Decidability of Arity-Bounded Higher-Order Matching

Manfred Schmidt-Schauß

Institut für Informatik, J.-W.-Goethe-Universität
R. Mayer-Str 11-15 Postfach 11 19 32
D-60054 Frankfurt, Germany
schauss@cs.uni-frankfurt.de
Tel: (+49)69-798-28597, Fax: (+49)69-798-28919

Abstract. The decidability of higher-order matching is a longstanding open problem. This paper contributes to this question by investigating the naïve method of generating arity-bounded unifiers in η-long lifted form and then testing whether they are solutions. Arity-bounded means that the arity of types and subtypes of terms in unifiers in lifted form are bounded by some given number N. It is shown that this generation can be enforced to terminate without losing unifiers, if compared using a variant of Padovani's behavioral equivalence. The same method also shows decidability of higher-order matching under the restriction that every subterm of a unifier in η-long β-normal form (or in β-normal form, respectively) contains at most N different free variables.
The paper also gives an improved account of a variant of Padovani's behavioral equivalence.

1 Introduction

The higher-order matching problem is the question, whether for terms s, t in the simply typed lambda calculus, where s may contain free variables, but t is a closed term, there is a substitution σ, replacing the free variables of s by closed terms, such that the $\beta\eta$-normal forms of $\sigma(s)$ and t are equal up to renaming of bound variables. The decidability of this problem is open.

Higher-order matching is required as a basic procedure for higher-order rewriting [Nip91,NP98] and has applications in implementations of higher-order logic [Pfe01,Mil91,Dow01].

There are several results on decidability of fragments of higher-order matching. Higher-order matching where right hand sides are elementary constants is decidable [Pad96,Loa97]. If only linear lambda-terms are considered, then higher-order matching is $\mathcal{NP}$-complete [dG00]. The generalization to so-called k-duplicating terms, where every bound variable in solutions occurs at most k times is also decidable [DW02]. Restricting the order of the variables to small orders makes higher-order matching decidable: if the order is 2, then higher-order matching is $\mathcal{NP}$-complete, if the order is not greater than 3, higher-order matching is decidable [Dow92], and the complexity is EXPTIME-complete [Wie99]; it is also decidable for order 4 [Pad00] using finiteness of equivalence classes and interpolation equations [Pad95]. A reformulation of higher-order matching and an algorithm using automata was given in [CJ97]. A lower non-elementary bound for higher-order matching is given in [Wie99].

A nonterminating procedure for higher-order matching was already given by [Hue75]; and a terminating algorithm by [Wol93], however, without a proof of completeness. The hand book article [Dow01] contains a recent overview on higher-order unification and higher-order matching.

A recent result, which is a hint on the complexity of the higher-order matching problem, is the undecidability result of higher-order matching if only $\alpha\beta$-equality, but no η-equality is used [Loa01].

In this paper we show that for a given number N and a higher-order matching problem Γ the arity-bounded and the fv-bounded restrictions are decidable. An arity-bounded (fv-bounded, respectively) higher-order matching problem Γ has a solution σ, if σ is in η-long lifted form (β-normal form, respectively) and if every subterm of the terms $\sigma(x)$ has at most arity N (has at most N free variables, respectively). A term is in lifted form if maximal abstractions have no free variables. It is also shown that the restriction to η-normal forms can be omitted. The restrictions do not enforce a bound on the size of substitutions nor do they guarantee a finite number of most general unifiers. It is demonstrated that the arity-bound restriction is a proper generalization of the class of k-duplicating higher-order matching problems [DW02] (see Remark 6.9 and Proposition 7.2).

Every naïve algorithmic generation of potential solutions of a higher-order matching problem has to face the problem of termination. Usually, one tries to generate substitutions where terms are in η-long lifted form or β-normal form. This guarantees that the types in the substituted terms are restricted: the order is bounded by the maximal order of the symbols in the problem, however, the arity may grow indefinitely. Bounding this arity to be not greater than a number N guarantees that the size of types in the substitution is bounded. The result of this paper implies that indeed the generation process can be stopped by a loop-check without losing unifiers if compared wrt. the Padovani-equivalence. The loop check is based on a computable upper bound on the size of equivalence classes (see Proposition 5.1. However, this does not mean that the number of most general unifiers of a higher-order matching problem is finite.

The proof requires the finiteness of equivalence classes wrt. a variant of the Padovani-equivalence [Pad00] of terms and thus motivates a simplified proof of the finiteness of the number of equivalence classes.

Increasing the bounds N and calling the decision algorithm for arity-bounded higher-order matching problem is able to enumerate all solutions of the higher-order matching problem Γ.

To show decidability of the fourth order case of higher-order matching, Padovani [Pad00] cuts solutions by first showing that relevant elementary subterms of matchers have only a bounded number of free variables, and second by also bounding the nesting depth of elementary subterms of matchers. The first argument can be easily extended to show that fourth-order matching can be seen as fv-bounded higher-order matching. Then the second argument could be replaced by using decidability of fv-bounded higher-order matching. We use a cutting method that uses only local properties of the matcher, whereas Padovani requires global information from the interpolation problem.

The paper is structured as follows. After recalling the basic notation and facts, section 3 define the behavioral and contextual equivalence wrt a given higher-order matching problem, which are shown to coincide in section 4. In section 5 a computable upper bound on the size of equivalence classes is shown, which is used in section 6 the decision algorithm is defined and proved correct.

2 Simply Typed Lambda Calculus

Definition 2.1. *The types are defined according to the grammar*

$$T ::= T_0 \mid (T \to T)$$

where T_0 represents the nonempty set of elementary types. *The symbols α, τ range over types, and ι ranges over elementary types.*

A shorter notation for types of the form $\tau = (\alpha_1 \to (\alpha_2 \ldots (\alpha_n \to \iota) \ldots))$ is $(\alpha_1 \to \alpha_2 \to \ldots \to \alpha_n \to \iota)$. The number n is called the *arity* of the type τ, denoted $ar(\tau)$, and ι is called the *target type* of τ.

