Decidable structures between Church-style and Curry-style

Ken-etsu Fujita[1] and Aleksy Schubert[2]

1 **Gunma University**
Tenjin-cho 1-5-1, Kiryu 376-8515, Japan
fujita@cs.gunma-u.ac.jp
2 **The University of Warsaw**
ul. Banacha 2, 02-097 Warsaw, Poland
alx@mimuw.edu.pl

Abstract

It is well-known that the type-checking and type-inference problems are undecidable for second order λ-calculus in Curry-style, although those for Church-style are decidable. What causes the differences in decidability and undecidability on the problems? We examine crucial conditions on terms for the (un)decidability property from the viewpoint of partially typed terms, and what kinds of type annotations are essential for (un)decidability of type-related problems. It is revealed that there exists an intermediate structure of second order λ-terms, called a style of hole-application, between Church-style and Curry-style, such that the type-related problems are decidable under the structure. We also extend this idea to the omega-order polymorphic calculus F_ω, and show that the type-checking and type-inference problems then become undecidable.

1998 ACM Subject Classification D.3.1 Formal Definitions and Theory, F.4.1 Mathematical Logic

Keywords and phrases 2nd-order λ-calculus, type-checking, type-inference, Church-style and Curry-style

Digital Object Identifier 10.4230/LIPIcs.RTA.2013.190

1 Introduction

Traditionally, following the fathers [6, 7], we have two styles of λ-terms with types [2], Church-style[1] and Curry-style. Terms in the style of Church contain full type annotation, so that this style enjoys uniqueness of typing derivations. On the other hand, terms in the style of Curry are the same as those of the type free λ-calculus, and a type inference algorithm may compute their types.

The two styles give no distinction to solvability of type-related problems of simply typed λ-calculus. However, in the case of second order λ-calculus (Girard and Reynolds), it is well-known that the type checking and type inference problems are decidable for Church-style[2] but undecidable for Curry-style [32]. The two definitions of λ-terms are so different, and our motivation behind this work is to make it clear what is a crucial condition on terms for the (un)decidable property of the problems.

[1] The terminology, Church-style terms here are also called pseudo-terms *á la* de Bruijn in the recent literature [3].

[2] The problems are, in general, decidable for normalizing PTS (Pure Type Systems) with a finite set of sorts [31].

© Ken-etsu Fujita and Aleksy Schubert;
licensed under Creative Commons License CC-BY

24th International Conference on Rewriting Techniques and Applications (RTA'13).
Editor: Femke van Raamsdonk; pp. 190–205

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Decidability of TCP, TIP, and TPP for $\lambda 2$-terms with intermediate styles.

Styles	TCP		TIP		TPP
Church	Yes	$\hookrightarrow$	Yes	$\hookrightarrow$	No [30]
Hole-application	*Yes*	$\longleftrightarrow$	*Yes*	$\hookrightarrow$	*No*
Domain-free	No [11]	$\longleftrightarrow$	No [11]	$\longleftrightarrow$	No [24]
Type-free	No [13]	$\longleftrightarrow$	No [13]	$\hookleftarrow$	No [13]
Curry	No [32]	$\longleftrightarrow$	No [32]	$\hookleftarrow$	No [32]

For this principal objective, we introduce three intermediate structures called domain-free, hole-application, and type-free, see Table 1, between Church-style and Curry-style based on the previous work [11]. In the table, TCP denotes type checking, TIP type inference, and TPP typability problems, respectively. "Yes" means that a corresponding problem is decidable, and "No" undecidable. Arrows ($\hookrightarrow, \longleftrightarrow, \hookleftarrow$) denote reduction relations between problems following forthcoming Proposition 4. Our idea on the intermediate structures is quite natural from the viewpoints of type erasure mapping and partially typed terms. Terms in the style of hole-application contain domains of λ-abstraction $\lambda x\!:\!A.M$ just like Church-style, but omit the information on a polymorphic instance such as $M[]$ instead of $M[A]$. On the other hand, terms in the style of domain-free contain the information on a polymorphic instance $M[A]$ like Church-style, but omit domains of λ-abstraction such as $\lambda x.M$ rather than $\lambda x\!:\!A.M$. Terms in the style of type-free contain no type information at all like Curry-style, but contain information holders [] to be filled with a type. We will introduce an order on the styles via type erasure mappings, and in terms of the intermediate structures, we will identify the boundaries between decidability and undecidability with respect to the type-related problems, see also Figure 1 in the next section.

The partial type reconstruction problem can be regarded as a type inference problem for mixed styles of the intermediate structures. Following Boehm [5] and Pfenning [26, 27], partial type reconstruction is in general undecidable. From the viewpoint of partially typed terms, the intermediate structures including Church and Curry-styles can be regarded as a unifying framework, under which various systems can be compared comfortably.

Our work concerns both theoretical and practical aspects of programming. From the perspective of designing programming languages, we investigate a trade-off between decidability for type-related problems and comfortable programming with less annotations (overheads) in terms. This paper makes the following particularly theoretical contributions (i, ii, iii).

It is proved that (i) TIP is decidable (*Yes* in Table 1), but (ii) TPP is undecidable (*No* in Table 1) for hole-application $\lambda 2$. Hence, compared with Church-style, type inference problems remain decidable even after deleting polymorphic instance information $M[]$ from $M[A]$, but deleting a polymorphic domain $\lambda x.M$ from $\lambda x\!:\!A.M$ makes the problems undecidable. The introduction of hole-application reveals that putting polymorphic domains on terms is very important to design systems with decidable type inference. Notably, the annotation of function signatures with types is used in main-stream languages such as C or Java, so this annotational overhead seems to be acceptable for the community of programmers.

Finally, we extend this idea to the omega-order polymorphic calculus F_ω, and then show that (iii) TCP and TIP for hole-application F_ω become undecidable.

This paper is organized as follows. We introduce the second order λ-calculus $\lambda 2$ in five styles and basic definitions, and show fundamental properties of the system in Section 2. Section 3 demonstrates that the typability problem for hole-application $\lambda 2$ is undecidable. Next, a type inference algorithm for hole-application $\lambda 2$ is provided, and we prove that the

algorithm is sound and complete. Then, the subject reduction property for hole-application $\lambda 2$ is proved. Section 4 handles hole-application F_ω. Section 5 summarizes results for $\lambda 2$ in five styles (Table 1), concluding remarks, and related work.

2 Second-order lambda-calculus $\lambda 2$ in five styles

2.1 Church-style and Curry-style $\lambda 2$

We introduce the second-order lambda-calculus $\lambda 2$ (Girard and Reynolds) in the styles of Church and Curry, respectively. Types, λ-terms for each style, and inference rules are usually defined as follows:

▶ **Definition 1** ($\lambda 2$ in Church-style and Curry-style)**.**

- $\lambda 2$-types:

$$A ::= X \mid (A \to A) \mid \forall X.A$$

- $\lambda 2$-terms in Church-style:

$$M ::= x \mid (\lambda x\!:\!A.M) \mid (MM) \mid (\Lambda X.M) \mid (M[A])$$

- Contexts:
  A context denoted by Γ or Σ is a set of a declaration of the form $x : A$ with distinct variables as subjects. We write $\Gamma(x) = A$ for $x\!:\!A \in \Gamma$ and $\mathrm{dom}(\Gamma)$ for $\{x \mid x\!:\!A \in \Gamma\}$.
- Inference rules for Church-style:

$$\frac{}{\Gamma, x\!:\!A \vdash_{\mathrm{Ch}} x : A}\ (\mathrm{var})$$

$$\frac{\Gamma, x\!:\!A_1 \vdash_{\mathrm{Ch}} M : A_2}{\Gamma \vdash_{\mathrm{Ch}} \lambda x\!:\!A_1.M : A_1 \to A_2}\ (\to I) \qquad \frac{\Gamma \vdash_{\mathrm{Ch}} M_1 : A_1 \to A_2 \quad \Gamma \vdash_{\mathrm{Ch}} M_2 : A_1}{\Gamma \vdash_{\mathrm{Ch}} M_1 M_2 : A_2}\ (\to E)$$

