Linear higher-order matching is NP-complete

Philippe de Groote

LORIA UMR n^o 7503 – INRIA
Campus Scientifique, B.P. 239
54506 Vandœuvre lès Nancy Cedex – France
e-mail: degroote@loria.fr

Abstract. We consider the problem of higher-order matching restricted to the set of linear λ-terms (i.e., λ-terms where each abstraction $\lambda x.\, M$ is such that there is exactly one free occurrence of x in M). We prove that this problem is decidable by showing that it belongs to NP. Then we prove that this problem is in fact NP-complete. Finally, we discuss some heuristics for a practical algorithm.

1 Introduction

Higher-order unification, which is the problem of solving a syntactic equation (modulo β or $\beta\eta$) between two simply typed λ-terms is known to be undecidable [8], even in the second-order case [7]. On the other hand, if one of the two λ-terms does not contain any unknown, the problem (which is called, in this case, higher-order matching) becomes simpler. Indeed, second-order [10], third-order [5], and fourth-order [17] matching have been proved to be decidable (See also [21] for a survey, including lower and upper bounds on the complexity). In the general case, however, the decidability of higher-order matching is still open.

Since higher-order unification is known to be undecidable, it is natural to investigate whether one can recover decidability for some restricted form of it. For instance, Miller's higher-order paterns form a class of λ-terms for which unification is decidable [14]. Another restricted form of higher-order unification (in fact, in this case, second-order unification) is context unification [3, 4], which may be seen as a generalisation of word unification. While the decidability of context unification is open in general, some subcases of it have been shown to be decidable [3, 4, 11, 12, 19].

In this paper, we study higher-order matching for a quite restricted class of λ-terms, namely, the λ-terms that correspond to the proofs of the implicative fragment of multiplicative linear logic [6]. These λ-terms are *linear* in the sense that each abstraction $\lambda x.\, M$ is such that there is exactly one free occurrence of x in M. Our main result is the decidability of *linear higher-order matching*. Whether linear higher-order unification is decidable is an open problem related, in the second-order case, to context unification [11].

Linear higher-order unification (in the sense of this paper) has been investigated by Cervesato and Pfenning [2], and by Levy [11]. Cervesato and Pfenning

consider the problem of higher-order unification for a λ-calculus whose type system corresponds to full intuitionistic linear logic. They are not interested in decidability results—they know that the problem they study is undecidable because the simply-typed λ-calculus may be embedded in the calculus they consider—but in giving a semi-decision procedure in the spirit of huet's pre-unification algorithm [9]. Levy, on the other hand, is interested in decidability results but only for the second-order case, whose decidability implies the decidability of context unification.

Both in the case of Levy and in the case of Cervesato and Pfenning, there was no reason for considering matching rather than unification. In the first case, the matching variant of the problem is subsumed by second-order matching, which is known to be decidable. In the second case, the matching variant of the problem subsumes full higher-order matching, which is known to be a hard open problem.

Our own motivations in studying linear higher-order matching come from the use of categorial grammars in computational linguistics. Natural language parsing using modern categorial grammars [15, 16] amounts to automated deduction in logics akin to the multiplicative fragment of linear logic. Consequently, the syntactic structures that result from categorial parsing may be seen, through the Curry-Howard correspondance, as linear λ-terms. As a consequence, higher-order unification and matching restricted to the set of linear λ-terms have applications in this categorial setting. In [13], for instance, Morrill and Merenciano show how to use linear higher-order matching to generate a syntactic form (i.e., a sentence in natural language) from a given logical form.

The paper is organised as follows. In the next section, we review several basic definitions and define precisely what is a linear higher-order matching problem. In Section 3, we prove that linear higher-order matching is decidable while, in Section 4, we prove its NP-completeness. Finally, in Section 5, we specify a more practical algorithm and we discuss some implementation issues.

2 Basic definitions

Definition 1. *Let $\mathscr{A}$ be a finite set, the elements of which are called* atomic types. *The set $\mathscr{F}$ of* linear functional types *is defined according to the following grammar:*

$$\mathscr{F} \quad ::= \quad \mathscr{A} \quad | \quad (\mathscr{F} \multimap \mathscr{F}).$$

■

We let the lowercase Greek letters $(\alpha, \beta, \gamma, \ldots)$ range over $\mathscr{F}$.

Definition 2. *Let $(\Sigma_\alpha)_{\alpha\in\mathscr{F}}$ be a family of pairwise disjoint finite sets indexed by $\mathscr{F}$, whose almost every member is empty. Let $(\mathscr{X}_\alpha)_{\alpha\in\mathscr{F}}$ and $(\mathscr{Y}_\alpha)_{\alpha\in\mathscr{F}}$ be two families of pairwise disjoint countably infinite sets indexed by $\mathscr{F}$, such that $(\bigcup_{\alpha\in\mathscr{F}} \mathscr{X}_\alpha) \cap (\bigcup_{\alpha\in\mathscr{F}} \mathscr{Y}_\alpha) = \varnothing$. The set $\mathscr{T}$ of raw λ-terms is defined according to the following grammar:*

$$\mathscr{T} \quad ::= \quad \Sigma \quad | \quad \mathscr{X} \quad | \quad \mathscr{Y} \quad | \quad \lambda\mathscr{X}.\,\mathscr{T} \quad | \quad (\mathscr{T}\,\mathscr{T}),$$

where $\Sigma = \bigcup_{\alpha\in\mathscr{F}} \Sigma_\alpha$, $\mathscr{X} = \bigcup_{\alpha\in\mathscr{F}} \mathscr{X}_\alpha$, *and* $\mathscr{Y} = \bigcup_{\alpha\in\mathscr{F}} \mathscr{Y}_\alpha$. ∎

In the above definition, the elements of Σ are called the constants, the elements of $\mathscr{X}$ are called the λ-*variables*, and the elements of $\mathscr{Y}$ are called the *meta-variables* or the *unknowns*. We let the lowercase roman letters $(\mathrm{a}, \mathrm{b}, \mathrm{c}, \ldots)$ range over the constants, the lowercase italic letters $(x, y, z, \ldots)$ range over the λ-variables, the uppercase bold letters $(\mathbf{X}, \mathbf{Y}, \mathbf{Z}, \ldots)$ range over the unknowns, and the uppercase italic letters $(M, N, O, \ldots)$ range over the λ-terms. The notions of free and bound occurrences of a λ-variable are defined as usual, and we write $\mathrm{FV}(M)$ for the set of λ-variables that occur free in a λ-term M. Finally, a λ-term that does not contain any unknown is called a *pure* λ-*term*.