We will use two signatures in the following: a finite signature Σ_0, and an infinite signature Σ_1 with $\Sigma_0 \subseteq \Sigma_1$. We assume that for $i = 0, 1$, the signature Σ_i contains elementary constants for every elementary type mentioned in the types of the symbols in Σ_i. For Σ_1 we assume that for every type τ, there are countably infinitely many constant of type τ. In order to define terms below, we assume there is a (countably) infinite set of variables $\mathcal{V}^\tau$ for every type τ of Σ_1. The set of all variables is denoted as $\mathcal{V}$

Definition 2.2. *For $j = 0, 1$, a signature Σ_j and for every type τ we define the set $Term_j^\tau$ of terms of type τ according to the grammar*

$$Term_j^\tau ::= f^\tau \mid x^\tau \mid (Term_j^{\tau' \to \tau}\ Term_j^{\tau'}) \mid \lambda x^{\tau_1}.Term_j^{\tau_2}$$

where $f \in \Sigma_j$ is a function symbol of type τ, and x^τ is a variable of type τ, and x^{τ_1} is a variable of type τ_1. The term $\lambda x^{\tau_1}.Term_j^{\tau_2}$ is only valid if $\tau_1 \to \tau_2 = \tau$. The set of all terms wrt. signature Σ_j is written as $Term_j$.

If $t \in Term_j^\tau$, we say t *has type* τ and denote this by $type(t) = \tau$. Let $ar(t) := ar(type(t))$. Terms of the form $(t_1\ t_2)$ are called *applications*, and terms of the form $\lambda x.t$ are called *abstractions*. We will write nested applications $(\ldots((t_0\ t_1)\ t_2)\ldots\ t_n)$ as $(t_0\ t_1\ t_2 \ldots t_n)$.

The notions of bound and free variables in a term and open and closed (or ground) terms are as usual. The set of free variables in a term t is denoted as

$FV(t)$. We say that s, t are α-equal ($=_\alpha$), if s and t differ only by a sequence of renamings of bound variables of equal types. To avoid too clumsy notation and to avoid distraction from the essential, we assume the *distinct variable convention*: all bound variables in terms are distinct, and whenever an operation makes it necessary to rename bound variables, this is performed.

The *head-symbol* of an application is the subterm that is in the leftmost position in the flattened representation. For example, f is the head-symbol of $(f\ t_1\ \ldots\ t_n)$.

Definition 2.3. *Several measures will be needed.*

- *The* order $ord(\tau)$ *of a type* τ *is defined as follows:*
  - $ord(\iota) = 1$.
  - *If* $\tau = \alpha_1 \to \ldots \alpha_n \to \iota$, *then* $ord(\tau) = 1 + max\{ord(\alpha_1), \ldots, ord(\alpha_n)\}$.

  The order $ord(t)$ *of a term* t *is the order of its type.*
- *The* size *of type* τ *is defined as follows:*
  - $size(\iota) = 1$
  - *If* $\tau = \alpha_1 \to \ldots \to \alpha_n \to \iota$, *then* $size(\tau) = 1 + n + \sum_{i=1}^{n}\ size(\alpha_i)$.
- *The* argsize *of a type is defined as follows:* $\text{argsize}(\tau)(\iota) = 0$ *and* $\text{argsize}(\tau_1 \to \ldots \to \tau_n \to \iota) = n + \sum\limits_{i=1}^{n}(argsize(\tau_i))$.
- *The* size *of a term* t *is defined as follows:* $size(x) = 1, size(f) = 1, size(\lambda x.t) = size(t) + 2$, *and* $size(s\ t) = 1 + size(s) + size(t)$.

A replacement $u[s/x]$ is only valid, iff $type(s) = type(x)$. The replacement $u[s/x]$ does not capture free variables, due to the distinct variable convention. A context $C[.]$ is defined by the syntax $[.] \mid (C\ t) \mid (t\ C) \mid \lambda x.C$, where $[.]$ represents the hole, t is a term and x is a variable. A *substitution* θ replaces several variables simultaneously. We consider only type-correct substitutions.

Variable renaming (α-renaming) is always used, so we will mention it only if necessary. Since we use the $\beta\eta$ rules for the simply typed lambda-calculus, there are the following equations between terms:

$$\begin{array}{lll}(\alpha) & \lambda x.t & = \lambda y.t[y/x] \quad y \text{ is a fresh variable} \\ (\beta) & ((\lambda x.t)\ s) & = t[s/x] \\ (\eta) & t & = \lambda x^\tau.(t\ x) \quad \text{if } type(t) = \tau \to \tau_1 \text{ and } x \notin FV(t).\end{array}$$

Of course we also assume that the thus defined equality $=_{\beta\eta}$ is an equivalence relation and a congruence, i.e. $s =_{\beta\eta} t \Rightarrow C[s] =_{\beta\eta} C[t]$ for all C.

Usually, the equations for $(\beta), (\eta)$ are directed. We will employ η-expansion, denoted as $\overline{\eta}$. For all contexts C:

(β) $C[(\lambda x.t)\ s]$	$\to C[t[s/x]]$	
(η) $C[\lambda y.(t\ y)]$	$\to C[t]$	If $y \notin FV(t)$.
$(\overline{\eta})$ $C[t]$	$\to C[\lambda y.(t\ y)]$	if t is not an abstraction, $type(t)$ is not an elementary type, and t in $C[t]$ is a maximal application. The variable y must be a fresh variable of appropriate type.

If a term cannot be further reduced by $\overline{\eta}$, then it is in η-long form. If it cannot be further reduced by $\beta\overline{\eta}$, then it is in $\beta\overline{\eta}$-normal form, also called η-long β-normal form. It is well-known that the reduction relation defined by these two reductions is strongly terminating and Church-Rosser [Bar84,Hue76,Wol93]. Hence for every term t, there is a $\beta\overline{\eta}$-normal form $t{\downarrow_{\beta\overline{\eta}}}$, which is unique up to $=_\alpha$. It is also well-known that $\beta\eta$ reduction is also strongly terminating and Church-Rosser.