$$\frac{\Gamma \vdash_{\mathrm{Ch}} M : A}{\Gamma \vdash_{\mathrm{Ch}} \Lambda X.M : \forall X.A}\ (\forall I)^\star \qquad \frac{\Gamma \vdash_{\mathrm{Ch}} M : \forall X.A}{\Gamma \vdash_{\mathrm{Ch}} M[A_1] : A[X := A_1]}\ (\forall E)$$

  where $(\forall I)^\star$ denotes that the eigenvariable condition $X \notin \mathrm{FV}(\Gamma)$ is imposed on the application, such that X never appears free in Γ.
- $\lambda 2$-terms in Curry-style:

$$M ::= x \mid (\lambda x.M) \mid (MM)$$

  Inference rules for Curry-style:

$$\frac{}{\Gamma, x\!:\!A \vdash_{\mathrm{Cu}} x : A}\ (\mathrm{var})$$

$$\frac{\Gamma, x\!:\!A_1 \vdash_{\mathrm{Cu}} M : A_2}{\Gamma \vdash_{\mathrm{Cu}} \lambda x.M : A_1 \to A_2}\ (\to I) \qquad \frac{\Gamma \vdash_{\mathrm{Cu}} M_1 : A_1 \to A_2 \quad \Gamma \vdash_{\mathrm{Cu}} M_2 : A_1}{\Gamma \vdash_{\mathrm{Cu}} M_1 M_2 : A_2}\ (\to E)$$

$$\frac{\Gamma \vdash_{\mathrm{Cu}} M : A}{\Gamma \vdash_{\mathrm{Cu}} M : \forall X.A}\ (\forall I)^\star \qquad \frac{\Gamma \vdash_{\mathrm{Cu}} M : \forall X.A}{\Gamma \vdash_{\mathrm{Cu}} M : A[X := A_1]}\ (\forall E)$$

  where $(\forall I)^\star$ denotes the eigenvariable condition $X \notin \mathrm{FV}(\Gamma)$.

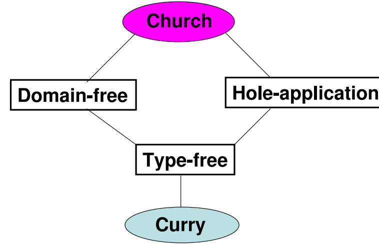

Figure 1 λ2-terms with intermediate structures between Curry-style and Church-style.

2.2 Intermediate structures between Church and Curry

Next we define λ-terms with intermediate structures [11] between Church-style and Curry-style, which are called domain-free, hole-application, and type-free. We simply write Ch, Df, Ha, Tf, and Cu, respectively, for the styles, and we employ the terminology λ-terms in s-style for each $s \in \{\mathrm{Ch}, \mathrm{Df}, \mathrm{Ha}, \mathrm{Tf}, \mathrm{Ch}\}$.

▶ **Definition 2** (Domain-free, hole-application, and type-free).

- Domain-free style λ2-terms:

$$M ::= x \mid (\lambda x.M) \mid (MM) \mid (\Lambda X.M) \mid (M[A])$$

- Hole-application style λ2-terms:

$$M ::= x \mid (\lambda x\!:\!A.M) \mid (MM) \mid (\Lambda X.M) \mid (M[\,])$$

- Type-free style λ2-terms:

$$M ::= x \mid (\lambda x.M) \mid (MM) \mid (\Lambda.M) \mid (M[\,])$$

Inference rules for domain-free, hole-application, and type-free styles, respectively, are defined similarly.

Based on the Curry-Howard isomorphism [17], well-typed λ2-terms play the role of codes for proofs. From the viewpoint of proof codes, well-typed λ-terms contain three kinds of information on proofs: (i) what inference rule is applied, (ii) where it is applied, and (iii) how to instantiate a rule. A mapping, which erases some of above information one by one from Church-style provides more abstract λ-terms with an intermediate structure. We examine what kind of information on λ-terms is the most essential for (un)decidability of type-related problems.

▶ **Definition 3** (Order on styles and erasure mapping). We define an order on the styles, see Figure 1, such that Cu $<$ Tf $<$ Df $<$ Ch and Tf $<$ Ha $<$ Ch. For styles $s, t \in \{\mathrm{Cu}, \mathrm{Tf}, \mathrm{Ha}, \mathrm{Df}, \mathrm{Ch}\}$ with $s < t$, an erasure $|\cdot|^t_s$ is defined naturally as a function from t-style λ2-terms to s-style λ2-terms as follows:

- $|x|^{\mathrm{Ch}}_{\mathrm{Df}} = x$, $|\lambda x\!:\!A.M|^{\mathrm{Ch}}_{\mathrm{Df}} = \lambda x.|M|^{\mathrm{Ch}}_{\mathrm{Df}}$, $|M_1M_2|^{\mathrm{Ch}}_{\mathrm{Df}} = |M_1|^{\mathrm{Ch}}_{\mathrm{Df}}|M_2|^{\mathrm{Ch}}_{\mathrm{Df}}$, $|\Lambda X.M|^{\mathrm{Ch}}_{\mathrm{Df}} = \Lambda X.|M|^{\mathrm{Ch}}_{\mathrm{Df}}$, $|M[A]|^{\mathrm{Ch}}_{\mathrm{Df}} = |M|^{\mathrm{Ch}}_{\mathrm{Df}}[A]$; and similarly defined for the rest cases.

2.3 Basic properties of the systems

▶ Proposition 1 (Uniqueness of types for Church-style).
If $\Gamma \vdash_{\mathrm{Ch}} M : A_1$ and $\Gamma \vdash_{\mathrm{Ch}} M : A_2$ then we have $A_1 \equiv A_2$.

▶ Proposition 2 (Erasure and lifting). Let $s, t \in \{\mathrm{Cu}, \mathrm{Tf}, \mathrm{Ha}, \mathrm{Df}, \mathrm{Ch}\}$ with $s < t$.
1. If $\Gamma \vdash_t M : A$ then $\Gamma \vdash_s |M|^t_s : A$.
2. If $\Gamma \vdash_s M : A$ then there exists a t-style $\lambda 2$-term N such that $|N|^t_s = M$ and $\Gamma \vdash_t N : A$.

▶ Proposition 3 (Generation lemma). Let $s \in \{\mathrm{Tf}, \mathrm{Df}, \mathrm{Ha}, \mathrm{Ch}\}$.
1. If $\Gamma \vdash_s x : A$ then $\Gamma(x) = A$.
2. If $\Gamma \vdash_s \lambda x : A_0.M : A_1$ then $\Gamma, x : A_0 \vdash_s M : A_2$ and $A_1 = (A_0 \to A_2)$ for some A_2, provided that $s \geq \mathrm{Ha}$.
3. If $\Gamma \vdash_s \lambda x.M : A_1$ then $\Gamma, x : A_0 \vdash_s M : A_2$ and $A_1 = (A_0 \to A_2)$ for some A_0, A_2, provided that $s \leq \mathrm{Df}$.
4. If $\Gamma \vdash_s M_1 M_2 : A_1$ then $\Gamma \vdash_s M_1 : A_0 \to A_1$ and $\Gamma \vdash_s M_2 : A_0$ for some A_0.
5. If $\Gamma \vdash_s \Lambda X.M : A_1$ then $\Gamma \vdash_s M : A_2$ and $A_1 = \forall X.A_2$ together with $X \notin \mathrm{FV}(\Gamma)$ for some A_2, provided that $s > \mathrm{Tf}$.
6. If $\Gamma \vdash_{\mathrm{Tf}} \Lambda.M : A_1$ then $\Gamma \vdash_{\mathrm{Tf}} M : A_2$ and $A_1 = \forall X.A_2$ together with $X \notin \mathrm{FV}(\Gamma)$ for some A_2.
7. If $\Gamma \vdash_s M[A] : A_1$ then $\Gamma \vdash_s M : \forall X.A_2$ and $A_1 = A_2[X := A]$ for some A_2, provided that $s \geq \mathrm{Df}$.
8. If $\Gamma \vdash_s M[] : A_1$ then $\Gamma \vdash_s M : \forall X.A_2$ and $A_1 = A_2[X := A]$ for some A, A_2, provided that $s \leq \mathrm{Ha}$.