If $\mathrm{P} = \{i_0, i_1, \ldots, i_n\}$ is a (linearly ordered) set of indices, we write $\lambda\overline{x}_\mathrm{P}.\, M$ for the λ-term $\lambda x_{i_0}.\, \lambda x_{i_1}.\, \ldots \lambda x_{i_n}.\, M$. Similarly, we write $M\,(\overline{N}_\mathrm{P})$ or $M\,(N_i)_{i\in\mathrm{P}}$ for $(\ldots((M\,N_{i_0})\,N_{i_1})\,\ldots N_{i_n})$. As in set theory, we let $n = \{0, 1, \ldots, n-1\}$. These notations will be extensively used in Section 5.

We then define the notion of typed linear λ-term.

Definition 3. *The family* $(\mathscr{T}_\alpha)_{\alpha\in\mathscr{F}}$ *of sets of typed linear* λ*-terms is inductively defined as follows:*

1. *if* $\mathrm{a} \in \Sigma_\alpha$ *then* $\mathrm{a} \in \mathscr{T}_\alpha$*;*
2. *if* $\mathbf{X} \in \mathscr{Y}_\alpha$ *then* $\mathbf{X} \in \mathscr{T}_\alpha$*;*
3. *if* $x \in \mathscr{X}_\alpha$ *then* $x \in \mathscr{T}_\alpha$*;*
4. *if* $x \in \mathscr{X}_\alpha$, $M \in \mathscr{T}_\beta$, *and* $x \in \mathrm{FV}(M)$, *then* $\lambda x.\, M \in \mathscr{T}_{(\alpha\multimap\beta)}$*;*
5. *if* $M \in \mathscr{T}_{(\alpha\multimap\beta)}$, $N \in \mathscr{T}_\alpha$, *and* $\mathrm{FV}(M) \cap \mathrm{FV}(N) = \varnothing$, *then* $(M\,N) \in \mathscr{T}_\beta$. ∎

Clauses 4 and 5 imply that any typed linear λ-term $\lambda x.\, M$ is such that there is exactly one free occurrence of x in M. Remark, on the other hand, that constants and unknowns may occur several times in the same linear λ-term.

From now on, we define $\mathscr{T}$ to be $\bigcup_{\alpha\in\mathscr{F}} \mathscr{T}_\alpha$ (which is a proper subset of the set of raw λ-terms). It is easy to prove that the sets $(\mathscr{T}_\alpha)_{\alpha\in\mathscr{F}}$ are pairwise disjoint. Consequently, we may define the type of a typed linear λ-term M to be the unique linear type α such that $M \in \mathscr{T}_\alpha$.

We take for granted the usual notions of α-conversion, η-expansion, β-redex, one step β-reduction $(\to_\beta)$, n step β-reduction $(\overset{n}{\to}_\beta)$, many step β-reduction $(\twoheadrightarrow_\beta)$, and β-conversion $(=_\beta)$. We use Δ (possibly with a subscript) to range over β-redexes, and we write $\Delta \subset M$ to say that Δ is a β-redex occurring in a λ-term M.

We let $M[x{:=}N]$ denote the usual capture-avoiding substitution of a λ-variable by a λ-term. Similarly, $M[\mathbf{X}{:=}N]$ denotes the capture-avoiding substitution of an unknown by a λ-term. Note that any β-redex $(\lambda x.\, M)\,N$ occurring in a linear λ-term is such that $\mathrm{FV}(\lambda x.\, M) \cap \mathrm{FV}(N) = \varnothing$. Moreover we may suppose, by α-conversion, that $x \notin \mathrm{FV}(N)$. Consequently, when writing $M[x{:=}N]$ (or $M[\mathbf{X}{:=}N]$) we always assume that $\mathrm{FV}(M) \cap \mathrm{FV}(N) = \varnothing$. Finally we abbreviate $M[x_0{:=}N_0] \cdots [x_{n-1}{:=}N_{n-1}]$ as $M[x_i{:=}N_i]_{i\in n}$.

In Section 5, we will also consider a more semantic version of substitution, i.e., a function $\sigma : \mathscr{Y} \to \mathscr{T}$ that is the identity almost everywhere. The finite subset

of $\mathscr{Y}$ where $\sigma(\mathbf{X}) \neq \mathbf{X}$ is called the domain of the substitution (in notation, $dom(\sigma)$). It is clear that such a substitution σ determines a unique syntactic substitution $(\mathbf{X} := \sigma(\mathbf{X}))_{\mathbf{X} \in dom(\sigma)}$, and conversely.

We now give a precise definition of the matching problem with which we are concerned.