Proposition 2.4. *The following equivalence holds:*

$$s =_{\alpha\beta\eta} t \Leftrightarrow s{\downarrow_{\beta\overline{\eta}}} =_\alpha t{\downarrow_{\beta\overline{\eta}}} \Leftrightarrow s{\downarrow_{\beta\eta}} =_\alpha t{\downarrow_{\beta\eta}}$$

Lemma 2.5. *A term t is in $\beta\overline{\eta}$-normal form, iff the following holds:*

- *it is in β-normal form, and*
- *every proper subterm s of t, such that s is not an abstraction and s has a non-elementary type is embedded in a superterm of the form $(s\ s')$.*

Example 2.6. The $\beta\overline{\eta}$-normal form of the term $\lambda x^{\iota\to\iota}.x$ is the term $\lambda x^{\iota\to\iota}.(\lambda y^{\iota}.(x\ y))$. The $\beta\overline{\eta}$-normal form of the function symbol $f^{\iota\to\iota\to\iota}$ is the term $\lambda x^{\iota}, y^{\iota}.(f\ x\ y)$

Proposition 2.7. *Let s, t, be terms of equal type, and f be a function symbol. Then*

- $f\ s =_{\beta\eta} f\ t \Leftrightarrow s =_{\beta\eta} t$

- *If type(s) = type$(t) = \alpha_1 \to \ldots$ is not elementary, and f a fresh function symbol of type α_1, then $s =_{\beta\eta} t \Leftrightarrow s\ f =_{\beta\eta} t\ f$.*

For terms in η-long form iterated β-reduction is sufficient to find the $\beta\overline{\eta}$-normal form. In particular, $\beta\eta$-equality of two terms in η-long form can be decided by β-reducing them to normal forms, and then comparing them using α-equality.

Instead of $\lambda x_1.(\lambda x_2.\ldots(\lambda x_n.t)\ldots))$ we write $\lambda x_1,\ldots,x_n.t$, or even $\lambda\overrightarrow{x}.t$. If this is a subterm of another term, we assume that the direct superterm is not an abstraction, and that t is a term of elementary type.
Under these assumptions, every subterm of a term t has one of the forms

- x, where x is a variable
- f, where f is a function symbol
- $(t_0\ t_1\ldots t_n)$ where t_0 is a function symbol, a variable or an abstraction, and $ar(t_0) \geq n \geq 1$, and if t is η-long, then $n = ar(t_0)$.
- $\lambda\overrightarrow{x}.s$

In a term $\lambda x_1,\ldots,x_n.(h\ t_1\ldots\ t_n)$, we say h is the *head* of t. The only possibilities are that h is a constant, a free variable, or some variable x_i.
A term t is in *lifted form*, if it is of the form

$$t = \lambda\overrightarrow{y}.h\ (\lambda\overrightarrow{z_1}.t_1\ \overrightarrow{y}\,\overrightarrow{z_1})\ldots(\lambda\overrightarrow{z_k}.t_k\ \overrightarrow{y}\,\overrightarrow{z_k})$$

and if the following holds: t is closed, for all i: t_i are in lifted form, h is either a function symbol or a variable from $\overrightarrow{y}$, and $ar(t) \leq |\overrightarrow{y}|$ [1].
If t is in addition η-long, then $ar(t) = |\overrightarrow{y}|$, and $ar(h) = k$.
Note that every term has a η-long lifted form.

We will use *positions* which are words of nonnegative integers as follows: $t|_\varepsilon = t$, $(t_0\ t_1 \dots t_n)|_{i.p} = t_i|_p$, and $(\lambda \overrightarrow{x}.t)|_{0.p} = t|_p$.

Definition 2.8. *We define the* irrelevant positions *p of constants in a term s as positions satisfying the following condition: If s' is the term constructed from s by plugging in a fresh constant at position p, then s and s' have the same $\beta\overline{\eta}$-normal form.*

2.1 Higher-Order-Matching Problems

Definition 2.9. *A* higher-order matching problem (HOMP) *Γ is a set of equations $s_i \doteq b_i$, $i = 1, \dots, n$, where s_i, b_i are of elementary type, and all terms b_i are closed. We also assume that for all i the terms s_i, b_i are in η-long β-normal form. A* unifier *is a (capture-free) instantiation for the free variables in s_i, such that all the equations are solved wrt. $\beta\eta$-equality.*

Definition 2.10. *Let N be a positive integer and Γ be a HOMP.*
Then the pair $(\Gamma, N)_{ab}$ is called an arity-bounded HOMP (AB-HOMP). *A* unifier *σ of $(\Gamma, N)_{ab}$ is a substitution that unifies Γ, and satisfies the following syntactic constraints for every variable $x \in FV(\Gamma)$:*

– *$\sigma(x)$ is in η-long lifted form,*
– *The arity of every subterm of $\sigma(x)$ is at most N.*

The pair $(\Gamma, N)_{fvb}$ is called a fv-bounded HOMP (FVB-HOMP). *A* unifier *σ of $(\Gamma, N)_{fvb}$ is a substitution that unifies Γ, and satisfies the following syntactic constraints for every variable $x \in FV(\Gamma)$:*

– *$\sigma(x)$ is in η-long β-normal form,*
– *Every subterm of $\sigma(x)$ has at most N different free variables*[2].

The two latter problems are related, but there appears to be no obvious translation.

3 Behavioral and Contextual Equivalences

In the following we give a simplified treatment of a slight variation of Padovani's development of equivalence classes for higher-order matching, which avoids the problem of free variables in representative terms, and which allows easier proofs. In particular, we get as a corollary that there is a computable upper bound for the number of equivalence classes of terms of a given type.

[1] We denote the number of variables in $\overrightarrow{v_i}$ as $|\overrightarrow{v_i}|$.
[2] We do not mean number of occurrences.

