# Second-order unification and functional arity

Aleksy Schubert

▶ **To cite this version:**

Aleksy Schubert. Second-order unification and functional arity. 37th International Workshop on Unification (UNIF 2023), Veena Ravishankar; Christophe Ringeissen, Jul 2023, Rome, Italy. hal-04128012

**HAL Id: hal-04128012**
**https://inria.hal.science/hal-04128012**

Submitted on 14 Jun 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

This space is reserved for the EPiC Series header, do not use it

# Second-order unification and functional arity

Aleksy Schubert[1]*

Faculty of Mathematics, Informatics and Mechanics,
University of Warsaw
ul. Stefana Banacha 2
02-097 Warsaw
alx@mimuw.edu.pl

**Abstract**

We consider expressions that represent functional types where the main binary symbol is $\rightarrow$ and arbitrarily many constants are available. There are natural notions of rank and arity for such expressions that arise in the study of functional programs. We present an argument for undecidability of second-order unification in case solutions are restricted to expressions that have bounded rank, but unbounded arity.

## 1 Introduction

The interest in second-order unification in the context of functional programming languages arose with the work of Pfenning [10] where second-order unification turned out to be the basic mechanism for type reconstruction in a model functional programming language based upon System F of Girard and Reynolds [5, 11].

Further studies on the type reconstruction problems revealed that natural classes of restricted type reconstruction problems lead to very restricted forms of second-order unification [3]. In particular undecidability can be proved already in case second-order variables are applied only to ground terms (i.e. terms with no occurrences of substitutable variables) [12]. An interesting conclusion from these studies is that if each $\lambda$-abstraction term $\lambda x.M$ does not have type annotation for the variable $x$ while all other type annotations are mandatory, then type reconstruction is undecidable. The natural consequence of these results is that in type systems based upon System F we cannot omit type information for bound variables.

These works proved undecidability for even more restricted forms of second-order unification instances, namely ones where arguments of second-order variables are of depth 1 only. In this way, we obtained a very thin difference with Miller patterns [8], a decidable form of second-order unification, which in essence allowed only constants (of depth 1) in such arguments.

The basic idea behind application of unification to type checking and type reconstruction problems can be traced back to Morris [9] with more systematic accounts given by Hindley [6] and Curry [1]. It relies on the basic observation that lambda calculus application $MN$ forces

*This work was supported by IDUB POB 3 programme.

unification constraints of the form $X_M \doteq X_N \to X_{MN}$, where $X_M, X_N, X_{MN}$ represent (first-order unification) variables that eventually hold types of terms $M, N$ and $MN$ respectively.

The need to use second-order unification arises when polymorphic types are added. In certain simplification, we deal here with terms of the form $MA_1 \dots A_n N$ where $M, N$ are terms of lambda calculus such that $N$ is an argument of the functional $M$ while expressions $A_1, \dots, A_n$ are types that instantiate the polymorphic type of $M$. In this case the type of $M$ must be polymorphic and the term forces a unification constraint of the form $\mathsf{F}_M A_1 \dots A_n \doteq X_N \to X_{MA_1 \dots A_n N}$. In that case, we operate with help of a second-order unification variable $\mathsf{F}_M$ that eventually holds the shape of the polymorphic type of the term $M$.

Undecidability results give rise to studies in which restricted forms of systems are considered. In the context of systems based upon System F we can mention here the work of Gianini and Ronchi Della Rocca [4] where a variant of System F was proposed in which the typing of $MA_1 \dots A_n N$ is only possible when the type of $M$ is restricted so that variables are allowed only up to some depth $n$. This translates to a requirement on a solution of the corresponding equation $\mathsf{F}_M A_1 \dots A_n \doteq X_N \to X_{MA_1 \dots A_n N}$ which says that the bound variables used in the solution for $\mathsf{F}_M$ must be on depth $n$ at most. The pragmatic rationale behind this restriction is that programmers are able to digest typing information only of limited size. Seemingly similar motivation, to restrict depth of types substituted for unification variables, is present also in type systems based upon other principles (for instance in the case of intersection types [2]).

An interesting question that arises in this context is how firm the restriction on type depth is? In particular, is it possible to have a decidable system in which we allow arbitrary functional arity of types, but restrict functional rank to be bounded? Or, symmetrically, is it possible to have a decidable system in which we allow arbitrary rank, but restrict functional arity to be bounded? In this note, we give a partially negative answer to these questions. We give a class of instances for second-order unification problem for which the unification problem with any of the above mentioned restrictions is undecidable.