Remarked that similar generation lemma holds for Curry-style $\lambda 2$, see [2, 32].

2.4 Type-related problems and relations between problems

▶ **Definition 4** (Type-related problems parameterized with styles).
1. Type checking problem of s-style terms denoted by $\mathrm{TCP}(s)$:
   Given an s-style λ-term M, a type A, and a context Γ, determine whether $\Gamma \vdash_s M : A$.
2. Type inference problem of s-style λ-terms denoted by TIP(s):
   Given an s-style λ-term M and a context Γ, determine whether $\Gamma \vdash_s M : A$ for some type A.
3. Typability problem of s-style terms denoted by TPP(s):
   Given an s-style λ-term M, determine whether $\Gamma \vdash_s M : A$ for some context Γ and type A.

We show relations between type-related problems. For instance, if $\mathrm{TCP}(s)$ is reduced to $\mathrm{TIP}(s)$, then we write $\mathrm{TCP}(s) \hookrightarrow \mathrm{TIP}(s)$ for this. We write $\mathrm{TCP}(s) \longleftrightarrow \mathrm{TIP}(s)$ for both $\mathrm{TCP}(s) \hookrightarrow \mathrm{TIP}(s)$ and $\mathrm{TIP}(s) \hookrightarrow \mathrm{TCP}(s)$.

▶ Proposition 4 (Reductions between type-related problems).
1. $\mathrm{TCP}(s) \hookrightarrow \mathrm{TIP}(s)$ for $s \in \{\mathrm{Cu}, \mathrm{Tf}, \mathrm{Df}, \mathrm{Ha}, \mathrm{Ch}\}$.
2. $\mathrm{TIP}(s) \hookrightarrow \mathrm{TCP}(s)$ for $s \in \{\mathrm{Cu}, \mathrm{Tf}, \mathrm{Df}, \mathrm{Ha}\}$.
3. $\mathrm{TIP}(s) \hookrightarrow \mathrm{TPP}(s)$ for $s \in \{\mathrm{Df}, \mathrm{Ha}, \mathrm{Ch}\}$.
4. $\mathrm{TPP}(s) \hookrightarrow \mathrm{TIP}(s)$ for $s \in \{\mathrm{Cu}, \mathrm{Tf}, \mathrm{Df}\}$.

Proof. We show only the case of 3 where $s = \mathrm{Df}$ here.
3. Let $\Gamma = \{a_1 : A_1, \ldots, a_n : A_n\}$. $\Gamma \vdash_s M : B$ for some B if and only if $\Sigma \vdash_s M_0 : B$ for some B and some Σ, where z_0, z_1, z, y, Y are fresh variables, and
   $M_0 = z_0(z_1(z[\forall X.X]))(z_1 z)\ (z[(A_1 \to \cdots \to A_n \to Y) \to Y](\lambda a_1 \ldots \lambda a_n.yM))$.

Suppose that $\Gamma \vdash_s M : B$ for some B. Then M_0 is typable under some context Σ such as $\Sigma(z) = \forall X.X$.
In turn, if M_0 is typable then type of z should be a universal type, to say, $\forall X.\mathsf{F}(X)$, where F is a second-order variable with arity 1. From consistent typability of the two occurrences of z_1, we have the following unification equation: $\mathsf{F}(\forall X.X) \doteq \forall X.\mathsf{F}(X)$. Observe that the only solution to the unification equation is $[\mathsf{F} := (X \mapsto X)]$, i.e., the identity function, which implies that type of z is $\forall X.X$. Hence, we can recover the context Γ. ◀

3 Hole-application $\lambda 2$

We show that typability problems for hole-application $\lambda 2$ are undecidable. Next, in order to show decidability of type checking and type inference problems for hole-application $\lambda 2$, we provide a sound and complete algorithm for type inference. First, inference rules for hole-application $\lambda 2$ are listed in the following:

$$\frac{}{\Gamma \vdash_{\text{hole}} x : \Gamma(x)} \ (\text{var})$$

$$\frac{\Gamma, x{:}A_1 \vdash_{\text{hole}} M : A_2}{\Gamma \vdash_{\text{hole}} \lambda x{:}A_1.M : A_1 \to A_2} \ (\to I) \qquad \frac{\Gamma \vdash_{\text{hole}} M_1 : A_1 \to A_2 \quad \Gamma \vdash_{\text{hole}} M_2 : A_1}{\Gamma \vdash_{\text{hole}} M_1 M_2 : A_2} \ (\to E)$$

$$\frac{\Gamma \vdash_{\text{hole}} M : A}{\Gamma \vdash_{\text{hole}} \Lambda X.M : \forall X.A} \ (\forall I)^\star \qquad \frac{\Gamma \vdash_{\text{hole}} M : \forall X.A}{\Gamma \vdash_{\text{hole}} M[\,] : A[X := B]} \ (\forall E)$$

3.1 TPP for hole-application $\lambda 2$

In order to show that TPP(Ha) is undecidable, we first introduce a restricted version of second-order unification, called a flat form [12], which can fit type constraints induced from hole-application terms. Then the undecidable unification problem is reduced to TPP(Ha) for hole-application $\lambda 2$. Although this reduction method is similar to that used in the previous work [12, 13, 14], we introduce the flat form and the encoding here to make the paper self-contained.

3.2 Second-order unification in flat form

We define expressions for unification problems. For this, the set of type variables is divided into three countable subsets: the set of first-order variables $\mathcal{V}_1$, the set of second-order functional variables $\mathcal{V}_2$, and the set of first-order constants $\mathcal{C}$. Then unification expressions are defined from first-order variables denoted by X and constants denoted by C, together with a binary constant $\to$ and second-order functional variables $\mathsf{F}^{(n)}A_1 \cdots A_n$ with arity n. The set of first-order expressions is denoted by $\mathcal{UE}_1$, and the expressions of first-order part are written by A, B as follows:

$A, B \in \mathcal{UE}_1 ::= X \mid C \mid (A \to B)$.

The sets of variables, constants, and sub-expressions in unification expressions are defined respectively as follows:

- $\mathrm{UVar}(X) = \{X\}$, $\mathrm{UVar}(C) = \emptyset$, $\mathrm{UVar}(A \to B) = \mathrm{UVar}(A) \cup \mathrm{UVar}(B)$, $\mathrm{UVar}(\mathsf{F}^{(n)}A_1 \cdots A_n) = \{\mathsf{F}^{(n)}\}$.
- $\mathrm{UCon}(X) = \emptyset$, $\mathrm{UCon}(C) = \{C\}$, $\mathrm{UCon}(A \to B) = \mathrm{UCon}(A) \cup \mathrm{UCon}(B)$, $\mathrm{UCon}(\mathsf{F}^{(n)}A_1 \cdots A_n) = \emptyset$.
- $\mathrm{UExp}(X) = \{X\}$, $\mathrm{UExp}(C) = \{C\}$, $\mathrm{UExp}(A \to B) = \{A \to B\} \cup \mathrm{UExp}(A) \cup \mathrm{UExp}(B)$, $\mathrm{UExp}(\mathsf{F}^{(n)}A_1 \cdots A_n) = \emptyset$.

The set E of unification equations in flat form is defined as follows:
$E ::= \emptyset \mid \{A \doteq B\} \cup E$
$\mid \{\mathsf{F}X_1 \ldots X_n \doteq X \to A_1 \to \cdots \to A_n \to o, \mathsf{F}C_1 \ldots C_n \doteq X' \to C_1 \to \cdots \to C_n \to o\} \cup E$,
provided that F is a functional variable with arity $n \geq 1$, X, X', X_i are fresh 1st-order variables, $C_1, \ldots, C_n$ are new and pair wise distinct constants appeared nowhere else, o is a distinguished constant, and $\mathrm{UVar}(A_1, \ldots, A_n) = \emptyset$.

We remark that each argument of a functional variable is restricted so that they are all 1st-order variables or pair wise distinct constants. Moreover, an equation with a functional variable always consists in such a pair of two equations, and functional variables in flat form appear only in this way.