Definition 3.1. *Let Γ be a higher-order matching problem wrt. the signature Σ_0. $S_0(\Gamma)$ be the set of right-hand sides of Γ. Note that we assume that all bound variables of terms in $S_0(\Gamma)$ are different.*
Let $S_1(\Gamma) := \{s \mid s$ is a subterm of some $s_0 \in S_0(\Gamma)$ and s is of elementary type$\}$.
Let Θ be a set of constants $\{c_x, d_x \mid x \in FV(S_1(\Gamma)), type(c_x) = type(d_x) = type(x), c_x, d_x \in \Sigma_1\}$, such that $x \neq y$ implies $c_x \neq c_y$ and $d_x \neq d_y$, for all $x, y : c_x \neq d_y$, and $\Theta \subseteq \Sigma_1 \setminus \Sigma_0$.
We define the set $S(\Gamma)$ of instances of right hand sides *of Γ as follows:*

$$S(\Gamma) := \{s_1\theta \mid s_1 \in S_1, \theta : FV(S_1) \to \Theta \text{ is a substitution}\}$$

In the following we will omit the argument Γ, if this does not lead to ambiguity.

Example 3.2. Let $S_0 = \{\lambda x.(x\ (\lambda y.f\ y\ y))\}$, where $type(x) = (\iota_1 \to \iota_2) \to \iota_3, type(f) = \iota_1 \to \iota_1 \to \iota_2, type(y) = \iota_1$. Let $S_1 = \{x\ (\lambda y.f\ y\ y), f\ y\ y, y\}$ and $\Theta = \{c_x, c_y, d_x, d_y\} \subseteq \Sigma_1 \setminus \Sigma_0$, where $type(c_x) = type(d_x) = type(x), type(c_y) = type(d_y) = type(y) = \iota_1$. Then $S = \{c_y, d_y, c_x\ (\lambda y.f\ y\ y), d_x\ (\lambda y.f\ y\ y), f\ c_y\ c_y, f\ d_y\ d_y, \}$,

In the following we assume that Γ, Σ_0, S are fixed.

Definition 3.3. *Let Γ be a HOMP, and S be the set of instances of right-hand sides of Γ. Let s, t be two closed terms of equal type.*
The terms s, t are contextually equivalent for S, *notation $s \equiv_{c,S} t$ iff*

> *for all terms $u \in \mathrm{Term}_1$[3] of elementary type, such that $FV(u) = \{x\}$ and for all terms $r \in S$: $u[s/x]\downarrow_{\beta\overline{\eta}} =_\alpha r \Leftrightarrow u[t/x]\downarrow_{\beta\overline{\eta}} =_\alpha r$.*

Definition 3.4. *Let Γ be a HOMP, and S be the set of instances of right-hand sides of Γ. Let s, t be two closed terms of equal type.*
The terms s, t are behaviorally equivalent for S, *notation $s \equiv_{b,S} t$ iff*

> *for all closed terms $t_1, \dots, t_n \in \mathrm{Term}_1$ (of appropriate type where $n = \mathrm{ar}(s)$) and for all $r \in S$: $(s\ t_1 \dots t_n)\downarrow_{\beta\overline{\eta}} =_\alpha r \Leftrightarrow (t\ t_1 \dots t_n)\downarrow_{\beta\overline{\eta}} =_\alpha r$.*

4 The Context Lemma

In the following we show that behavioral equivalence is equivalent to contextual equivalence. The proof is inspired from Padovani [Pad95,Pad00], but we close a gap in his proof.

Lemma 4.1. *Let Γ be a HOMP and S be the set of instances of right hand sides of Γ (see Definition 3.1). Let s, t be two closed terms. Then*

> *$s \equiv_{b,S} t$ is equivalent to $s \equiv_{c,S} t$.*

[3] terms wrt. the signature $\Sigma_1 \supseteq \Sigma_0$.

Proof. The direction $s \equiv_{c,S} t \Rightarrow s \equiv_{b,S} t$ follows from the definition, since $(x\ t_1\ \dots\ t_n)$ is a term u that is permitted in the definitions of contextual equivalence.
The main part is to show that $s \equiv_{b,S} t \Rightarrow s \equiv_{c,S} t$.
Let u be an arbitrary term of elementary type with $FV(u) = \{x\}$, let $u[s/x]\downarrow_{\beta\overline{\eta}} =_\alpha r \in S$. We have to show that $u[t/x]\downarrow_{\beta\overline{\eta}} =_\alpha r$.
It is no restriction to assume that s, t, u are in $\beta\overline{\eta}$-normal form.
The induction is

1. on the maximal number of β-reductions of $u[s/x]$ to $\beta\overline{\eta}$-normal form.
2. on the size of u

The term u cannot be of the form $\lambda\overrightarrow{y}.u'$, since its type is elementary.
Case: u has a function symbol $f \in \Sigma_1$ as head-symbol.
Then u is of the form $f\ \lambda\overrightarrow{y_1}.u_1\ \dots\ \lambda\overrightarrow{y_k}.u_k$.

If for some i, we have $x \notin FV(\lambda\overrightarrow{y_i}.u_i)$, then the corresponding subterms $u_i[s/x] =_\alpha u_i =_\alpha u_i[t/x]$.
Let i be an index, such that $x \in FV(\lambda\overrightarrow{y_i}.u_i)$. We fix this index for the rest of the arguments in this part of the proof, and we use induction as follows:
Let $u[s/x] = f\ \lambda\overrightarrow{y_1}.u_1[s/x] \dots \lambda\overrightarrow{y_n}.u_n[s/x] =_\alpha r \in S$, and $r = f\ \lambda\overrightarrow{y_1}.r_1 \dots \lambda\overrightarrow{y_n}.r_n$. We assume that all $y_{i,j}$ are different. The definition of S implies that $r_i\theta \in S$ for all substitutions $\theta : FV(r_i) \to \Theta$.
The goal is to show by induction that $(\lambda\overrightarrow{y_i}.u_i[s/x])\downarrow_{\beta\overline{\eta}} =_\alpha (\lambda\overrightarrow{y_i}.u_i[t/x])\downarrow_{\beta\overline{\eta}}$.

The problem which had to be faced is that we cannot simply replace the free variables in s, r_i, u_i by constants in Θ and then use induction, since the constants may already appear in r_i, u_i, which prevents lifting the equality to the abstraction. We use the fact that there are more constants in Θ than constant occurrences in $r\theta$, and then argue that most of the constant occurrences in s, u_i are irrelevant.