## 2 Preliminaries

We define here expressions for unification problems. We assume that a countably infinite set of type variables is divided into three subsets: the first one contains first-order variables denoted by $X, Y, \dots$, the second one first-order constants denoted by $c, d, \dots$, and the third one second-order variables denoted by $\mathsf{F}^n, \mathsf{G}^n, \dots$ where the superscript $n$ indicates their arity. Whenever it does not lead to confusion, we drop the superscript with arity. The expressions we deal with in unification equations have the following form:

$$A, B \quad ::= \quad X \mid c \mid (A \to B) \mid \mathsf{F}^n A_1 \cdots A_n \ .$$

The expressions which do not contain second-order variables are called *first-order expressions*. A set of free variables in expressions of unification problems is defined as follows: $\mathrm{FV}_u(c) = \emptyset$; $\mathrm{FV}_u(X) = \{X\}$; $\mathrm{FV}_u(\mathsf{F}A_1 \dots A_n) = \{\mathsf{F}\} \cup \bigcup_{i=1}^n \mathrm{FV}_u(A_i)$; $\mathrm{FV}_u(A \to B) = \mathrm{FV}_u(A) \cup \mathrm{FV}_u(B)$, where $c$ is a constant, $X$ is a first-order variable and $\mathsf{F}$ is a second-order variable.

To define substitutions we also need to introduce second-order expressions. These have the form $\lambda X_1, \dots, X_n.A$ where $A$ is an expression such that $\mathrm{FV}_u(A) \subseteq \{X_1, \dots, X_n\}$. We define the *functional arity* of a first-order expression as $\mathrm{arity}(c) = 0$ for a constant $c$ and $\mathrm{arity}(A_1 \to \cdots \to A_n \to c) = \max(\mathrm{arity}(A_1), \dots, \mathrm{arity}(A_n), n)$. For a second-order expression, somewhat unexpectedly, $\mathrm{arity}(\lambda X_1, \dots, X_n.A) = \mathrm{arity}(A)$. We define the *rank* as $\mathrm{rank}(c) = 0$ for a constant $c$, and $\mathrm{rank}(A \to B) = \max(\mathrm{rank}(A) + 1, \mathrm{rank}(B))$, and

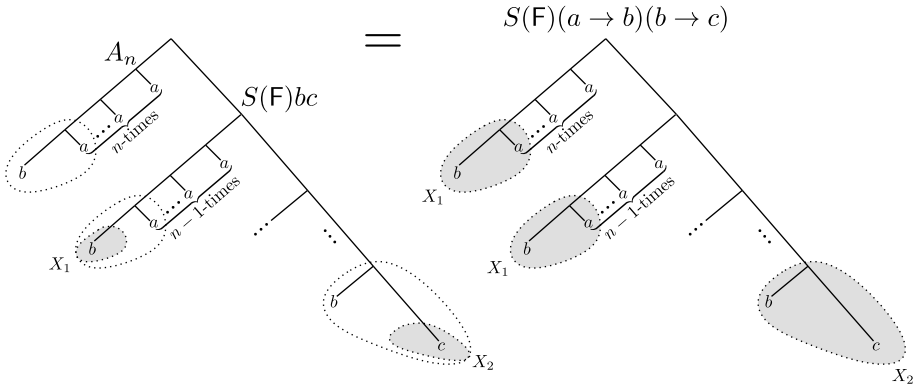


Figure 1: Rewriting considered in Example 2.2.

$\mathrm{rank}(\lambda X_1, \ldots, X_n.A) = \mathrm{rank}(A)$. A substitution $S$ is a partial function from the set of first-order and second-order variables to expressions and second-order expressions such that it maps first-order variables to first-order expressions and second-order variables to second-order expressions. We write $A[X_1 := A_1, \ldots, X_n := A_n]$ for a substitution that replaces all occurrences of first-order variables $X_1, \ldots, X_n$ in $A$ with respective expressions $A_1, \ldots, A_n$. In general case, application of a substitution $S$ to an expressions $A$ is $S(A)$ defined congruently by structure of $A$ with the critical cases defined as $S(A) = S(X)$ for $A = X$ being a first-order variable and $S(A) = B[X_1 := S(A_1), \ldots, X_n := S(A_n)]$ for $A = \mathsf{F}A_1 \ldots A_n$ and $S(\mathsf{F}) = \lambda X_1 \ldots X_n.B$.