Let E be a finite set of unification equations in flat form. Then $\mathrm{UVar}(E)$, $\mathrm{UCon}(E)$, and $\mathrm{UExp}(E)$ are naturally defined as well. A substitution is a partial function from the set of variables of unification expressions $\mathcal{V}_1 \cup \mathcal{V}_2$ to $\mathcal{UE}_1 \cup \{(X_1, \ldots, X_n) \mapsto A \mid A \in \mathcal{UE}_1\}$. Let S, S_1, and S_2 range over the set of substitutions. A substitution S is naturally extended into a function S' from $\mathcal{UE}_1 \cup \mathcal{V}_2$ to $\mathcal{UE}_1 \cup \{(X_1, \ldots, X_n) \mapsto A \mid A \in \mathcal{UE}_1\}$, such that

$S'(X) = S(X)$, $S'(C) = C$, $S'(A \to B) = S'(A) \to S'(B)$, and

$S'(\mathsf{F}A_1 \ldots A_n) = B[X_1 := S'(A_1), \ldots, X_n := S'(A_n)]$,

where F is a second-order variable with arity n and $S(\mathsf{F}) = (X_1, \ldots, X_n) \mapsto B$ for $B \in \mathcal{UE}_1$. We may write simply S for S'. An instance E is solvable if there exists a substitution S such that $S(A) = S(B)$ and $S(\mathsf{F}A_1 \ldots A_n) = S(B')$ for all unification equations in E in the form of either $A \doteq B$ or $\mathsf{F}A_1 \ldots A_n \doteq B'$.

▶ Proposition 5 ([12]). The second-order unification problem in flat form is undecidable.

3.3 Reduction from flat form to TPP(Ha)

For encoding an instance of second-order unification E in flat form, we assume one-to-one mappings between unification expressions and term variables of $\lambda 2$. Based on this, we write x_A, y_A for $A \in \mathrm{UExp}(B)$ where $B \in \mathcal{UE}_1$, and $x_{\mathsf{F}}, y_{\mathsf{F}}$ for $\mathsf{F} \in \mathcal{V}_2$. In particular, the distinguished constant $o \in \mathcal{C}$ provides x_o, y_o, y_{o_2}, and so on. We write $o^k \to o$ for type $(o \to (\cdots \to (o \to o)))$ with $(k+1)$-times o. As a shorthand, we define $\lambda 2$-terms such that $M[]^{n+1} = (M[])[]^n$, $M[]^0 = M$.

▶ **Definition 5** (Encoding of unification expressions). **1.** Case E of $\emptyset$: $[\![E]\!] = x_o$

2. Case E of $\{A \doteq B\} \cup E_0$: $[\![E]\!] = y_{o_4}\ (y_A x_B)\ [\![A]\!]\ [\![B]\!]\ [\![E_0]\!]$

3. Case E of $E_f \cup E_0$, where $E_f = \{\mathsf{F}X_1 \ldots X_n \doteq B_1, \mathsf{F}C_1 \ldots C_n \doteq B_2\}$ together with $B_1 = (X \to A_1 \to \cdots \to A_n \to o)$ and $B_2 = (X' \to C_1 \to \cdots \to C_n \to o)$:

$$\begin{aligned}[\![E]\!] \ = \ & y_{o_9}\ (y_{\mathsf{F}} x_{\mathsf{F}})\ (y_{\mathsf{F}}(\Lambda Z_1 \ldots \Lambda Z_n.(\Lambda Z.\lambda z\!:\!Z.\lambda z_1\!:\!Z_1 \ldots \lambda z_n\!:\!Z_n.x_o)[])) \\ & (y_{B_1}(x_{\mathsf{F}}[]^n))\ (y_{B_1}((\Lambda Z.\lambda z\!:\!Z.\lambda z_1\!:\!A_1 \ldots \lambda z_n\!:\!A_n.x_o)[])) \\ & (y_{B_2}(x_{\mathsf{F}}[]^n))\ (y_{B_2}((\Lambda Z'.\lambda z'\!:\!Z'.\lambda z_1\!:\!C_1 \ldots \lambda z_n\!:\!C_n.x_o)[])) \\ & [\![B_1]\!]\ [\![B_2]\!]\ [\![E_0]\!]\end{aligned}$$

4. For $A \in \mathcal{UE}_1$, $[\![A]\!]$ is defined as follows:

- **a.** $[\![X]\!] = y_{o_1}(y_X x_X)$
- **b.** $[\![C]\!] = y_{o_1}(y_C x_C)$
- **c.** $[\![A \to B]\!] = y_{o_4}\ (y_B(x_{A\to B} x_A))\ (y_{A\to B} x_{A\to B})\ [\![A]\!]\ [\![B]\!]$

5. $\Sigma_\Delta = \Sigma_o \cup \Sigma(\Delta)$, where Σ_o and $\Sigma(\Delta)$ are defined as follows for $\Delta = A$ or E:

- **a.** $\Sigma_o = \{x_o : o, y_{o_1} : o \to o, y_{o_2} : o \to o \to o, \ldots, y_{o_k} : o^k \to o\}$ for $k = 9$
- **b.** $\Sigma(\Delta) = \{x_C : C, y_C : C \to o \mid C \in \mathrm{UCon}(\Delta)\}$

An idea on encoding of first-order follows the structure of expressions, such that a unification expression provides an λ-term consisting of consecutive application of variables associated to each sub-expression, which induces type constraints leading to substitutions for the unification expression. An idea on encoding of second-order is such that a term variable x_{F} associated to a functional variable F of unification should have a universal type, whose instance by application of $(\forall E)$ must be equivalent to a substitution instance of the right-hand side B of the corresponding unification equation $\mathsf{F}(\cdots) \doteq B$.

▶ **Lemma 6.** *Let $A \in \mathcal{UE}_1$, $\Sigma_A = \Sigma_o \cup \Sigma(A)$, and S be a substitution from $\mathcal{UE}_1$ to $\mathcal{UE}_1$. Then we have $\Sigma_A, \Gamma \vdash_{\text{hole}} [\![A]\!] : o$ for some context Γ such that $\Gamma(x_B) = S(B)$ and $\Gamma(y_B) = (S(B) \to o)$ for each $B \in \text{UExp}(A)$.*

Proof. We remark that Γ should declare statements for all free variables x_B, y_B in $[\![A]\!]$ where $B \in \text{UExp}(A)$. By induction on the structure of A. We show one case here.

1. Case of $A = (A_1 \to A_2)$:
   From the induction hypotheses, we have $\Sigma_{A_1}, \Gamma_1 \vdash_{\text{hole}} [\![A_1]\!] : o$ and $\Sigma_{A_2}, \Gamma_2 \vdash_{\text{hole}} [\![A_2]\!] : o$, such that $\Gamma_1(x_{B_1}) = S(B_1)$, $\Gamma_1(y_{B_1}) = S(B_1) \to o$ for each $B_1 \in \text{UExp}(A_1)$ and $\Gamma_1(x_{B_2}) = S(B_2)$, $\Gamma_1(y_{B_2}) = S(B_2) \to o$ for each $B_2 \in \text{UExp}(A_2)$. Then we can merge Γ_1 and Γ_2 into Γ so that $\Gamma(x_{A_1 \to A_2}) = S(A_1) \to S(A_2)$ and $\Gamma(y_{A_1 \to A_2}) = \Gamma(x_{A_1 \to A_2}) \to o$, since $\Gamma_1(x_B) = S(B) = \Gamma_2(x_B)$ for $B \in \text{UExp}(A_1) \cap \text{UExp}(A_2)$. Hence, We have $\Sigma_A, \Gamma \vdash_{\text{hole}} [\![A_1 \to A_2]\!] : o$ for some Γ with the desired property. ◀

▶ Proposition 6. A flat form E is solvable if and only if $\Sigma_E, \Gamma \vdash_{\text{hole}} [\![E]\!] : o$ for some Γ.