Let $m = ar(\lambda\overrightarrow{y_i}.u_i)$. Now assume that θ is any substitution $\theta : y_{i,j} \mapsto c_j$ for $j = 1, \dots, m$, where $c_j \in \Theta$. Then by induction, since at least the size of u is smaller, we have $((u_i[s/x])\theta)\downarrow_{\beta\overline{\eta}} =_\alpha r_i\theta =_\alpha ((u_i[t/x])\theta)\downarrow_{\beta\overline{\eta}}$, since $r_i\theta \in S$. We construct terms s', t' from s, t, respectively by replacing the constant positions in s, t that are irrelevant in $((u_i[s/x])\theta)$ and $((u_i[t/x])\theta)$ by arbitrary constants that are in $\Sigma_1 \setminus (\Theta \cup \Sigma_0)$. We do the same for positions in u_i that are irrelevant in $((u_i[s/x])\theta)$ and in $((u_i[t/x])\theta)$. The property of irrelevant positions implies that the equations

$$((u_i[s/x])\theta)\downarrow_{\beta\overline{\eta}} =_\alpha ((u_i'[s'/x])\theta)\downarrow_{\beta\overline{\eta}} =_\alpha ((u_i'[t'/x])\theta)\downarrow_{\beta\overline{\eta}} =_\alpha ((u_i[t/x])\theta)\downarrow_{\beta\overline{\eta}}$$

$=_\alpha r_i\theta$ hold.

Let θ' now be another substitution $\theta' : y_{i,j} \mapsto c_j'$ where c_j' for $j = 1, \dots, m$ are pairwise different constants from Θ, which in addition do not occur in u_i', s' or t'. This is possible according to the construction of Θ, since Θ contains for every free variable in some term in S_0 two appropriate constants c_x, d_x. As argued above, we have $((u_i[s/x])\theta')\downarrow_{\beta\overline{\eta}} =_\alpha ((u_i[t/x])\theta')\downarrow_{\beta\overline{\eta}}$. Since the equations differ only by irrelevant constants, and since $r_i\theta' \in S$, we also have

$$((u'_i[s'/x])\theta')\downarrow_{\beta\overline{\eta}} =_\alpha ((u_i[s/x])\theta')\downarrow_{\beta\overline{\eta}} =_\alpha ((u_i[t/x])\theta')\downarrow_{\beta\overline{\eta}} =_\alpha ((u'_i[t'/x])\theta')\downarrow_{\beta\overline{\eta}}$$
$$=_\alpha r_i\theta'.$$

Since the constants c'_j for $j = 1,\ldots,m$ do not occur in r_i, it is obvious that $\lambda\overrightarrow{y_i}.u_i[s/x]\downarrow_{\beta\overline{\eta}} =_\alpha \lambda\overrightarrow{y_i}.u_i[t/x]\downarrow_{\beta\overline{\eta}}$.
Case: There are no reductions of $u[s/x]$. Then there are also no reductions of $u[t/x]$, since s, t are of the same type and s, t, u are in $\beta\overline{\eta}$-normal form. Hence $u[s/x]$ is in $\beta\overline{\eta}$-normal form.

Due to the first case, it is sufficient to consider the case that the head-symbol of u is a variable. It is not possible that u is of the form $(x\ u_1 \ldots u_k)$ for $k \geq 1$, since then a reduction of $(s\ u_1 \ldots u_k)$ would be possible. If $u = x$, then $s \equiv_{b,S} t$ implies that either $s\downarrow_{\beta\overline{\eta}} \in S$ and $s\downarrow_{\beta\overline{\eta}} =_\alpha t\downarrow_{\beta\overline{\eta}}$, or $s\downarrow_{\beta\overline{\eta}} \notin S$. In either case, the claim of the lemma holds.
Case: There are reductions of $u[s/x]$.

The only case that has to be treated is that u is of the form $(x\ u_1 \ldots u_k)$ with $k \geq 1$. Let $s = \lambda\overrightarrow{y}.w$. Then one reduction step of $u[s/x]$ gives $w[u_1[s/x]/y_1, \ldots u_k[s/x]/y_k]$, which normalizes to $r \in S$. Let $u' = w[u_1/y_1, \ldots u_k/y_k]$. If there are no occurrences of x in u', then the lemma is trivial. Otherwise, induction on the maximal number of reductions shows that $u'[t/x]\downarrow_{\beta\overline{\eta}} =_\alpha (w[u_1[t/x]/y_1, \ldots u_k[t/x]/y_k])\downarrow_{\beta\overline{\eta}} =_\alpha r \in S$. Hence $(s\ u_1[t/x] \ldots u_k[t/x])\downarrow_{\beta\overline{\eta}} =_\alpha r \in S$.

The assumption of the lemma can be used: $s \equiv_{b,S} t$ implies that $(t\ u_1[t/x] \ldots u_k[t/x])\downarrow_{\beta\overline{\eta}} =_\alpha r \in S$. We finally obtain $(t\ u_1[t/x] \ldots u_k[t/x])\downarrow_{\beta\overline{\eta}} =_\alpha (u[t/x])\downarrow_{\beta\overline{\eta}} =_\alpha r$.
We have checked all cases, hence the lemma is proved. □

In the following we write $\equiv_S$ instead of $\equiv_{b,S}$ or $\equiv_{c,S}$.

5 A Computable Upper Bound for the Number of Equivalence Classes

The goal of this section is to argue that for a fixed Γ there is a computable function that gives an upper bound for the number of equivalence classes for a given type.

Proposition 5.1. *Assume given a finite signature Σ_0, a HOMP Γ, and let S be the set of instances of right hand sides as defined in section 3. Let the function Φ on types be defined by $\Phi(\iota) = |S| + 1$, and $\Phi(\tau) := 2|S| \times \Phi(\tau_1) \times \ldots \times \Phi(\tau_n)$ for $\tau = \tau_1 \to \ldots \to \tau_n \to \iota$. Then for every type τ the number of equivalence classes w.r.t. $\equiv_S$ is not greater than $\Phi(\tau)$, the function Φ is computable and $\Phi(\tau) \leq (2|S|)^{(\text{argsize}(\tau))}$ for non-elementary τ.*

Proof. The construction in the proof is similar to Padovani's [Pad95,Pad00]. The induction is on the structure of the type τ. If τ is elementary, the number of equivalence classes is at most $|S| + 1$. The normal forms of the terms may be (modulo $=_\alpha$) in S, or not in S. This is a contribution of $|S| + 1$.