An instance of the unification problem consists of a set of equations

$$E = \{A_1 \doteq B_1, \ldots, A_n \doteq B_n\}.$$

An instance $E$ is solvable if there exists a substitution $S$ such that $S(A_1) = S(B_1), \ldots, S(A_n) = S(B_n)$. We say that a solution $S$ obeys functional arity (rank) restriction on level $n$ when for each second-order variable $\mathsf{F}$ we have $\mathrm{arity}(S(\mathsf{F})) \leq n$ ($\mathrm{rank}(S(\mathsf{F})) \leq n$). We write $\mathrm{dom}(S)$ for the domain of $S$. We provide in this paper a sketch of the proof for the following theorem.

**Theorem 2.1.** `(undecidability of restricted unification)`
*The following problem is undecidable, given a set of equations $E$ decide if there is a substitution $S$ that solves $E$ and obeys the rank restriction (functional arity) restriction on level $n = 3$.*

The construction below is symmetric so it is enough to present the proof for only one of the restrictions. The descriptions below concern undecidability of the problem where we require the solutions to obey the rank restriction on level $n = 3$.

The basic unification mechanism used in our proof is illustrated by the following example.

**Example 2.2.** *Consider a ground term rewriting system with only one reduction rule $a \to b \rightsquigarrow b$. Consider expressions $A_n = \underbrace{a \to \cdots \to a}_{n\text{-}times} \to b$ and second-order equations*

$$A_n \to \mathsf{F}bc \doteq \mathsf{F}(a \to b)(b \to c)$$

*Each such equation has a solution $S = [\mathsf{F} := \lambda X_1 X_2.B_{n-1}(X_1) \to B_{n-2}(X_1) \to \ldots \to B_0(X_1) \to X_2]$ where $B_0(X_1) = X_1$ and $B_n(X_1) = a \to B_{n-1}(X_1)$. Figure 1 shows in the form of binary trees terms resulting from application of such a solution to the equation. We can see immediately that $S(\mathsf{F}(a \to b)(b \to c)) = A_n \to A_{n-1} \to \cdots \to A_0 \to c$ represents naturally*

*a sequence of rewrites $A_n \rightsquigarrow A_{n-1} \rightsquigarrow \cdots \rightsquigarrow A_0$. The mechanism of the rewrite can be traced on Figure 1 by observing the areas surrounded by dotted lines. Of course this equation has more solutions, for instance $S' = [\mathsf{F} := \lambda XY.B_{n-1}(X) \to A_{n-1} \to B_{n-1}(X) \to \ldots \to B_0(X) \to Y]$. They also represent a sequence of rewrites, but we allow to make stops in some places, as in $A_n \rightsquigarrow A_{n-1} = A_{n-1} \rightsquigarrow \cdots \rightsquigarrow A_0$. This construction naturally generalises to all ground rewriting systems, see [7] for details.*

# 3 Unification undecidability

The proof of undecidability is by reduction of the halting problem for Turing Machines. To facilitate the presentation we fix a deterministic Turing Machine $\mathcal{M} = \langle \Sigma, Q, q_t, f \rangle$ where $\Sigma$ is the tape alphabet of the machine, $Q$ is the set of states, $q_t \in Q$ is the terminal state and $f$ is a transition function described below. Here is a number of remarks that characterise the model in greater detail.

A configuration $C_i$ of $\mathcal{M}$ is represented as a sequence $a_{i,0}, \ldots, a_{i,m-1}$ of symbols from $\Sigma' = \Sigma \cup \Sigma \times Q$ where $\Sigma$ is the alphabet of the machine $\mathcal{M}$ and $Q$ is the set of states of $\mathcal{M}$. This sequence represents the content of the machine's tape in the natural way. The sequence contains a unique element $\langle a, q \rangle \in \Sigma \times Q$. The position of the element is the position of the machine's head, and $q$ is the current state of the machine. There is exactly one terminal state $q_t$ which is reached only in case the configuration has one element.

The history of $\mathcal{M}$ consists of a sequence $C_0, C_1, \ldots, C_{n-1}$ of configurations. We assume that these configurations may have different lengths. We also assume that the final configuration $C_{n-1}$ has length 1.