Definition 4. *A* linear higher-order matching problem modulo β *(respectively,* modulo $\beta\eta$*) is a pair of typed linear* λ*-terms* $\langle M, N\rangle$ *of the same type such that* N *is pure (i.e., does not contain any unknown). Such a problem admits a solution if and only if there exists a substitution* $(\mathbf{X}_i{:=}O_i)_{i\in n}$ *such that* $M[\mathbf{X}_i{:=}O_i]_{i\in n} =_\beta N$ *(respectively,* $M[\mathbf{X}_i{:=}O_i]_{i\in n} =_{\beta\eta} N$*).* ■

In the sequel of this paper, a pair of λ-terms $\langle M, N\rangle$ obeying the conditions of the above definition will also be called a *syntactic equation*. Moreover, we will assume that the right-hand side of such an equation (namely, N) is a pure *closed* λ-term. There is no loss of generality in this assumption because, for $x \in \mathrm{FV}(N)$, $\langle M, N\rangle$ admits a solution if and only if $\langle \lambda x.\, M, \lambda x.\, N\rangle$ admits a solution.

3 Decidability

We first define the size of a term.

Definition 5. *The size* $|M|$ *of a linear* λ*-term* M *is inductively defined as follows:*

1. $|\mathrm{a}| = 1$
2. $|\mathbf{X}| = 0$
3. $|x| = 1$
4. $|\lambda x.\, M| = |M|$
5. $|M\, N| = |M| + |N|$ ■

The set of linear typed λ-term is a subset of the simply typed λ-terms that is closed under β-reduction. Consequently it inherits the properties of confluence, subject reduction, and strong normalisation. This last property is also an obvious consequence of the following easy lemma.

Lemma 1. *Let* M *and* N *be two linear typed* λ*-terms such that* $M \to_\beta N$*. Then* $|M| = |N| + 1$*.*

Proof. A direct consequence of the fact that any λ-abstraction $\lambda x.\, M$ is such that there is one and only one free occurrence of x in M. □

As a consequence of this lemma, we obtain that all the reduction sequences from one term to another one have the same length.

Lemma 2. *Suppose that* M *and* N *are two linear typed* λ*-terms such that* $M \overset{n}{\to}_\beta N$ *and* $M \overset{m}{\to}_\beta N$*. Then* $m = n$*.*

Proof. By iterating Lemma 1, we have $n = |M| - |N| = m$. □

The above lemma allows the following definition to be introduced.

Definition 6. *The reducibility degree $\nu(M)$ of a linear typed λ-term M is defined to be the unique natural number n such that $M \xrightarrow{n}_{\beta} N$, where N is the β-normal form of M.* ■

We also introduce the following notions of complexity.

Definition 7. *The complexity $\rho(\alpha)$ of a linear type α is defined to be the number of "$\multimap$" it contains:*

1. $\rho(\mathrm{a}) = 0$
2. $\rho(\alpha \multimap \beta) = \rho(\alpha) + \rho(\beta) + 1$

The complexity $\rho(\Delta)$ of a β-redex $\Delta = (\lambda x.\, M)\, N$ is defined to be the complexity of the type of the abstraction $\lambda x.\, M$. ■

Lemma 3. *Let $M \in \mathscr{T}_{\alpha \multimap \beta}$, and $N \in \mathscr{T}_{\alpha}$. Then*

$$\sum_{\Delta \subset M\, N} \rho(\Delta) \leq \sum_{\Delta \subset M} \rho(\Delta) + \sum_{\Delta \subset N} \rho(\Delta) + \rho(\alpha \multimap \beta).$$

Proof. In case M is not an abstraction, we have $\sum_{\Delta \subset M\, N} \rho(\Delta) = \sum_{\Delta \subset M} \rho(\Delta) + \sum_{\Delta \subset N} \rho(\Delta)$. Otherwise we have $\sum_{\Delta \subset M\, N} \rho(\Delta) = \sum_{\Delta \subset M} \rho(\Delta) + \sum_{\Delta \subset N} \rho(\Delta) + \rho(\alpha \multimap \beta)$, because $M\, N$ itself is a β-redex whose complexity is $\rho(\alpha \multimap \beta)$. □

Lemma 4. *Let $M \in \mathscr{T}$, and $N, x \in \mathscr{T}_{\alpha}$ be such that $x \in \mathrm{FV}(M)$. Then*

$$\sum_{\Delta \subset M[x:=N]} \rho(\Delta) \leq \sum_{\Delta \subset M} \rho(\Delta) + \sum_{\Delta \subset N} \rho(\Delta) + \rho(\alpha).$$

Proof. By induction on the structure of M. The only case that is not straightforward is when $M \equiv x\, O$ and N is an abstraction. In this case, one additional redex of complexity $\rho(\alpha)$ is created. □

We now prove the key lemma of this section.

Lemma 5. *Let M be any linear typed λ-term. The following inequality holds:*

$$\nu(M) \leq \sum_{\Delta \subset M} \rho(\Delta) \tag{1}$$

Proof. The proof is done by induction on the length of M. We distinguish between two cases according to the structure of M.

$M \equiv \xi\, N_0 \ldots N_{n-1}$ where $\xi \equiv \mathrm{a}$, $\xi \equiv \mathbf{X}$, or $\xi \equiv x$, and where the sequence of terms $(N_i)_{i \in n}$ is possibly empty. We have that $\nu(M) = \sum_{i \in n} \nu(N_i)$ and

that $\sum_{\Delta\subset M}\rho(\Delta) = \sum_{i\in n}\sum_{\Delta\subset N_i}\rho(\Delta)$. Consequently, (1) holds by induction hypothesis.