Proof. The only-if part can be verified so that Γ is given by a unifier of E. We show here the if-part. Suppose that the encoding $[\![E]\!]$ has type o under Σ_E and some context Γ. From consistent type of the encoding of first-order equations $A' \doteq B' \in E$, we have $\Gamma(x_{A'}) = \Gamma(x_{B'})$. From this and Lemma 6, we can define a substitution S for first-order variables in $\text{UVar}(E)$ such that $S(A') = \Gamma(x_{A'}) = \Gamma(x_{B'}) = S(B')$. Next, we verify a consistent type of the encoding of second-order equations. Considering the first and second arguments of y_{o_9}, the term x_{F} has type $\forall Z_1 \ldots \forall Z_n.(A \to Z_1 \to \cdots \to Z_n \to o)$ for some A. Here, we can assume that A should contain no quantifiers $\forall$, since the type A is simply related to the first argument type of x_{B_1} and x_{B_2}. Even if A contained for instance $\forall Y.B$, then one could replace this with $B[Y := Y']$ using a fixed type variable Y'. Then, from consistent type of the three occurrences of each argument of y_{B_1}, the three terms, x_{B_1}, $x_{\mathsf{F}}[]^n$, and $(\Lambda Z.\lambda z\!:\!Z.\lambda z_1\!:\!A_1 \ldots \lambda z_n\!:\!A_n.x_o)[]$, all have the same type $(A[Z_1 := A_1, \ldots, Z_n := A_n] \to A_1 \to \cdots \to A_n \to o)$. Following a similar pattern, the three arguments of y_{B_2} are also well-typed. Therefore, a flat form E becomes solvable under a substitution S such that

$S(\mathsf{F}) = (Z_1, \ldots, Z_n) \mapsto (A \to Z_1 \to \cdots \to Z_n \to o)$, $S(X_i) = A_i$ for $1 \le i \le n$,
$S(X) = A[Z_1 := A_1, \ldots, Z_n := A_n]$, and $S(X') = A[Z_1 := C_1, \ldots, Z_n := C_n]$. ◀

▶ **Theorem 7** (TPP for hole-application $\lambda 2$)**.** TPP *is undecidable for hole-application $\lambda 2$.*

Proof. A flat form E is solvable

iff $\Gamma, \Sigma_E \vdash_{\text{hole}} [\![E]\!] : o$ for some Γ by Proposition 6
iff $\Gamma \vdash_{\text{hole}} \lambda v\!:\!\forall X.(X \to o).v[](\lambda \vec{z}\!:\!\Sigma_E(\vec{z}).[\![E]\!]) : A$ for some A and some Γ.

Here, we write $\lambda \vec{z} : \Sigma_E(\vec{z}).[\![E]\!]$ for $\lambda z_1 : \Sigma_E(z_1) \ldots \lambda z_n : \Sigma_E(z_n).[\![E]\!]$ with $\{z_1, \ldots, z_n\} = \text{dom}(\Sigma_E)$. ◀

3.4 TCP and TIP for hole-application $\lambda 2$

From Proposition 4, the problems TCP(Ha) and TIP(Ha) are equivalent, and we provide a type inference algorithm $\mathsf{type}(\Gamma; M)$, which computes a type of a hole-application term M under a context Γ. The algorithm involves a unification procedure, for which we introduce

new type variables called unification variables consisting of first-order variables denoted by α, β and functional variables denoted by F. For the technical reason, the following syntax $\hat{A} \in \mathsf{Uexp}$ is defined from types and first-order unification variables:

$\hat{A} \in \mathsf{Uexp} ::= X \mid \alpha \mid (\hat{A} \to \hat{A}) \mid \forall X.\hat{A}$

A substitution denoted by S used in unification should operate only on unification variables, such that $S(\alpha), S(\mathsf{F})(\beta) \in \mathsf{Uexp}$ and $S(X) = X$, i.e., the domain of substitutions is the set of unification variables, and the range is Uexp.

▶ **Definition 8** (Type inference algorithm type)**.**

1. $\mathsf{type}(\Gamma; x) = \Gamma(x)$
2. $\mathsf{type}(\Gamma; \lambda x\!:\!A.M) = (A \to \mathsf{type}(\Gamma, x\!:\!A; M))$
3. $\mathsf{type}(\Gamma; MN) =$
   let $\hat{B}_1 = \mathsf{type}(\Gamma; M)$ and (* α is a fresh unification variable *)
   let $\hat{B}_2 = \mathsf{type}(\Gamma; N)$ and $S = \mathsf{unify}(\hat{B}_1, \hat{B}_2 \to \alpha)$ in $S(\alpha)$
4. $\mathsf{type}(\Gamma; \Lambda X.M) =$ (* S is an arbitrary substitution for unification variables *)
   let $\hat{B} = \mathsf{type}(\Gamma; M)$ and $X \notin \mathrm{FV}(\Gamma)$ in $\forall X.S(\hat{B})$
5. $\mathsf{type}(\Gamma; M[]) =$
   let $\hat{B} = \mathsf{type}(\Gamma; M)$ and $S = \mathsf{unify}(\forall X.\mathsf{F}(X), \hat{B})$ (* β is a fresh unification variable,
   in $S(\mathsf{F})(\beta)$ and F is a fresh functional unification variable with arity 1 *)

We remark that second-order unification used in the algorithm is a special case of patterns unification of Miller [23], such that arguments of a functional variable are distinct bound variables in expressions. Since unification of patterns is decidable and gives a most general unifier if unifiable [23], the unification problem such as $\mathsf{unify}(\forall X.\mathsf{F}(X), \hat{B})$ is decidable. Hence, the unification procedure returns the most general solution to the type inference problem. In this sense, `type` gives rise to a decidable sub-language of which is derived by the general translation V of Pfenning [26] in the case of the omega-order calculus F_ω.

Let $\bot \equiv \forall X.X$. We show an example of $\mathsf{type}(\langle\rangle; \Lambda Z.\lambda x\!:\!\bot.x[]x)$ in the following:

1. $\mathsf{type}(x\!:\!\bot; x[]) = (X \mapsto X)\beta = \beta$ for a fresh unification variable β,
   where $\mathsf{unify}(\forall X.\mathsf{F}(X), \bot) = [\mathsf{F} := (X \mapsto X)]$.
2. $\mathsf{type}(x\!:\!\bot; x[]x) = \alpha$ for a fresh unification variable α,
   where $\mathsf{unify}(\beta, \bot \to \alpha) = [\beta := (\bot \to \alpha)]$.
3. $\mathsf{type}(\langle\rangle; \lambda x\!:\!\bot.x[]x) = (\bot \to \mathsf{type}(x\!:\!\bot; x[]x)) = (\bot \to \alpha)$
4. $\mathsf{type}(\langle\rangle; \Lambda Z.\lambda x\!:\!\bot.x[]x) = \forall Z.\mathsf{s}(\mathsf{type}(\langle\rangle; \lambda x\!:\!\bot.x[]x)))$
   $= \forall Z.(\bot \to \mathsf{s}(\alpha))$ for an arbitrary substitution s.

In addition, we show a proof figure below, which provides a type for the term. Although the term may have yet another type, all possible types for $\Lambda Z.\lambda x\!:\!\bot.x[]x$ can be expressed by the inferred type $\mathsf{type}(\langle\rangle; \Lambda Z.\lambda x\!:\!\bot.x[]x)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{x\!:\!\bot \vdash_{\text{hole}} x : \bot}{x\!:\!\bot \vdash_{\text{hole}} x[] : \bot \to Z}\,(\forall E) \qquad x\!:\!\bot \vdash_{\text{hole}} x : \bot}{x\!:\!\bot \vdash_{\text{hole}} x[]x : Z}\,(\to E)}{\vdash_{\text{hole}} \lambda x\!:\!\bot.x[]x : \bot \to Z}\,(\to I)}{\vdash_{\text{hole}} \Lambda Z.\lambda x\!:\!\bot.x[]x : \forall Z.(\bot \to Z)}\,(\forall I)^\star$$

One of the points of the algorithm is that any type to be filled into a hole [] can be represented by a unification variable, which is handled by a decidable fragment of second-order unification. Another point is that a universal type of a term $\Lambda X.M$ should be in the form of $\forall X.S(\hat{A})$, where $\hat{A}$ is a type of M, and S is an arbitrary substitution for unification variables in $\hat{A}$. In the process of unification, such an arbitrary substitution is handled as delayed substitutions at an object level.

▶ Proposition 7 (Soundness and completeness of type).