To facilitate the presentation, the transition function $f$ of $\mathcal{M}$ is not represented as an element of $\Sigma \times Q \to \Sigma \times Q \times \{L, R, S\}$ where $f(a, q) = \langle a', q', d \rangle$ is interpreted so that *when the machine is in state $q$ and its head reads the symbol $a$ then write $a'$, transfer to state $q'$ and move head as $d$ prescribes (left, right, stay, respectively).* Instead we represent the transition function $f$ of $\mathcal{M}$ equivalently as an element of $\Sigma' \times \Sigma' \times \Sigma' \to \Sigma'$ so that $f(a_{i,j-1}, a_{i,j}, a_{i,j+1}) = a_{i+1,j}$ holds for elements of configurations.

For simplicity we assume that there is some symbol $\bullet \in \Sigma$ that never occurs on a tape of $\mathcal{M}$, but is used to mark that the last position of the machine's tape is missing. It is used to handle both deleting machine cells and inserting new cells.

In our language of unification expressions, we represent elements of $Q, \Sigma, \Sigma'$ as constants. A configuration $C_i = a_{i,0}, \ldots, a_{i,m-1}$ is represented as $|C_i| = a_{i,0} \to \ldots \to a_{i,m-1} \to o$ where $o$ is a special constant outside of $Q, \Sigma, \Sigma'$. We represent history $C_0, \ldots, C_{n-1}$ as

$$\mathsf{Hist} = |C_0| \to |C_1| \to \ldots \to |C_{n-2}| \to |C_{n-1}| \to o'.$$

where $o'$ is another special constant outside of $Q, \Sigma, \Sigma'$.

Given each triple $\mathsf{t} \in \Sigma' \times \Sigma' \times \Sigma'$, we introduce a first-order variable $X_{\mathsf{t}}$. Suppose that $C_i = a_{i,0}, \ldots, a_{i,j-2}, a_{i,j-1}, a_{i,j}, a_{i,j+1}, a_{i,j+2}, \ldots, a_{i,n-1}$ where $j$ is the position of the head of $\mathcal{M}$, i.e. $a_{i,j} \in \Sigma \times Q$. Consider triples $\mathsf{u1} = \langle a_{i,j-2}, a_{i,j-1}, a_{i,j} \rangle$, $\mathsf{u2} = \langle a_{i,j1}, a_{i,j}, a_{i,j+1} \rangle$, $\mathsf{u3} = \langle a_{i,j}, a_{i,j+1}, a_{i,j+2} \rangle$. We can now define $[C_i]$ in three forms

$$\begin{array}{ll} (!) & [C_i] = a_{i,0} \to \ldots \to a_{i,j-2} \to X_{\mathsf{u1}} \to X_{\mathsf{u2}} \to X_{\mathsf{u3}} \to a_{i,j+2} \to \cdots \to a_{i,n-1} \to o, \\ (!!) & [C_i] = a_{i,0} \to \ldots \to a_{i,n-3} \to X_{\mathsf{u1}} \to X_{\mathsf{u2}}, \\ (!!!) & [C_i] = a_{i,0} \to \ldots \to a_{i,n-2} \to X_{\mathsf{u1}} \to X_{\mathsf{u2}} \end{array}$$

where the case (!) is used in all standard situations, case (!!) is used when $f(\mathsf{u2}) = \bullet$, i.e. the rule deletes the last cell of the Turing Machine tape, and (!!!) is used when $\pi_2(\mathsf{u2}) = \bullet$, i.e. the

rule inserts a new last cell of the Turing Machine tape. We immediately see that

$$\begin{array}{lllll}
(A) & [C_i][X_{\mathsf{u1}} := \pi_2(\mathsf{u1}), X_{\mathsf{u2}} := \pi_2(\mathsf{u2}), X_{\mathsf{u3}} := \pi_2(\mathsf{u3})] & = |C_i| & \text{(case (!), standard)} \\
(B) & [C_i][X_{\mathsf{u1}} := f(\mathsf{u1}), X_{\mathsf{u2}} := f(\mathsf{u2}), X_{\mathsf{u3}} := f(\mathsf{u3})] & = |C_{i+1}| & \text{(case (!), standard)} \\
(A) & [C_i][X_{\mathsf{u1}} := \pi_2(\mathsf{u1}), X_{\mathsf{u2}} := \pi_2(\mathsf{u2}) \to o] & = |C_i| & \text{(case (!!), delete)} \\
(B) & [C_i][X_{\mathsf{u1}} := f(\mathsf{u1}), X_{\mathsf{u2}} := o] & = |C_{i+1}| & \text{(case (!!), delete)} \\
(A) & [C_i][X_{\mathsf{u1}} := \pi_2(\mathsf{u1}), X_{\mathsf{u2}} := o] & = |C_i| & \text{(case (!!!), insert)} \\
(B) & [C_i][X_{\mathsf{u1}} := f(\mathsf{u1}), X_{\mathsf{u2}} := f(\mathsf{u2}) \to o] & = |C_{i+1}| & \text{(case (!!!), insert)}
\end{array} \tag{1}$$