Let $\tau = \tau_1 \to \ldots \to \tau_n \to \iota$. Assume we fix a set of representatives of the equivalence classes of terms of type τ_i for $i = 1, \ldots, n$.

For a term s of type τ, we define

$$\begin{aligned} Chartup(s) := \{(t_1,\dots,t_n,r,op) \mid\ & t_i \text{ is a representative of} \\ & \text{the } \equiv_S\text{-equivalence classes of terms of type } \tau_i, \\ & r \in S, \text{ and } (s\ t_1 \dots t_n)\ op\ r \text{ where } op \in \{\equiv_S, \not\equiv_S\}\} \end{aligned}$$

Here we assume, that for every tuple of representatives and every term r, there is exactly one tuple in the set, either with operator $\equiv_S$, or $\not\equiv_S$.
Given two terms s, t of type τ, we prove that $s \equiv_S t$ is equivalent to $Chartup(s) = Chartup(t)$:
Let $s \equiv_S t$. Then $(s\ t_1 \dots t_n){\downarrow_{\beta\overline{\eta}}} =_\alpha r$ is equivalent to $(t\ t_1 \dots t_n){\downarrow_{\beta\overline{\eta}}} =_\alpha r$. Hence $Chartup(s) = Chartup(t)$.
Now assume that $Chartup(s) = Chartup(t)$. Let $a_1, \dots, a_n$ be any closed terms, such that $(s\ a_1 \dots a_n)$ is of elementary type, and $(s\ a_1 \dots a_n){\downarrow_{\beta\overline{\eta}}} =_\alpha r \in S$. Then the context lemma 4.1 shows that $(s\ t_1 \dots t_n){\downarrow_{\beta\overline{\eta}}} =_\alpha r$ for the representatives $t_i \equiv_S a_i$ for $i = 1, \dots, n$. Since $Chartup(s) = Chartup(t)$, we obtain $(t\ t_1 \dots t_n){\downarrow_{\beta\overline{\eta}}} =_\alpha r$, and again by the context lemma, we get $(t\ a_1 \dots a_n){\downarrow_{\beta\overline{\eta}}} =_\alpha r$. Since this holds for all appropriate a_i, and the proof also holds for the symmetric case, we have proved $s \equiv_S t$.

Now we can estimate an upper bound for the number of equivalence classes by counting the possible tuples. Since the number of representatives of type τ_i is at most $\Phi(\tau_i)$, there are at most $2|S| \times \Phi(\tau_1) \times \dots \times \Phi(\tau_n)$ such possibilities. □

6 An Algorithm for Arity-Bounded and fv-Bounded HOMPs

Definition 6.1. *Let Σ_0 be a signature, and τ be a type.*
The (non-deterministic) standard generation of all terms of type τ in lifted form (η-long or not) uses a tree as data structure: Let a_p denote closed terms wrt. the signature Σ_0, where p is a tree-position, and $a_\varepsilon : \tau$ is the root.
The following rule is used for generation:

$$a_p \to \lambda\overrightarrow{y}.(h\ (\lambda\overrightarrow{z_1}.(a_{p.1}\overrightarrow{y}\overrightarrow{z_1}))\ \dots (\lambda\overrightarrow{z_{k_p}}.(a_{p,k_p}\overrightarrow{y}\overrightarrow{z_{k_p}})))$$

where either $h \in \Sigma_0$ or h is variable in $\overrightarrow{y}$. The inequation $arity(h) \geq k_p$ must hold, whereas for η-normal forms, the equation $arity(h) = k_p$ must hold.

To reach the (η-long) β-normal form, it is necessary to make β-reductions of the terms $a_{p.i}\overrightarrow{y}\overrightarrow{z_i}$ in order to plug in the variables $\overrightarrow{y}, \overrightarrow{z_i}$.

The non-determinism comes from the choice of the symbol h. In the η-long case, the leaf terms are projections to elementary variables or constants, whereas in the other case, the leaf terms are projections to variables or constants. Note that all terms must be simply typed.

It is easy to see that this generates all terms, and that moreover, there is no bound on the arity of types of a_p.
For arity-bounded or fv-bounded HOMPS, the generation can be optimized to:

Definition 6.2. *Let Σ_0 be a signature, and τ be a type, let $N > 0$ be an integer. The (non-deterministic) arity-bounded generation (fv-bounded, respectively) with bound N of terms of type τ in η-long lifted form uses the following rule:*

$$a_p \to \lambda \overrightarrow{y}.(h\ (\lambda \overrightarrow{z_1}.(a_{p.1} \overrightarrow{v_1} \overrightarrow{z_1}))\ \ldots (\lambda \overrightarrow{z_{k_p}}.(a_{p,k_p} \overrightarrow{v_{k_p}} \overrightarrow{z_{k_p}})))$$

where either $h \in \Sigma_0$ or h is variable in $\overrightarrow{y}$, and where $\overrightarrow{v_i}$ is a subset of the variables $\overrightarrow{y}$, such that:
for fv-bounded generation $|\overrightarrow{v_i}| \leq N$
for arity-bounded generation: $|\overrightarrow{v_i}| + |\overrightarrow{z_i}| \leq N$ and $ar(h) \leq N$.

Now there is an additional source of nondeterminism: the selection of variables $\overrightarrow{v_i}$ among $\overrightarrow{y}$. Note that for fv-bounded generation it is not possible to remove components from $\overrightarrow{z_i}$, since the expression must be simply typed.

We define the algorithm for deciding the solvability of an arity-bounded (or fv-bounded, respectively) HOMP.

