proposition 13. *Assume s, t are second-order λ-terms such that s is linear and shares no variables with t. Furthermore, s contains no bound variables. Then the unification problem $s = t$ is decidable.*

The above unification problems are at least NP-hard, as they subsume second-order matching, which is NP-complete [3].

6 Applications

As mentioned in the introduction, higher-order unification is currently used in several theorem provers, programming languages, and logical frameworks. With the above results we can now develop simplified and somewhat restricted versions of the above applications that enjoy decidable unification. It should be mentioned that several systems such as Elf [27] and Isabelle[3] have already resorted to higher-order patterns, where unification behaves much like the first-order case.

There is an interesting variety of applications where linearity is a common and sometimes also useful restriction. For instance, narrowing [18] is a general method to solve equations modulo a theory given by a term rewrite system. Then we can define a second-order version of narrowing with decidable unification as long as the left-hand sides of the used rules are linear patterns. This is in fact a common restriction for constructor-based narrowing [32] and for functional logic languages [19]. Using the results of this work, different versions of higher-order narrowing are developed in [28]. Usually, the lhs's of the rewrite rules in these applications are restricted and fulfill the requirement for linear patterns. For instance, we could use rules such as

$$map(F, cons(X,Y)) \longrightarrow cons(F(X), map(F,Y)).$$

Notice that systems which work only with higher-order patterns cannot express this rule, as the right-hand side is not a pattern. Then narrowing or rewriting with this rule may yield a non-pattern term and repeated narrowing needs higher-order unification with the linear left-hand side. So far, most functional logic languages even with higher order terms only use first-order unification. Interestingly, when coding functions such as *map* into predicates, as for instance done in higher-order logic programming [22], the head of the literal, e.g. $mapP(F, cons(X,Y), cons(F(X),L)) :- mapP(F,Y,L)$, is not linear. However, when invoking this rule only with goals of the form $mapP(t,t',Z)$, where Z is a fresh variable,[4] then the unification problem is decidable as it is equivalent to a unification with a linear term. Thus our results also explain to some extent why unification in higher-order logic programming rarely diverges.

Furthermore we open the way for finding decidable second-order matching problems w.r.t. higher-order equational theories. First results on second-order matching modulo first-order theories can be found in [5].

Higher-order theorem provers often work with some form of a sequent calculus, where most rules have linear premises and conclusions e.g.

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \& B}$$

Furthermore, non-linear unification problems occur mostly with rewriting, e.g. with rules such as $P\&P \longrightarrow P$. For rewriting, however, only matching is required.

[3] Isabelle still uses full higher-order pre-unification, if the terms are not patterns.

[4] Such variables are also called "output-variables" in [29]

Another application area is type inference, which is mostly based on unification, whereby decidable static type inference for programming languages is desired. In many advanced type systems such as Girard's system F [11] variables may range over functions from types to types, i.e. second-order type variables. In particular, Pfenning [26] relates type inference in the nth-order polymorphic λ-calculus with nth-order unification. Thus progress in higher-order unification may help finding classes where type inference is decidable.

Acknowledgements. The author wishes to thank Tobias Nipkow, Gilles Dowek, and the anonymous referees for valuable comments and discussions.