$M \equiv (\lambda x.\, N)\, N_0 \ldots N_{n-1}$. Let the type of $\lambda x.\, N$ be $\alpha \multimap \beta$. If $n = 0$ (i.e., the sequence of terms $(N_i)_{i\in n}$ is empty) the induction is straightforward. If $n = 1$, we have:

$$\begin{aligned}
\nu(M) &= \nu(N[x{:=}N_0]) + 1 \\
&\leq \textstyle\sum_{\Delta\subset N[x:=N_0]}\rho(\Delta) + 1 && \text{(by induction hypothesis)} \\
&\leq \textstyle\sum_{\Delta\subset N}\rho(\Delta) + \sum_{\Delta\subset N_0}\rho(\Delta) + \rho(\alpha) + 1 && \text{(by lemma 4)} \\
&\leq \textstyle\sum_{\Delta\subset N}\rho(\Delta) + \sum_{\Delta\subset N_0}\rho(\Delta) + \rho(\alpha \multimap \beta) \\
&= \textstyle\sum_{\Delta\subset M}\rho(\Delta)
\end{aligned}$$

Finally, if $n \geq 2$:

$$\begin{aligned}
\nu(M) &= \nu(N[x{:=}N_0]\, N_1 \ldots N_{n-1}) + 1 \\
&\leq \textstyle\sum_{\Delta\subset N[x:=N_0]\, N_1 \ldots N_{n-1}}\rho(\Delta) + 1 && \text{(by induction hypothesis)} \\
&= \textstyle\sum_{\Delta\subset N[x:=N_0]\, N_1}\rho(\Delta) + \sum_{i\in n-2}\sum_{\Delta\subset N_{i+2}}\rho(\Delta) + 1 \\
&\leq \textstyle\sum_{\Delta\subset N[x:=N_0]}\rho(\Delta) + \rho(\beta) + \sum_{i\in n-1}\sum_{\Delta\subset N_{i+1}}\rho(\Delta) + 1 && \text{(by lemma 3)} \\
&\leq \textstyle\sum_{\Delta\subset N}\rho(\Delta) + \rho(\alpha) + \rho(\beta) + \sum_{i\in n}\sum_{\Delta\subset N_i}\rho(\Delta) + 1 && \text{(by lemma 4)} \\
&= \textstyle\sum_{\Delta\subset N}\rho(\Delta) + \rho(\alpha \multimap \beta) + \sum_{i\in n}\sum_{\Delta\subset N_i}\rho(\Delta) \\
&= \textstyle\sum_{\Delta\subset M}\rho(\Delta) && \square
\end{aligned}$$

Linearity plays a central role in the above Lemmas. In fact, Lemmas 1, 2, 4, and 5 do not hold for the simply typed λ-calculus. This is quite clear for Lemmas 1 and 2. Moreover, without the latter, Definition 6 does not make sense. Nevertheless, one might try to adapt this definition to the case of the simply typed λ-calculus by defining the reducibility degree of a λ-term M to be the *maximal* natural number n such that $M \xrightarrow{n}_\beta N$ (where N is the β-normal form of M). Lemma 4 could also be adapted by taking into account the number of free occurrences of x in M. But then, any attempt in adapting Lemma 5 would fail because linearity does not play a part only in the statement of this last lemma, but also in its proof. Indeed, this proof is done by induction on the length of a term M and, in case $M \equiv (\lambda x.\, N)\, N_0$, we apply the induction hypothesis to the term $N[x{:=}N_0]$. Now, if x occurs more than once in N, there is no reason why the length of $N[x{:=}N_0]$ should be less than the length of $(\lambda x.\, N)\, N_0$.

We now prove the main result of this paper.

Proposition 1. *Linear higher-order matching (modulo β) is decidable.*

Proof. We prove that the length of any possible solution is bounded, which implies that the set of possible solutions is finite.

Let $\langle M, N\rangle$ be a linear higher-order matching problem, and assume, without loss of generality, that M and N are β-normal. Let $(\mathbf{X}_i)_{i\in n}$ be the unknowns that occur in M, and suppose that $(\mathbf{X}_i{:=}O_i)_{i\in n}$ is a solution to the problem where the O_i's are β-normal. By Lemma 1, we have:

$$\nu(M[\mathbf{X}_i{:=}O_i]_{i\in n}) = |M[\mathbf{X}_i{:=}O_i]_{i\in n}| - |N| \qquad (1)$$

Let n_i be the number of occurrences of $\mathbf{X}_i$ in M. Equation (1) may be rewritten as follows:

$$\nu(M[\mathbf{X}_i{:=}O_i]_{i\in n}) = \sum_{i\in n} n_i|O_i| + |M| - |N| \tag{2}$$

On the other hand, by Lemma 5, we have:

$$\nu(M[\mathbf{X}_i{:=}O_i]_{i\in n}) \leq \sum_{\Delta\in M[\mathbf{X}_i := O_i]_{i\in n}} \rho(\Delta) \tag{3}$$

Since M and $(O_i)_{i\in n}$ are β-normal, the β-redexes that occur in $M[\mathbf{X}_i{:=}O_i]_{i\in n}$ are created by the substitution, which is the case whenever some O_i is an abstraction and some subterm of the form $\mathbf{X}_i\, P$ occurs in M. Consequently, we have:

$$\sum_{\Delta\in M[\mathbf{X}_i := O_i]_{i\in n}} \rho(\Delta) \leq \sum_{i\in n} n_i\rho(\alpha_i) \tag{4}$$

where α_i is the type of the unknown $\mathbf{X}_i$. From (3) and (4), we have

$$\nu(M[\mathbf{X}_i{:=}O_i]_{i\in n}) \leq \sum_{i\in n} n_i\rho(\alpha_i) \tag{5}$$

Finally, from (2) and (5), we obtain

$$\sum_{i\in n} n_i|O_i| \leq |N| - |M| + \sum_{i\in n} n_i\rho(\alpha_i) \tag{6}$$

which gives an upper bound to the length of the solution. □

As a corollary to this proposition, we immediately obtain that higher-order linear matching modulo $\eta\beta$ is decidable because the set of η-expanded β-normal forms closed by abstraction and application is provably closed under substitution and β-reduction [10].