Definition 6.3. *The following non-deterministic algorithm generates instantiations σ for the arity-bounded (or fv-bounded, respectively) higher order matching problem. Let (Γ, N) be the input.*
Compute the set T of all types and subtypes of variables and function symbols in Γ.
Let $\overline{N} := \sum_{n \leq N, \tau_i \in T, \iota \in T} \Phi(\tau_1 \to \ldots \to \tau_n \to \iota)$ for the arity-bounded, and $\overline{N} := \sum_{n \leq N+N_0, \tau_i \in T, \iota \in T} \Phi(\tau_1 \to \ldots \to \tau_n \to \iota)$, for the fv-bounded, where N_0 is the maximal arity of the types in T.

1. *For every variable $x \in FV(\Gamma)$ compute a term $\sigma(x)$ using a tree as follows: Let a_ε be the root of the tree.*
   - *Iterate the following as long as there are leaves without attached term:*
     - *If the length of p is greater than $\overline{N}$, then fail*
     - *Let p be a position where a_p is the label at the leaf.*
     - *Extend the tree by replacing a_p according to the rule given in definition 6.2.*
   - *For a FVB-HOMP, reduce $\sigma(x)$ to β-normal form and then check that every subterm of has at most N free different variables; otherwise fail.*
2. *Now check whether σ is a matcher of Γ. If yes, then a solution was found, otherwise fail.*

The algorithm returns "solvable", if at least one non-deterministic run finds a matcher, otherwise "unsolvable".

Theorem 6.4. *The algorithm in definition 6.3 decides solvability of arity-bounded higher-order matching problems.*

Proof. The algorithm terminates, since for every variable, the depth of a path in the generating tree is bounded by a computable number. Using König's lemma,

it is possible to deterministically compute all possibilities in finite time.
It is obvious that if the algorithm answers “solvable”, then the AB-HOMP Γ is unifiable.
We show completeness:
Let $(\Gamma, N)_{ab}$ be an AB-HOMP, and σ be a minimal matcher in η-long lifted form. Let $x \in FV(\Gamma)$. It is possible to control the generation such that $\sigma(x)$ will be the result. The generated instance for x satisfies the condition that the arity of all a_p is not greater than N.

By induction we can show that the types of the arguments of a_p are in the set T. Proposition 5.1 shows that there is a computable upper bound on the number of different equivalence classes wrt. $\equiv_S$ for a given type. Thus, in every sequence of terms of length at least $\overline{N}$, where the terms have a type of arity at most N, there are at least two terms that are equivalent wrt. $\equiv_S$.

Since a_ε was chosen of minimal size, it is not possible that there are two $\equiv_S$-equivalent $a_p, a_{p'}$, such that a_p is an ancestor of $a_{p'}$, since in this case we can strictly decrease the size by replacing a_p by $a_{p'}$, which is correct due to the context lemma 4.1. Hence, whenever there is a path in the term tree that is longer than $\overline{N}$, it is correct to stop with failure.

We have shown that the algorithm in definition 6.3 decides solvability of arity-bounded higher-order matching problem. $\square$

Corollary 6.5. *The arity-bounded HOMP is decidable.*

Theorem 6.6. *The algorithm in definition 6.3 decides solvability of fv-bounded HOMPs.*

Proof. The proof is very similar to the last proof, hence we only point out the differences.

Let $(\Gamma, N)_{fvb}$ be an fv-bounded HOMP. Let $x \in FV(\Gamma)$. Then we use the generation of terms as defined in 6.2 for fv-bounded problems and for η- normal forms.

The term for x is generated such that the arity of all a_p is not greater than $N + N_0$ where N_0 is the maximum arity of all the types and subtypes that occur in the types of the free variables in Γ and the function symbols in Σ_0.

Similar arguments as in the last proof show that the maximal length of a path in the generating tree does not exceed $\overline{N}$, as defined for fv-bounded HOMPs.

For the rest we can use the same arguments as in the proof of Theorem 6.4. $\square$

Corollary 6.7. *Solvability of fv-bounded HOMPs is decidable.*

Increasing the bounds N and calling the decision algorithm for arity-bounded (fv-bounded, respectively) higher-order matching problem, it is possible to enumerate all solutions of the HOMP Γ.

Note that the computability of an upper bound on the number of equivalence classes is required, since it is not known whether $\equiv_S$ is decidable. Using Padovanis result, $\equiv_S$ is decidable for orders ≤ 5.

Corollary 6.8. *Assume given a HOMP Γ, and a computable function γ that computes for every higher-order matching problem Γ a bound $\gamma(\Gamma)$. Then the arity-bounded as well as the fv-bounded higher-order matching problem $(\Gamma, \gamma(\Gamma))$ can be solved by the given algorithm.*

Remark 6.9. Every k-duplicating [DW02] higher-order matching problem can be encoded as an arity-bounded higher-order matching problem, where N can be computed from k and Σ_0, by checking the finitely many potential matchers, as proved in [DW02]. Then we can apply Corollary 6.8.

This shows that the class of arity-bounded higher-order matching is in a sense a generalization of k-duplicating higher-order matching.

We do not require unifiers to be in η-normal form, since then the theorems also hold, though the generation process and the proof differ slightly.

We call the corresponding problems β-arity-bounded higher-order matching and β-fv-bounded higher-order matching.

Theorem 6.10. *β-fv-bounded higher-order matching and β-arity-bounded higher-order matching are decidable*

Proof. The only differences to the proof of Theorems 6.4 and 6.6 is the generation of expressions in Definition 6.2. The same arguments and loop detection can be used as in the proof of Theorem 6.4. $\square$

Remark 6.11. In the fourth order case of higher-order matching, following the arguments of Padovani [Pad00], it is not hard to see that in solutions of fourth-order matching problems (he considers in fact so-called dual interpolation problems), it is sufficient to only consider matchers, such that every elementary subterm has a bounded number of free variables, where the argument is different for elementary variables and non-elementary variables. The bound can be computed from the right hand sides.
Decidability of fourth-order matching can then be proved using the decidability of fv-bounded higher-order matching[4]. We use a cutting method that uses only local properties of the matcher, namely a bound on the arity or number of free variables, whereas Padovani requires global information from the interpolation problem about the possible argument lists.