References

1. Peter B. Andrews, Sunil Issar, Dan Nesmith, and Frank Pfenning. The TPS theorem proving system. In M.E. Stickel, editor, *Proc. 10th Int. Conf. Automated Deduction*, pages 641–642. LNCS 449, 1990.
2. Hendrik Pieter Barendregt. *The Lambda Calculus, its Syntax and Semantics.* North Holland, 2nd edition, 1984.
3. L. D. Baxter. *The complexity of Unification.* PhD thesis, University of Waterloo, Waterloo, Canada, 1976.
4. Robert Constable, S. Allen, H. Bromly, W. Cleaveland, J. Cremer, R. Harper, D. Howe, T. Knoblock, N. Mendler, P. Panangaden, J. Sasaki, and S. Smith. *Implementing Mathematics With the Nuprl Proof Development System.* Prentice-Hall, New Jersey, 1986.
5. Régis Curien. Second-order E-matching as a tool for automated theorem proving. In *EPIA '93.* Springer LNCS 725, 1993.
6. Michael R. Donat and Lincoln A. Wallen. Learning and applying generalised solutions using higher order resolution. In E. Lusk and R. Overbeek, editors, *9th International Conference On Automated Deduction*, pages 41–61. Springer Verlag, 1988.
7. Gilles Dowek. Third order matching is decidable. In *Proceedings, Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 2–10, Santa Cruz, California, 22–25 June 1992. IEEE Computer Society Press.
8. Gilles Dowek. Personal communication, 1993.
9. W. M. Farmer. A unification algorithm for second-order monadic terms. *Annals of Pure and Applied Logic*, 39:131–174, 1988.
10. W. M. Farmer. Simple second-order languages for which unification is undecideable. *Theoretical Computer Science*, 87:25–41, 1991.
11. J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types.* Cambridge Tracts in Theoretical Computer Science 7. Cambridge University Press, 1989.
12. W. D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.
13. Masami Hagiya. Synthesis of rewrite programs by higher-order semantic unification. In S. Arikawa, S. Goto, S. Ohsuga, and T . Yokomori, editors, *Algorithmic Learning Theory*, pages 396–410. Springer, 1990.
14. Masateru Harao. Analogical reasoning based on higher-order unification. In S. Arikawa, S. Goto, S. Ohsuga, and T . Yokomori, editors, *Algorithmic Learning Theory*, pages 151–163. Springer, 1990.
15. J.R. Hindley and Jonathan P. Seldin. *Introduction to Combinators and λ-Calculus.* Cambridge University Press, 1986.

16. Gérard Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.
17. Gérard Huet. *Résolution d'équations dans les languages d'ordre 1,2,...ω.* PhD thesis, University Paris-7, 1976.
18. 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.
19. 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.
20. C. L. Lucchesi. The undecidability of the unification problem for third order languages. Technical Report CSRR 2059, University of Waterloo, Waterloo, Canada, 1972.
21. 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.
22. 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.
23. P. Narendran. Some remarks on second order unification. Technical report, Institute of Programming and Logics, Dep. of Computer Science, State Univ. of New York at Albany, 1989.
24. 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.
25. Lawrence C. Paulson. Isabelle: The next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*, pages 361–385. Academic Press, 1990.
26. F. Pfenning. Partial polymorphic type inference and higher-order unification. In *ACM Conference on Lisp and Functional Programming*, pages 153–163, Snowbird, Utah, July 1988. ACM-Press.
27. Frank Pfenning. Logic programming in the LF logical framework. In Gérard Huet and Gordon D. Plotkin, editors, *Logical Frameworks.* Cambridge University Press, 1991.
28. Christian Prehofer. Higher-order narrowing. In *Proceedings, Ninth Annual IEEE Symposium on Logic in Computer Science.* IEEE Computer Society Press, 1994. To appear.
29. U. S. Reddy. On the relationship between logic and functional languages. In D. DeGroot and G. Lindstrom, editors, *Logic Programming: Functions, Relations, and Equations*, pages 3–36. Prentice-Hall, Englewood Cliffs, NJ, 1986.
30. Wayne Snyder and Jean Gallier. Higher-order unification revisited: Complete sets of transformations. *J. Symbolic Computation*, 8:101–140, 1989.
31. D. A. Wolfram. *The Clausal Theory of Types.* Cambridge Tracts in Theoretical Computer Science 21. Cambridge University Press, 1993.
32. J.-H. You. Enumerating outer narrowing derivation for constructor-based term rewriting systems. In C. Kirchner, editor, *Unification*, pages 541–564. Academic Press, 1990. Also in J. Symbolic Computation, 1989.