4 NP-completeness

Note that the upper bound given by Proposition 1 is polynomial in the length of the problem. Moreover, β- and $\beta\eta$-conversion between two pure linear λ-terms may be decided in polynomial time since normalization is linear. Hence, a non deterministic Turing machine may guess a substitution and check that this substitution is indeed a solution in polynomial time. Consequently, linear higher-order matching belongs to NP. In fact, as we show in this section, it is NP-complete.

Let Σ be an alphabet containing at least two symbols, and let $\mathcal{X}$ be a countable set of variables. We write Σ^* (respectively, Σ^+) for the set of words (respectively, non-empty words) generated by Σ. We denote the concatenation of two words u and v by $u \cdot v$. The next proposition, which states the NP-completeness of associative matching, is due to Angluin [1, Theorem 3.6].

Proposition 2. *Given $v \in (\Sigma \cup \mathcal{X})^*$ and $w \in \Sigma^*$, deciding whether there exists a non-erasing substitution $\sigma : \mathcal{X} \to \Sigma^+$ such that $\sigma(v) = w$ is NP-complete.* □

We need a slight variant of this proposition in order to take erasing substitutions into account.

Proposition 3. *Given $v \in (\Sigma \cup \mathcal{X})^*$ and $w \in \Sigma^*$, deciding whether there exists a substitution $\sigma : \mathcal{X} \to \Sigma^*$ such that $\sigma(v) = w$ is NP-complete.*

Proof. Consider a symbol, say $\#$, that does not belong to Σ. For any word $u \in (\Sigma \cup \mathcal{X})^*$, let $\underline{u}$ be inductively defined as follows:

1. $\underline{\epsilon} = \epsilon$, where ϵ is the empty word;
2. $\underline{\mathbf{a} \cdot u'} = \# \cdot \mathbf{a} \cdot \underline{u'}$, where $\mathbf{a} \in (\Sigma \cup \mathcal{X})$, and $u' \in (\Sigma \cup \mathcal{X})^*$.

Let $v' \in (\Sigma \cup \mathcal{X})^*$ and $w' \in \Sigma^*$. It is almost immediate that there exists a substitution $\sigma : \mathcal{X} \to \Sigma^+$ such that $\sigma(v') = w'$ if and only if there exists a substitution $\tau : \mathcal{X} \to (\Sigma \cup \{\#\})^*$ such that $\tau(\underline{v'}) = \underline{w'}$. □

In order to get our NP-completeness result, it remains to reduce the problem of the above proposition to a linear higher-order matching problem. The trick is to encode word concatenation as function composition.

Proposition 4. *Linear higher-order matching is NP-complete.*

Proof. Let $\iota \in \mathscr{A}$ be an atomic type, and let $\Sigma_{\iota \multimap \iota} = \Sigma$ and $\mathscr{Y}_{\iota \multimap \iota} = \mathcal{X}$. Finally, let $x \in \mathscr{X}_\iota$. For any word $u \in (\Sigma \cup \mathcal{X})^*$, we inductively define $\underline{u}$ as follows:

1. $\underline{\epsilon} = \lambda x.\, x$, where ϵ is the empty word;
2. $\underline{\mathbf{a} \cdot u'} = \lambda x.\, \mathbf{a}\,(\underline{u'}\, x)$, where $\mathbf{a} \in (\Sigma \cup \mathcal{X})$, and $u' \in (\Sigma \cup \mathcal{X})^*$.

It is easy to show that there exist a susbstitution $\sigma : \mathcal{X} \to \Sigma^*$ such that $\sigma(v) = w$ if and only if the syntactic equation $\langle \underline{v}, \underline{w} \rangle$ admits a solution (modulo β, or modulo $\beta\eta$). □

The existence of a set of constants $\Sigma_{\iota \multimap \iota}$ seems to be crucial in the above proof. Indeed, contrarily to the case of the simply typed λ-calculus, there is an essential difference between constants and free λ-variables. Clause 4 of Definition 3 implies that there is at most one free occurrence of any λ-variable in any linear λ-term. There is no such restriction on the constants. Consequently, a given constant may occur several time in the same linear λ-term. This fact is implicitely used in the above proof.

5 Heuristics for an implementation

In this section, we give a practical algorithm obtained by specializing Huet's unification procedure [9]. We first specify this algorithm by giving a set of transformations in the spirit of [11, 20]. These transformations obey the following form:

$$e \longrightarrow \langle S_e,\, \sigma_e \rangle \qquad (*)$$

where e is a syntactic equation, S_e is a set of syntactic equations, and σ_e is a substitution. Transformations such as $(*)$ may then be applied to pairs $\langle S, \sigma \rangle$ made of a set S of syntactic equations, and a substitution σ:

$$\langle S, \sigma \rangle \longrightarrow \langle \sigma_e((S \setminus \{e\}) \cup S_e), \sigma_e \circ \sigma \rangle \qquad (**)$$

provided that $e \in S$. By iterating $(**)$, one possibly exhausts the set S:

$$\langle S, \sigma \rangle \longrightarrow^* \langle \varnothing, \tau \rangle,$$

in which case τ is intended to be a solution of the system of syntactic equations S.

For the sake of simplicity, we specify an algorithm that solves the problem modulo $\beta\eta$. The set of transformations is given by the three schemes listed below. All the λ-terms occurring in these schemes are considered to be in η-expanded β-normal forms.