where $\pi_2(x)$ is the projection on the second coordinate of a triple. Consider now

$$D_{\mathsf{F}} = [C_0] \to [C_1] \to \cdots \to [C_{n-1}] \to C_n \to Z.$$

We can now order all triples in $\mathrm{dom}(f)$ as $\mathsf{s}1, \ldots, \mathsf{s}p$ and let expressions $A_1, \ldots, A_p$ and expressions $B_1, \ldots, B_p$ be defined as

$$\begin{array}{lll}
A_i = \pi_2(\mathsf{s}i), & B_i = f(\mathsf{s}i) & \text{whenever } \pi_2(\mathsf{s}i) \neq \bullet \text{ and } f(\mathsf{s}i) \neq \bullet; \\
A_i = \pi_2(\mathsf{s}i) \to o, & B_i = o & \text{whenever } \pi_2(\mathsf{s}i) \neq \bullet \text{ and } f(\mathsf{s}i) = \bullet \text{ (delete)}; \\
A_i = o, & B_i = f(\mathsf{s}i) \to o & \text{whenever } \pi_2(\mathsf{s}i) = \bullet \text{ and } f(\mathsf{s}i) \neq \bullet \text{ (insert)}.
\end{array}$$

Given observations (1), we immediately see that

$$\begin{array}{l}
D_{\mathsf{F}}[X_{\mathsf{s}1} := A_1, \ldots, X_{\mathsf{s}p} := A_p, Z := C_{n-1} \to o'] = \mathsf{Hist} \\
C_0 \to (D_{\mathsf{F}}[X_{\mathsf{s}1} := B_1, \ldots, X_{\mathsf{s}p} := B_p, Z := o']) = \mathsf{Hist}
\end{array}$$

This shows that an equation

$$C_0 \to \mathsf{F}B_1 \ldots B_k o \doteq \mathsf{F}A_1 \ldots A_p(C_{n-1} \to o) \tag{2}$$

can simulate computation of $\mathcal{M}$. Still, this can be done under condition that the following consistency assumptions concerning variables are met

- variables $X_{\mathsf{u1}}, X_{\mathsf{u2}}, X_{\mathsf{u3}}$ take up consecutive places in a configuration;
- variables $X_{\mathsf{u1}}, X_{\mathsf{u2}}, X_{\mathsf{u3}}$ are consistent with the content of the configuration.

This can be done with help of another ground rewriting system. In this system we work with extended set of constants. These include in addition to constants used so far: pairs from $\Delta = \Sigma' \times \{o, 4\}$, triples from $\Gamma = \Sigma' \times \Sigma' \times \Sigma'$, pairs from $\Gamma \times \{1, 2, 3\}$ and a constant $o''$. We consider here three groups of rewriting rules

1. Rules that serve to ascertain the proper form of configurations from the position $n+2$ to the end of the tape where $n$ is the TM head's position. These rules have the form

   $$a \to \langle b, o\rangle \rightsquigarrow \langle a, o\rangle \qquad \text{and} \qquad a \to o \rightsquigarrow \langle a, o\rangle$$

   for all $a, b \in \Sigma'$. Note that each configuration suffix eventually reduces to some symbol $\langle a, o\rangle$ for some $a \in \Sigma'$.