1. If $\mathsf{type}(\Gamma; M) = \hat{A}$ then $\Gamma \vdash_{\text{hole}} M : \hat{A}$.
2. Given a context Γ and a term M, let A be a type such that $\Gamma \vdash_{\text{hole}} M : A$. Then we have $\mathsf{type}(\Gamma; M) = \hat{B}$ such that $A = S(\hat{B})$ under some substitution S for unification variables.

Proof. A type system for hole-application $\lambda 2$ to handle $\hat{A}$ can be naturally introduced, such that infer $\Gamma \vdash_{\text{hole}} M[] : A[X := \hat{B}]$ from $\Gamma \vdash_{\text{hole}} M : \forall X.A$. We claim that if $\Gamma \vdash_{\text{hole}} M : \hat{A}$ then $\Gamma \vdash_{\text{hole}} M : S(\hat{A})$ for any substitution S for unification variables.

In the following, we show some the cases here. The algorithm is proved to be sound by induction on the structure of M.

1-1. Case of $\mathsf{type}(\Gamma; M[]) = S(\mathsf{F})(\beta)$, where $S = \mathsf{unify}(\forall X.\mathsf{F}(X), \mathsf{type}(\Gamma; M))$:
From the induction hypothesis, we have $\Gamma \vdash M : \mathsf{type}(\Gamma; M)$, and then $\Gamma \vdash M : S(\mathsf{type}(\Gamma; M))$ where $S(\mathsf{type}(\Gamma; M)) = \forall X.S(\mathsf{F})(X)$. Hence, we establish that $\Gamma \vdash M[] : S(\mathsf{F})(\beta)$ where β is a fresh unification variable.

1-2. Case of $\mathsf{type}(\Gamma; \Lambda X.M) = \forall X.S(\mathsf{type}(\Gamma; M))$ for any S, where $X \notin \mathrm{FV}(\Gamma)$:
From the induction hypothesis, we have $\Gamma \vdash M : \mathsf{type}(\Gamma; M)$, and then $\Gamma \vdash M : S(\mathsf{type}(\Gamma; M))$ for any substitution S for unification variables. Hence, $\Gamma \vdash \Lambda X.M : \forall X.S(\mathsf{type}(\Gamma; M))$.

The completeness property is proved by induction on the derivation of $\Gamma \vdash_{\text{hole}} M : A$.

2-1. $\Gamma \vdash \Lambda X.M : \forall X.A$ from $\Gamma \vdash M : A$, where $X \notin \mathrm{FV}(\Gamma)$:
From the induction hypothesis, we have $\mathsf{type}(\Gamma; M) = \hat{A}_1$ where $A = S(\hat{A}_1)$ for some S. Then we confirm that $\forall X.A = \forall X.S(\hat{A}_1) = \mathsf{type}(\Gamma; \Lambda X.M)$.

2-2. $\Gamma \vdash M[] : A[X := B]$ from $\Gamma \vdash M : \forall X.A$:
From the induction hypothesis, we have $\mathsf{type}(\Gamma; M) = \hat{A}_1$ and $\forall X.A = S(\hat{A}_1)$ for some S. Then we have a unifier $S = \mathsf{unify}(\forall X.\mathsf{F}(X), \hat{A}_1)$, since $\forall X.S(\mathsf{F})(X) = S(\hat{A}_1) = \forall X.A$ where $S(\mathsf{F})(X) = A$. Hence, $A[X := B] = S(\mathsf{F})(B) = S(\mathsf{F})(\beta)[\beta := B] = S_1(\mathsf{type}(\Gamma; M[]))$ for some S_1, such that $S_1 = S \cup \{[\beta := B]\}$ where β is a fresh unification variable. ◀

3.5 Subject reduction of hole-application $\lambda 2$

We define reduction rules for hole-application terms. The idea is to introduce a fresh and distinguished type variable at each reduction of type variable abstraction. Then, from a typing derivation, we can extract a concrete type, by which the fresh type variable should be replaced.

▶ **Definition 9** (Reduction rules for hole-application $\lambda 2$).

(β) $(\lambda x{:}A.M)N \to M[x := N]$
(β_t) $(\Lambda X.M)[] \to M[X := \alpha]$ where α is a fresh type variable.

For instance, we have the following judgement: $\vdash_{\text{hole}} \Lambda Y.(\Lambda X.\lambda x{:}X.x)[] : \forall Y.((Y \to Y) \to Y \to Y)$. Then $\Lambda Y.(\Lambda X.\lambda x{:}X.x)[] \to \Lambda Y.\lambda x{:}\alpha.x$ where α is a fresh type variable. Now, from a derivation of the judgement:

$$\dfrac{\dfrac{\dfrac{\dfrac{x{:}X \vdash x : X}{\vdash \lambda x{:}X.x : X \to X}\,(\to I)}{\vdash \Lambda X.\lambda x{:}X.x : \forall X.(X \to X)}\,(\forall I)^{\star}}{\vdash (\Lambda X.\lambda x{:}X.x)[] : (Y \to Y) \to Y \to Y}\,(\forall E)}{\vdash \Lambda Y.(\Lambda X.\lambda x{:}X.x)[] : \forall Y.((Y \to Y) \to Y \to Y)}\,(\forall I)^{\star}$$

a replacement for α can be extracted such that $R(\alpha) = (Y \to Y)$. Note that this replacement should not be called a substitution, since free Y in $R(\alpha)$ is to be in the scope of ΛY of the example $\Lambda Y.\lambda x\!:\!R(\alpha).x$.

▶ Proposition 8 (Subject reduction). If $\Gamma \vdash_{\text{hole}} M : A$ and $M \to N$, then $\Gamma \vdash_{\text{hole}} R(N) : A$ for some replacement R for fresh variables.

Proof. By induction on the derivation $M \to N$. We show some of the interesting cases.

1. $\Gamma \vdash (\Lambda X.M)[] : A$ and $(\Lambda X.M)[] \to M[X := \alpha]$:
   From the generation lemma, we have $\Gamma \vdash \Lambda X.M : \forall X.A'$ where $A = A'[X := B]$ and $\Gamma \vdash M : A'$ where $X \notin \mathrm{FV}(\Gamma)$ for some A', B. Then we have $\Gamma \vdash M[X := B] : A'[X := B]$. Hence, $\Gamma \vdash R(M[X := \alpha]) : A'[X := B]$ for some replacement R such that $R(\alpha) = B$.
2. $\Gamma \vdash M[] : A$ and $M[] \to M_1[]$:
   From the generation lemma, we have $\Gamma \vdash M : \forall X.A'$ with $A = A'[X := B]$ for some A', B. From the induction hypothesis w.r.t. $\Gamma \vdash M : \forall X.A'$ and $M \to M_1$, we have $\Gamma \vdash R(M_1) : \forall X.A'$ for some R, and hence $\Gamma \vdash R(M_1)[] : A'[X := B]$. ◀

Finally, we extend the idea of hole-application to an omega-order system F_ω.

4 Hole-application F_ω

We introduce a formal system of hole-application F_ω. The system consists of kinds K, type constructors A, hole-application terms M, and contexts Γ. For a kind K, an order $\mathsf{ord}(K)$ is defined, such that $\mathsf{ord}(\star) = 2$ and $\mathsf{ord}(K_1 \to \cdots \to K_n \to \star) = \max\{\mathsf{ord}(K_i) \mid 1 \leq i \leq n\} + 1$. A fragment of F_ω restricted to $K = \star$, i.e., $\mathsf{ord}(K) = 2$, coincides with $\lambda 2$.

Compared with hole-application $\lambda 2$, hole-application F_ω has a hole $[]_K$ with a kind K, which is to be filled with a type constructor of kind K, see the inference rule (ΠE) in the following. Such a hole has already been introduced in Pfenning [26].

▶ **Definition 10** (Hole-application F_ω).

1. Kinds
$$K ::= \star \mid (K \to K)$$
2. Type constructors
$$A ::= X \mid (A \to A) \mid \Pi X\!:\!K.A \mid \Lambda X\!:\!K.A \mid AA$$
3. Hole-application terms
$$M ::= x \mid \lambda x\!:\!A.M \mid MM \mid \Lambda X\!:\!K.M \mid M[]_K$$
4. Contexts
$$\Gamma ::= \langle\rangle \mid X\!:\!K, \Gamma \mid x\!:\!A, \Gamma$$

Next we define inference rules for well-formed contexts, well-formed kinds, well-formed elements of a kind, and well-formed elements of a type, respectively. Here, we show rules only for well-formed elements of a type.