1. *Simplification*:

$$\begin{aligned} e &\equiv \langle\, \lambda\overline{x}_{\mathrm{P}}.\, \mathbf{a}(\overline{M}_{\mathrm{Q}}),\, \lambda\overline{x}_{\mathrm{P}}.\, \mathbf{a}(\overline{N}_{\mathrm{Q}}) \,\rangle \\ S_e &\equiv \{\, \langle \lambda\overline{x}_{\mathrm{P_i}}.\, M_i,\, \lambda\overline{x}_{\mathrm{P_i}}.\, N_i, \rangle \mid i \in \mathrm{Q} \,\} \\ \sigma_e &\equiv id \end{aligned}$$

   provided that $\mathrm{FV}(M_i) = \mathrm{FV}(N_i)$, and where $\mathbf{a}$ is either a constant or a bound variable, and the family of sets $(P_i)_{i \in Q}$ is such that $\{\, x_j \mid j \in P_i \,\} = \mathrm{FV}(M_i)$.

2. *Imitation*:

$$\begin{aligned} e &\equiv \langle\, \lambda\overline{x}_{\mathrm{P}}.\, \mathbf{X}(\overline{M}_{\mathrm{Q}}),\, \lambda\overline{x}_{\mathrm{P}}.\, \mathrm{a}(\overline{N}_{\mathrm{R}}) \,\rangle \\ S_e &\equiv \{\, e \,\} \\ \sigma_e &\equiv (\mathbf{X} := \lambda\overline{y}_{\mathrm{Q}}.\, a(\lambda\overline{z}_{n_i}.\, \mathbf{Y}_i(\overline{y}_{\mathrm{Q}_i})(\overline{z}_{n_i}))_{i \in \mathrm{R}}) \end{aligned}$$

   where $(\mathrm{Q}_i)_{i \in \mathrm{R}}$ is a family of disjoint sets such that $\bigcup_{i \in \mathrm{R}} \mathrm{Q}_i = Q$, $(\mathbf{Y}_i)_{i \in \mathrm{R}}$ is a family of fresh unknowns whose types may be inferred from the context.

3. *Projection*:

$$\begin{aligned} e &\equiv \langle\, \lambda\overline{x}_{\mathrm{P}}.\, \mathbf{X}(\overline{M}_{\mathrm{Q}}),\, \lambda\overline{x}_{\mathrm{P}}.\, \mathbf{a}(\overline{N}_{\mathrm{R}}) \,\rangle \\ S_e &\equiv \{\, e \,\} \\ \sigma_e &\equiv (\mathbf{X} := \lambda\overline{y}_{\mathrm{Q}}.\, y_k(\lambda\overline{z}_{n_i}.\, \mathbf{Y}_i(\overline{y}_{\mathrm{Q}_i})(\overline{z}_{n_i}))_{i \in m}) \end{aligned}$$

   where $k \in \mathrm{Q}$, m is the arity of y_k, $(\mathrm{Q}_i)_{i \in m}$ is a family of disjoint sets such that $\bigcup_{i \in m} \mathrm{Q}_i = \mathrm{Q} \setminus \{k\}$, $(\mathbf{Y}_i)_{i \in m}$ is a family of fresh unknowns of the appropriate type.

We now sketch the proof that the above set of transformations is correct and complete.

Lemma 6. *Let $e \longrightarrow \langle S_e, \sigma_e \rangle$ be one of the above transformations. Let S be a set of syntactic equations such that $e \in S$. If σ is a substitution that solves all the syntactic equations in $\sigma_e((S \setminus \{e\}) \cup S_e)$ then $\sigma \circ \sigma_e$ solves all the syntactic equations in S.*

Proof. It suffices to show that $\sigma \circ \sigma_e$ solves e whenever σ solves $\sigma_e(S_e)$, which is trivial for the three transformations. □

Lemma 7. *Let S and T be two sets of syntactic equations, and let τ be a substitution such that $\langle S, id \rangle \longrightarrow^* \langle T, \tau \rangle$. If σ is a substitution that solves all the syntactic equations in T then $\sigma \circ \tau$ solves all the syntactic equations in S.*

Proof. By iterating the previous lemma. □

As a direct consequence of this lemma, we obtain the correctness of the transformational algorithm.

Proposition 5. *Let $e = \langle M, N \rangle$ be a syntactic equation and σ be a substitution such that $\langle \{e\}, id \rangle \longrightarrow^* \langle \varnothing, \sigma \rangle$. Then $\sigma(M) = N$.* □

We now prove the completeness of our algorithm.

Proposition 6. *Let $e = \langle M, N \rangle$ be a syntactic equation and σ be a substitution such that $\sigma(M) = N$. Then, there exists a sequence of transitions such that $\langle \{e\}, id \rangle \longrightarrow^* \langle \varnothing, \sigma' \rangle$, and σ' agrees with σ on the set of unknowns occurring in M.*

Proof. Let S be a non-empty set of syntactic equations and let σ be a substitution that solves all the equations in S. One easily shows—see [20, Lemmas 4.16 and 4.17], for details—that there exists $e \in S$ together with a transformation

$$e \longrightarrow \langle S_e,\, \sigma_e \rangle, \tag{1}$$

and a substitution τ such that:

1. $\sigma = \tau \circ \sigma_e$,
2. τ solves $\sigma_e((S \setminus \{e\}) \cup S_e)$.

Consequently, by iterating (1) on some system R_0 that is solved by some substitution σ_0, one obtains a sequence of transitions:

$$\langle R_0,\, id \rangle \longrightarrow \langle R_1,\, \rho_1 \rangle \longrightarrow \langle R_2,\, \rho_2 \rangle \longrightarrow \cdots \tag{2}$$

together with a sequence of substitutions $(\sigma_0, \sigma_1, \sigma_2, \ldots)$ such that:

1. $\sigma_0 = \sigma_i \circ \rho_i$,
2. σ_i solves R_i.