Unfortunately, for orders 5 and higher, the same arguments cannot be used to bound the number of free variables in matchers.

7 Remarks on the Complexity

We use the ideas of the proof of a lower non-elementary worst-case complexity bound in [Wie99] which in turn is based on [Sta79].

The idea is to show that the arity-bounded (fv-bounded) higher-order matching can simulate the solution of the question whether for closed s, t it is $s =_{\alpha\beta\eta} t$, where t is assumed to be in $\beta\overline{\eta}$-normal form.

[4] Here I have to thank a referee for the hint.

The higher-order matching problem Γ is constructed by hiding redexes, i.e., if s contains a redex $C[s_1\ s_2]$, then s is replaced by $C[x\ s_1\ s_2]$ for a new variable x, and the match equation $x = \lambda y_1, y_2.(y_1\ y_2)$ of appropriate type is added . This can be done for all β-redexes, and such that there are no more β-redexes in the right hand sides of the constructed higher-order matching problem Γ. We know the potential solution by construction. The only question is whether it is a solution or not. We can ignore η-redexes, since the comparison of β-normal forms for η-equality is polynomial. Obviously, the arity-bound can be chosen to be the maximal arity of subexpression of s. The bound for the free variables can be chosen as 0. Then (Γ, N) has a solution iff s β-reduces to t', and t' is $\alpha\eta$-equal to t. Hence:

Proposition 7.1. *The decision problems arity-bounded higher-order matching and fv-bounded higher-order matching are not elementary recursive.*

Proposition 7.2. *The number of most general unifiers may be infinite for an arity-bounded higher-order matching problem.*

Proof. The example $\{x\ (\lambda y.y)\ a \doteq a, x\ (\lambda y.b)\ a \doteq b\}$ has an infinite number of most general unifiers, under both restrictions: $\sigma_i := x \mapsto \lambda x_1, x_2. \underbrace{x_1(x_1 \ldots (x_1}_{i} x_2))$. has only subexpressions of arity at most 1, and the number of different free variables is at most 2 in subexpression. □

This shows that the class of arity-bounded higher-order matching problems is a generalization of k-duplicating higher-order matching [DW02].

8 Conclusion

We have shown that under the condition that there is a bound on the arity of subterms or the number of free variables in subexpressions of matchers in η-long lifted form (or β-normal form, respectively), higher-order matching becomes decidable. We have argued, that this is a proper generalization of the class of k-duplicating higher-order matching (see Remark 6.9 and Proposition 7.2). Even fourth-order matching can be seen as a subcase of fv-bounded higher-order matching (see Remark 6.11). In the general case the loop detection does not work, since in the generation process the arity of the functions a_p is growing. Finding a computable upper bound to one of these numbers or to invent an improved loop-detection would solve the higher-order matching problem by showing decidability.

References

Bar84. Henk P. Barendregt. *The Lambda Calculus. Its Syntax and Semantics.* North-Holland, Amsterdam, New York, 1984.

CJ97. Hubert Comon and Yan Jurski. Higher-order matching and tree automata. In *Proc. of CSL 97*, volume 1414 of *LNCS*, pages 157–176, 1997.

dG00. Philippe de Groote. Linear higher-order matching is NP-complete. In *Proceedings of the 11th Int. Conf. on Rewriting Techniques and Applications*, volume 1833 of *LNCS*, pages 127–140. Springer-Verlag, 2000.

Dow92. Gilles Dowek. Third order matching is decidable. In *Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science*, pages 2–10, 1992.

Dow01. Gilles Dowek. Higher-order unification and matching. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 16, pages 1009–1062. North-Holland, 2001.

DW02. Dan Dougherty and ToMasz Wierzbicki. A decidable variant of higher order matching. In *Proc. RTA'02*, volume 2378 of *LNCS*, pages 340–351. Springer, 2002.

Hue75. Gérard Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.

Hue76. Gérard Huet. *Résolution d'équations dans des langages d'ordre 1,2,...ω.* Thèse de doctorat d'etat, Université Paris VII, 1976. In French.

Loa97. R. Loader. An algorithm for the minimal model. Technical report, 1997. 4 pages, available on www.

Loa01. Ralph Loader. Higher-order β matching is undecidable, 2001. draft.

Mil91. Dale Miller. A logic programming language with lambda-abstraction, function variables and simple unification. *J. of Logic and Computation*, 1(4):497–536, 1991.

Nip91. Tobias Nipkow. Higher-order critical pairs. In *Proc. 6th IEEE Symp. LICS*, pages 342–349, 1991.

NP98. Tobias Nipkow and Christian Prehofer. Higher-order rewriting and equational reasoning. In W. Bibel and P. Schmitt, editors, *Automated Deduction — A Basis for Applications. Volume I: Foundations*, volume 8 of *Applied Logic Series*, pages 399–430. Kluwer, 1998.

Pad95. Vincent Padovani. On equivalence classes of interpolation equations. In M. Dezani-Ciancagliani and G. Plotkin, editors, *Typed lambda-Calculi and Applications*, volume 902 of *LNCS*, pages 335–349. Springer-Verlag, 1995.

Pad96. Vincent Padovani. Decidability of all minimal models. In M. Coppo and S. Berardi, editors, *Proc. Types for Proofs and Programs*, volume 1158 of *Lecture Notes in Computer Science*, pages 201–215, 1996.

Pad00. Vincent Padovani. Decidability of fourth-order matching. *Mathematical Structures in Computer Science*, 10(3):361–372, 2000.

Pfe01. Frank Pfenning. Logical frameworks. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 17, pages 1063–1147. North-Holland, 2001.

Sta79. Richard Statman. The typed λ-calculus is not elementary recursive. *Theoretical Computer Science*, 9:73–81, 1979.

Wie99. ToMasz Wierzbicki. Complexity of the higher-order matching. In *Proc. 16^{th} CADE*, number 1632 in LNCS, pages 82–96. Springer-Verlag, 1999.

Wol93. David A. Wolfram. *The clausal theories of types.* Number 21 in Cambridge tracts in theoretical computer science. Cambridge University Press, 1993.