1. Well-formed elements of a type:
$$\frac{\vdash \Gamma \quad x\!:\!A \in \Gamma}{\Gamma \vdash x : A}\ (\text{var})$$
$$\frac{\Gamma \vdash A_1 : \star \quad \Gamma, x\!:\!A \vdash M : A_2}{\Gamma \vdash (\lambda x\!:\!A_1.M) : (A_1 \to A_2)}\ (\to I) \qquad \frac{\Gamma \vdash M_1 : (A_1 \to A_2) \quad \Gamma \vdash M_2 : A_1}{\Gamma \vdash M_1 M_2 : A_2}\ (\to E)$$

$$\frac{\Gamma \vdash K \quad \Gamma, X\!:\!K \vdash M : A}{\Gamma \vdash (\Lambda X\!:\!K.M) : (\Pi X\!:\!K.A)}\ (\Pi I) \qquad \frac{\Gamma \vdash M : (\Pi X\!:\!K.A_1) \quad \Gamma \vdash A_2 : K}{\Gamma \vdash M[]_K : A_1[X := A_2]}\ (\Pi E)$$

$$\frac{\Gamma \vdash M : A_1 \quad \Gamma \vdash A_2 : \star \quad A_1 =_{\beta\eta} A_2}{\Gamma \vdash M : A_2}\ (\text{conv})$$

Even in the case of omega-order, the two problems TCP(Ha) and TIP(Ha) are equivalent each other as proved by Proposition 4. Toward type inference for hole-application F_ω, type constructors are extended with fresh type variables called unification variables denoted by $\alpha, \beta, \mathsf{F}, \mathsf{G}$, as follows:

$\hat{A} ::= X \mid \alpha \mid (\hat{A} \to \hat{A}) \mid \Pi X\!:\!K.\hat{A} \mid \Lambda X\!:\!K.\hat{A} \mid \hat{A}\hat{A}$.

Here, we show that TIP and TCP for hole-application F_ω are undecidable. For this, we give a reduction from higher-order unification [16, 15] to TCP for hole-application F_ω.

The theory of simply type λ-calculus is defined as usual, but in terms of type constructors of F_ω. Here, we assume the following variable conventions: F, G for free variables, and X, Y, Z for constants or bound variables. In addition, $\star$ stands for an atomic type.

- Terms
$$t, s ::= X \mid \mathsf{F} \mid \Lambda X.t \mid (t\ s)$$
- Types
$$K ::= \star \mid (K \to K)$$

Given a well-typed term t of simply typed λ-calculus, then define a type constructor $t^\sharp$ of hole-application F_ω as follows, where a free variable F will be interpreted as a bound variable such as $\Pi\mathsf{F}\!:\!K.(\cdots)$ in F_ω.

1. $X^\sharp = X$
2. $\mathsf{F}^\sharp = \mathsf{F}$
3. $(\Lambda X.t)^\sharp = \Lambda X^\sharp\!:\!K.t^\sharp$, where X has type K
4. $(t\ s)^\sharp = t^\sharp\ s^\sharp$

Given an instance $s \doteq t$ of higher-order unification, where $\{\mathsf{F}_1 : L_1, \ldots, \mathsf{F}_n : L_n\} = \mathrm{FV}(s)$ together with type L_i for each free variable F_i in s, $\{\mathsf{G}_1 : L'_1, \ldots, \mathsf{G}_m : L'_m\} = \mathrm{FV}(t)$ with type L'_i for each free variable G_i in t, and the terms s and t both have type $(K_1 \to \cdots \to K_p \to \star)$. Then define a context $\Gamma_{s=t}$ of hole-application F_ω as follows:

$\{X_1 : K_1, \ldots, X_p : K_p, Z : \star,$
$\quad x_s : (\Pi\mathsf{F}_1\!:\!L_1 \ldots \Pi\mathsf{F}_n\!:\!L_n.(s^\sharp X_1 \ldots X_p \to Z)),\ x_t : (\Pi\mathsf{G}_1\!:\!L'_1 \ldots \Pi\mathsf{G}_m\!:\!L'_m.t^\sharp X_1 \ldots X_p)\}$

▶ Proposition 9 (TCP(hole-F_ω)). An instance of higher-order unification $s \doteq t$ is solvable if and only if $\Gamma_{s=t} \vdash x_s[]^n_{\vec{L}}(x_t[]^m_{\vec{L'}}) : Z$ in hole-application F_ω.

Proof. We show the if-part here. Suppose that $\Gamma_{s=t} \vdash x_s[]^n_{\vec{L}}(x_t[]^m_{\vec{L'}}) : Z$ in hole-application F_ω. Then we have the following judgements by a chain of applications of (ΠE):

$\Gamma_{s=t} \vdash x_s[]^n_{\vec{L}} : (s^\sharp X_1 \ldots X_p \to Z)[\mathsf{F}_1 := \alpha_1, \ldots, \mathsf{F}_n := \alpha_n]$, and
$\Gamma_{s=t} \vdash x_t[]^m_{\vec{L'}} : t^\sharp X_1 \ldots X_p[\mathsf{G}_1 := \beta_1, \ldots, \mathsf{G}_m := \beta_m]$,

where α_i, β_j are fresh type variables called unification variables with appropriate kinds. From consistent type of $x_s[]^n_{\vec{L}}(x_t[]^m_{\vec{L'}})$ under $\Gamma_{s=t}$, there exists a unifier for the unification equation:

$(s^\sharp X_1 \ldots X_p \to Z)[\mathsf{F}_1 := \alpha_1, \ldots, \mathsf{F}_n := \alpha_n] \doteq t^\sharp X_1 \ldots X_p[\mathsf{G}_1 := \beta_1, \ldots, \mathsf{G}_m := \beta_m] \to \alpha$,

where α is a unification variable with kind $\star$. That is, the following equation is solvable:

$s^\sharp X_1 \ldots X_p[\mathsf{F}_1 := \alpha_1, \ldots, \mathsf{F}_n := \alpha_n] \doteq t^\sharp X_1 \ldots X_p[\mathsf{G}_1 := \beta_1, \ldots, \mathsf{G}_m := \beta_m]$.

Hence, $s^\sharp \doteq t^\sharp$ is unifiable, and then exactly so is $s \doteq t$. ◀

▶ **Theorem 11** (TCP(hole-F_ω),TIP(hole-F_ω)). TCP(hole-F_ω) *and* TIP (hole-F_ω) *are equivalent and undecidable.*

Proof. From Propositions 4 and 9. ◀

Remarked that the use of kind-labels annotated to holes is not essential in the proof for undecidability. Since the context has $x_s : (\Pi\mathsf{F}_1 : L_1 \ldots \Pi\mathsf{F}_n : L_n.(s^\sharp X_1 \ldots X_n \to Z))$, we can apply (ΠE) to A only with kind L_1. In the next section, we will observe that another kind of labels is essential for undecidability of TCP(hole-$\lambda 2$) and TIP(hole-$\lambda 2$), contrary to this case of omega-order.

5 Concluding remarks and summary of results

The type-related problems (TCP, TIP, TPP) have been studied extensively from various viewpoints, e.g., type ranks [19, 20, 11], type levels [21, 13, 14], partially typed terms [5, 26, 27, 11]. Here, we discussed mainly from a perspective of type erasure mapping. We have examined three intermediate $\lambda 2$-terms between Church-style and Curry-style. In particular, TCP and TIP for hole-application $\lambda 2$-terms turn out to be decidable, providing a type inference algorithm that is sound and complete. The algorithm involves two important features: one is decidable second-order unification, which is a special case of patterns unification [23, 8], and another is delayed substitutions, which are employed to denote arbitrary substitutions at an object level. On the other hand, TPP(Ha) is undecidable for hole-application $\lambda 2$.

Next, we extended the idea of hole-application to F_ω, and proved that TCP and TIP then become undecidable for the system. Strictly speaking, the problems are undecidable for F_3 from undecidability of second-order unification [15].