It remains to prove that (2) eventually terminates and, therefore, exhausts the set R_0. To this end, define the size of a system S (in notation, $|S|$) to be the sum of the sizes of the right-hand sides of the syntactic equations occurring in S. Define also the size of a substitution σ with respect to a system S (in notation, $|\sigma|_S$) to be the sum of the sizes of the terms substituted for the unknowns occurring in S. Transformation (1) and substitution τ are such that

$$|\tau|_T \leq |\sigma|_S,$$

where $T = \sigma_e((S \setminus \{e\}) \cup S_e)$. It is then easy to show that each transition of (2) strictly decreases the pair $(|R_i|, |\sigma_i|_{R_i})$ according to the lexicographic ordering. □

The three transformations we have given specify a non-deterministic algorithm. Its practical implementation would therefore appeal to some backtracking mechanism. This is not surprising since we have proved linear higher-order matching to be NP-complete. Nevertheless, some source of non-determinism could be avoided. We conclude by discussing this issue.

A naive implementation of the transformational algorithm would give rise to backtracking steps for two reasons:

1. the current non-empty set of syntactic equations is such that no transformation applies;
2. the size of the current substitution is strictly greater than the upper bound given by proposition 1.

It is easy to see that the first case of failure can be detected earlier. Indeed, if no transformation applies to a system S, it means that all the syntactic equations in S have the following form:

$$\langle\, \lambda\overline{x}_m.\, \mathbf{a}(\overline{M}_n),\ \lambda\overline{x}_m.\, \mathbf{b}(\overline{N}_o)\,\rangle \tag{3}$$

where either $\mathbf{a} \neq \mathbf{b}$, or $\mathbf{a} = \mathbf{b}$ but there exists $k \in n$ such that $\mathrm{FV}(M_k) \neq \mathrm{FV}(N_k)$. Now, it is clear that such equations cannot be solved. Consequently, one may fail as soon as a system S contains at least one equation like (3). In addition, one may easily prove that any application of the *simplification* rule does not alter the set of possible solutions. Therefore *simplification* may be applied deterministically. These observations give rise to the following heuristic:

> *Start by Applying repeatedly* simplification *until all the heads of the left-hand sides of the equations are unknowns. If this is not possible, fail.*

This heuristic is not proper to our linear matching problem. In fact, it belongs to the folklore of higher-order unification. We end this section by giving some further heuristic principles that are specific to the *linear aspects* of our problem.

The next three lemmas, whose elementary proofs are left to the reader, will allow us to state another condition of possible failure that we may check before applying any transformation. Let $\#_{\mathrm{a}}(M)$ denote the number of occurrences of a given constant "a" in some λ-term M. Similarly, let $\#_{\mathbf{X}}(M)$ denote the number of occurrences in M of some unknown $\mathbf{X}$.

Lemma 8. *Let M and N be two linear λ-terms, and let $x \in \mathrm{FV}(M)$. Then, for every constant* a*, $\#_{\mathrm{a}}(M[x{:=}N]) = \#_{\mathrm{a}}(M) + \#_{\mathrm{a}}(N)$.* □

Lemma 9. *Let M and N be two linear λ-terms such that $M \twoheadrightarrow_\beta N$. Then, for every constant* a*, $\#_{\mathrm{a}}(M) = \#_{\mathrm{a}}(N)$.* □

Lemma 10. *Let M and N be two linear λ-terms, and $\mathbf{X}$ be some unknown. Then, for every constant* a*, $\#_{\mathrm{a}}(M[\mathbf{X}{:=}N]) = \#_{\mathrm{a}}(M) + \#_{\mathbf{X}}(M) \times \#_{\mathrm{a}}(N)$.* □

As a consequence of these lemmas, we have that the number of occurences of any constant in the left-hand side of any equation cannot decrease. This allows us to state the following failure condition.

> *Check, for every constant* a*, that each equation $\langle M, N\rangle$ is such that $\#_{\mathrm{a}}(M) \leq \#_{\mathrm{a}}(N)$. If this is not the case, fail.*

The above condition may be checked before applying any transformation, and then kept as an invariant. To this end, it must be incorporated as a proviso to the simplification rule. Then, the choice between *imitation* and/or *projection* must obey the following principle:

> *When considering an equation such as*
>
> $$\langle\, \lambda\overline{x}_{\mathrm{P}}.\, \mathbf{X}(\overline{M}_{\mathrm{Q}}),\ \lambda\overline{x}_{\mathrm{P}}.\, a(\overline{N}_{\mathrm{R}})\,\rangle$$
>
> *check whether there exist some equation $\langle A, B\rangle$ (including the above equation) such that $\#_{\mathrm{a}}(A) + \#_{\mathbf{X}}(A) > \#_{\mathrm{a}}(B)$. If this is the case,* projection *is forced. Otherwise, try* imitation *before trying* projection.

The reason for trying imitation first, which is a heuristic used by Paulson in Isabelle [18], is that each application of *imitation* gives rise to a subsequent *simplification*.

When applying *imitation*, we face the problem of guessing the family of sets $(\mathrm{Q}_i)_{i\in\mathrm{R}}$. This source of non-determinism is typical of linear higher-order unification.[1] Now, since any application of *imitation* may be immediately followed by a *simplification*, the family $(\mathrm{Q}_i)_{i\in\mathrm{R}}$ should be such that the subsequent *simplification* may be applied. We now explain how this constraint may be satisfied.

Consider some linear λ-term A whose η-expanded β-normal form is:

$$\lambda\overline{x}_{\mathrm{P}}.\, \mathbf{a}(\overline{M}_{\mathrm{Q}})$$

where $\mathbf{a}$ is not a bound variable. We define the incidence function of A to be the unique $f_A : \mathrm{P} \to \mathrm{Q}$ such that:

$$f_A(i) = j \quad \text{if and only if} \quad x_i \in \mathrm{FV}(M_j).$$

[1] It is due to the multiplicative nature of the connective "$\multimap$" and is reminiscent of the context-splitting problem one has to solve when trying to prove a multiplicative sequent of the form $\Gamma \vdash A \otimes B$ by a backward application of the $\otimes$-introduction rule.