2. Rules that serve to ascertain that our assumptions concerning variables are met. These rules have the form

   $$\begin{array}{l}
   \mathsf{s} \to \langle b, o\rangle \rightsquigarrow \langle \mathsf{s}, 1\rangle \text{ in case } \pi_3(\mathsf{s}) = b, \qquad \mathsf{s} \rightsquigarrow \langle \mathsf{s}, 2\rangle \text{ in case } \pi_3(\mathsf{s}) = \bullet, \text{ or } \pi_2(\mathsf{s}) = \bullet, \\
   \mathsf{s} \to \langle \mathsf{s}', 1\rangle \rightsquigarrow \langle \mathsf{s}, 2\rangle \text{ in case } \pi_3(\mathsf{s}) = \pi_2(\mathsf{s}') \text{ and } \pi_2(\mathsf{s}) = \pi_1(\mathsf{s}'), \\
   \mathsf{s} \to \langle \mathsf{s}', 2\rangle \rightsquigarrow \langle \mathsf{s}, 3\rangle \text{ in case } \pi_3(\mathsf{s}) = \pi_2(\mathsf{s}') \text{ and } \pi_2(\mathsf{s}) = \pi_1(\mathsf{s}'), \\
   a \to \langle \mathsf{s}, 3\rangle \rightsquigarrow \langle a, 4\rangle \text{ in case } \pi_1(\mathsf{s}) = a, \qquad a \to \langle b, 4\rangle \rightsquigarrow \langle a, 4\rangle.
   \end{array}$$

   We assume above that $\mathsf{s}, \mathsf{s}' \in \Sigma' \times \Sigma' \times \Sigma'$, and $a, b \in \Sigma'$. Note that the rules that involve $1, 2, 3$ can be applied only once.

3. The rules above served to ascertain that configurations have proper shapes. The final rules serve to ascertain that history has proper shape. These are

$$\langle a, 4\rangle \to o' \rightsquigarrow o', \qquad \langle \mathsf{s}, 3\rangle \to o' \rightsquigarrow o', \qquad o' \to o'' \rightsquigarrow o''.$$

We assume above that $a, b \in \Sigma'$.

We can now gather all the left-hand sides of the reduction rules as expressions $D_1, \ldots, D_l$ and right-hand sides as respectively $D'_1, \ldots, D'_l$. The rewriting ascertains that for a solution $S$ of the equation (2) we have $S(\mathsf{Fs1}\ldots\mathsf{s}p\,o') \rightsquigarrow^* o'$ only provided that consistency assumptions above are met. Therefore the equation

$$(\mathsf{Fs1}\ldots\mathsf{s}p\,o') \to \mathsf{G}D'_1\ldots D'_l \doteq \mathsf{G}D_1\ldots D_l \tag{3}$$

guarantees these consistency assumptions. Note that the solution for $\mathsf{F}$ has rank 2 while the solution for $\mathsf{G}$ has rank 3.

The sketch above requires many details to be filled, but this can be done using the technique of Schubert [12] or Levy and Veanes [7].

# References


[1] Haskell Curry. Modified basic functionality in combinatory logic. *Dialectica*, 23:83–92, 1969.

[2] Boris Düdder, Moritz Martens, Jakob Rehof, and Pawel Urzyczyn. Bounded combinatory logic. In Patrick Cégielski and Arnaud Durand, editors, *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, volume 16 of *LIPIcs*, pages 243–258. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.

[3] Ken-etsu Fujita and Aleksy Schubert. The undecidability of type related problems in the type-free style system F with finitely stratified polymorphic types. *Inf. Comput.*, 218:69–87, 2012.

[4] Paola Giannini and Simona Ronchi Della Rocca. Type inference in polymorphic type discipline. In Takayasu Ito and Albert R. Meyer, editors, *Theoretical Aspects of Computer Software, International Conference TACS '91, Sendai, Japan, September 24-27, 1991, Proceedings*, volume 526 of *Lecture Notes in Computer Science*, pages 18–37. Springer, 1991.

[5] J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmetique d'ordre supérieur*. PhD thesis, Université Paris VII, 1972.

[6] J. Roger Hindley. The principal type-scheme of an object in combinatory logic. *Transactions of the American Mathematical Society*, 146:29–60, Dec. 1969.

[7] Jordi Levy and Margus Veanes. On the undecidability of second-order unification. *Inf. Comput.*, 159(1-2):125–150, 2000.

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

[9] J. Morris. *Lambda-Calculus Models of Programming Languages*. PhD thesis, MIT, 1968.

[10] F. Pfenning. On the undecidability of partial polymorphic type reconstruction. *Fundamenta Informaticae*, 19(1,2):185–199, September-October 1993.

[11] J. C. Reynolds. Towards a theory of type structure. In Bernard J. Robinet, editor, *Programming Symposium, Proceedings Colloque sur la Programmation, Paris, France, April 9-11, 1974*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425. Springer, 1974.

[12] A. Schubert. Second-order unification and type inference for Church-style polymorphism. In *Proc. of POPL*, 1998.