Now, consider the two following λ-terms:

$$A \equiv \lambda \overline{x}_{\mathrm{P}}.\, \mathbf{X}(\overline{M}_{\mathrm{Q}})$$
$$B \equiv \lambda \overline{x}_{\mathrm{Q}}.\, \mathbf{a}(\overline{N}_{\mathrm{R}})$$

It is not difficult to show that the incidence function of $A[\mathbf{X}{:=}B]$ is such that:

$$f_{A[\mathbf{X}:=B]} = f_B \circ f_A \tag{4}$$

We will apply the above identity to the case of the *imitation* rule. Let A, B, and C be the terms involved in the definition of an *imitation* step:

$$A \equiv \lambda \overline{x}_{\mathrm{P}}.\, \mathbf{X}(\overline{M}_{\mathrm{Q}})$$
$$B \equiv \lambda \overline{x}_{\mathrm{P}}.\, \mathrm{a}(\overline{N}_{\mathrm{R}})$$
$$C \equiv \lambda \overline{y}_{\mathrm{Q}}.\, a(\lambda \overline{z}_{n_i}.\, \mathbf{Y}_i(\overline{y}_{\mathrm{Q_i}})(\overline{z}_{n_i}))_{i \in \mathrm{R}}$$

After the imitation step is performed, equation $\langle A, B\rangle$ is replaced by equation:

$$\langle A[\mathbf{X}{:=}C], B\rangle \tag{5}$$

Simplification may then be applied to (5) provided that

$$f_{A[\mathbf{X}:=C]} = f_B \tag{6}$$

which, by (4), is equivalent to

$$f_C \circ f_A = f_B \tag{7}$$

Note that both f_A and f_B are known, while f_C is uniquely determined by the family of sets $(\mathrm{Q}_i)_{i \in \mathrm{R}}$, and conversely, since

$$f_C(i) = j \quad \text{if and only if} \quad y_i \in \mathrm{Q}_j$$

Therefore, in order to find an appropriate family of sets $(\mathrm{Q}_i)_{i \in \mathrm{R}}$, it suffices to solve (7). Now, it is an elementary theorem of set theory that (7) admits a solution if and only if

$$(\forall i, j \in \mathrm{P}) f_A(i) = f_A(j) \Rightarrow f_B(i) = f_B(j),$$

which gives a condition that may be checked before applying any *imitation.*

Acknowledgement

I would like to thank G. Dowek and M. Rusinowitch for interesting discussions about some of the material in this paper.

References

1. D. Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.
2. I. Cervesato and F. Pfenning. Linear higher-order pre-unification. In *Proceedings of the 12th annual IEEE symposium on logic in computer science*, pages 422–433, 1997.
3. H. Comon. Completion of rewrite systems with membership constraints. Part I: Deduction rules. *Journal of Symbolic Computation*, 25(4):397–419, 1998.
4. H. Comon. Completion of rewrite systems with membership constraints. Part II: Constraint solving. *Journal of Symbolic Computation*, 25(4):421–453, 1998.
5. G. Dowek. Third order matching is decidable. *Annals of Pure and Applied Logic*, 69(2–3):135–155, 1994.
6. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
7. W. D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13(2):225–230, 1981.
8. G. Huet. The undecidability of unification in third order logic. *Information and Control*, 22(3):257–267, 1973.
9. G. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.
10. G. Huet. *Résolution d'équations dans les langages d'ordre* $1, 2, \ldots, \omega$. Thèse de Doctorat d'Etat, Université Paris 7, 1976.
11. J. Levy. Linear second-order unification. In H. Ganzinger, editor, *Rewriting Techniques and Applications, RTA'96*, volume 1103 of *Lecture Notes in Computer Science*, pages 332–346. Springer-Verlag, 1996.
12. J. Levy. Decidable and undecidable second-order unification problems. In T. Nipkow, editor, *Rewriting Techniques and Applications, RTA'98*, volume 1379 of *Lecture Notes in Computer Science*, pages 47–60. Springer-Verlag, 1998.
13. J. M. Merenciano and G. Morrill. Generation as deduction on labelled proof nets. In C. Retoré, editor, *Logical Aspects of Computational Linguistics, LACL'96*, volume 1328 of *Lecture Notes in Artificial Intelligence*, pages 310–328. Springer Verlag, 1997.
14. D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.
15. M. Moortgat. Categorial type logic. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, chapter 2. Elsevier, 1997.
16. G. Morrill. *Type Logical Grammar: Categorial Logic of Signs.* Kluwer Academic Publishers, Dordrecht, 1994.
17. V. Padovani. *Filtrage d'ordre supérieure.* Thèse de Doctorat, Université de Paris 7, 1996.
18. L. C. Paulson. *Isabelle, a generic theorem prover*, volume 828 of *Lecture Notes in Computer Science.* Springer-Verlag, 1994.
19. M. Schmidt-Schauss and K. U. Schulz. Solvability of context equations with two context variables is decidable. In H. Ganzinger, editor, *16th International Conference on Automated Deduction*, volume 1632 of *Lecture Notes in Computer Science*, pages 67–81. Springer Verlag, 1999.
20. W. Snyder and J. Gallier. Higher order unification revisited: Complete sets of transformations. *Journal of Symbolic Computation*, 8(1–2):101–140, 1989.

21. T. Wierzbicki. Complexity of the higher order matching. In H. Ganzinger, editor, *16th International Conference on Automated Deduction*, volume 1632 of *Lecture Notes in Computer Science*, pages 82–96. Springer Verlag, 1999.