We summarize the results on the type-related problems for $\lambda 2$. Table 1 shows the decidability results for $\lambda 2$ and relations on the type-related problems. Reduction relations $(\hookrightarrow, \longleftrightarrow, \hookleftarrow)$ between problems follow Proposition 4. To our knowledge, it is a new result that TCP, TIP, and TPP are all equivalent in the case of domain-free, which implies a corollary such that typability of domain-free $\lambda 2$ is undecidable [24]. While the table shows that TPP is undecidable for any style, TCP and TIP have the boundaries between hole-application and domain-free. Compared with Church-style, TIP remains decidable even after deleting polymorphic instance information on application of $(\forall E)$. However, on application of $(\to I)$, deleting polymorphic domains makes TIP undecidable. Following [11], finding out deleted polymorphic domains is to find a polymorphic context, which leads to undecidable unification (simple instances). Therefore, the introduction of hole-application reveals that polymorphic domains are considered as the most essential information for (un)decidable TIP.

We make some observations on our results from the viewpoint of partially typed terms.

Partial type reconstruction

Partially typed terms (preterms) are defined as follows:

$P ::= x \mid \lambda x{:}A.P \mid PP \mid \Lambda X.P \mid P[A] \mid \lambda x.P \mid P[\,]$

The problem of partial type reconstruction is a problem: given a context Γ and a preterm P, determine whether there exists a term M in Church-style such that $\Gamma \vdash_{\mathrm{Ch}} M : A$ and $|M| = P$ for some A. The problem has been studied extensively and proved to be, in general, undecidable by Boehm [5] and Pfenning [27]. Moreover, Pfenning [26] shows the precise correspondence such that the problem in the n-th order λ-calculus is equivalent to n-th order unification that is undecidable in general for $n \geq 2$. Along this line, partial type reconstruction problems for s-style terms can be defined naturally. Then TIP(s) (type inference for s-style terms) is equivalent to partial type reconstruction for s-style terms.

Some of intermediate structures, e.g., domain-free and type-free, are already known and investigated.

Domain-free style

Pure Type Systems [2] in domain-free style were studied in detail in Barthe and Sørensen [4]. Domain-free systems serve as a good source language for CPS-translation. For instance, domain-free $\lambda 2$ and $\lambda^{\exists}$ are demonstrated in [10]. Parigot's $\lambda\mu$-calculus [25] in domain-free style is investigated in [9] for call-by-value second-order language with control operators.

Type-free style

The type reconstruction problem for type-free style $\lambda 2$ was described in Pfenning [27] as an instance for terms completely devoid of types except for $[]$ and Λ, and this restricted problem had been open. Recently, a negative answer to the problem is proved in [13]. The type-free style gives a compact proof description, such that this style contains the complete information on which and where inference rules are applied *á la* Church-style, but no information on what types are involved in the rules *á la* Curry-style.

Partially typed terms with labels

Another interesting variant of $\lambda 2$-terms is partially typed terms together with labels [11]. Labels denoted by a are introduced into preterms as follows:

$P ::= x \mid \lambda x\!:\![A]^a.P \mid PP \mid \Lambda X.P \mid P[A]^a \mid \lambda x\!:\![]^a.P \mid P[]^a$

Here, the placeholder $[]^a$ indicates that a type has been erased, and moreover, the label a in $[]^a$ will be used to identify the occurrences of $[]$, i.e., the holes $[]$ with the same label should be obtained by erasing the same type.

Preterms with no labels can be translated to preterms with labels using fresh ones, such that $\lfloor \lambda x.M \rfloor = \lambda x\!:\![]^a.\lfloor M \rfloor$ for a fresh label a and $\lfloor M[] \rfloor = \lfloor M \rfloor []^a$ for a fresh label a. Hence, $\Gamma \vdash P : A$ without labels iff $\Gamma \vdash \lfloor P \rfloor : A$ with labels, which implies that the type-related problems of preterms can be embedded into those with labels. For instance, the type-related problems of domain-free style with labels:

$M ::= x \mid \lambda x^a.M \mid MM \mid \Lambda X.M \mid M[A]$,

and the problems of type-free style with labels:

$M ::= x \mid \lambda x^a.M \mid MM \mid \Lambda.M \mid M[]^a$

are also undecidable. In addition, TPP of hole-application style with labels:

$M ::= x \mid \lambda x\!:\!A.M \mid MM \mid \Lambda X.M \mid M[]^a$

is undecidable by a reduction from TPP for Church-style [30] without labels, as follows:

$\Gamma \vdash_{\mathrm{Ch}} M : A$ for some Γ and some A if and only if $\Gamma \vdash_{\mathrm{hole}^a} \lceil M \rceil : A$ for some Γ and some A, where $\lceil M[A] \rceil = (\lambda v\!:\!(A \to A).\lceil M \rceil []^a)(\lambda x\!:\!A.(\Lambda X.\lambda y\!:\!X.y)[]^a x)$.

Although TCP and TIP for hole-application without labels are to be decidable in this paper, the two problems with labels become undecidable by a reduction from TPP of hole-application with labels, as follows: Let $\{x_1, \dots, x_n\} = \mathrm{FV}(M)$.

$\Gamma \vdash_{\mathrm{hole}^a} M : A$ for some Γ, A iff $z_1 : \forall X.X, \dots, z_n : \forall X.X \vdash_{\mathrm{hole}^a} \langle M \rangle : A$ for some A, where $\langle x_i \rangle = z_i []^{a_i}$ for a fresh variable z_i and a fresh label a_i.

Hence, the type checking problem for hole-application with labels becomes undecidable by Proposition 4. These observations mean the use of labels for hole-application $\lambda 2$ is essential for undecidability of TCP(hole-$\lambda 2$) and TIP(hole-$\lambda 2$), contrary to the use of kind-labels in TCP(hole-F_ω) and TIP(hole-F_ω) in the previous section.

Related work (Scrap type applications [18] and ML^{F} [22, 28])

From the viewpoint of compiler writers, Jay and Peyton Jones [18] introduced implicit System F, called System IF. System IF allows redundant type arguments of $M[A]$ to be implicit such as M with no placeholders, whereas a scrapped argument A can be recovered via matching.

Our principal objective on this work is to find out an essential type annotation that governs (un)decidability of type-related problems. Compared with System IF, hole-application terms still have placeholders [], where our type inference mechanism is based on a decidable fragment of second-order unification. The detailed comparison must be interesting and should be given somewhere for practical application.

Le Botlan and Rémy [22] introduced a type system ML^{F}, by extending ML with full polymorphism as in System F. The language ML^{F} has a family of systems, and Rémy and Yakobowski [28] presented a Church-style version $x\mathrm{ML}^{\mathrm{F}}$ with full type information. As a generalization of a polymorphic type $\forall\alpha.\tau$ of System F, a significant feature of ML^{F} is a flexible quantification $\forall(\alpha \geq \sigma)\tau$, where type variables intuitively range over instances of σ. Accordingly, type abstractions are extended such as $\Lambda(\alpha \geq \sigma)a$. Moreover, as a generalization of type application, $x\mathrm{ML}^{\mathrm{F}}$ uses type instantiation $a\phi$, such that $\Gamma \vdash a\phi : \tau_2$ if $\Gamma \vdash a : \tau_1$ and $\Gamma \vdash \phi : \tau_1 \leq \tau_2$. Here, intuitively the instantiation ϕ transforms the type τ_1 of a into another type τ_2 that is an instance of τ_1. In order to handle instantiation formally, besides typing rules and β-reductions as usual, they introduced type instance rules, type instantiation on types, and reduction rules for terms with instantiations. In this way, ML^{F} established the powerful expressiveness successfully. Although the idea of hole-application is orthogonal, our work also proceeds from the same motivation as theirs using type annotations, still under the traditional framework.

Acknowledgements

We would like to thank deeply all the referees for their careful reading and constructive comments.

References

1. H. P. Barendregt: *The lambda Calculus. Its Syntax and Semantics*, North-Holland, second, revised edition, 1984.
2. H. P. Barendregt: *Lambda calculi with types*, In S. Abramsky, *et al.* editors, *Handbook of Logic in Computer Science*, Vol II, pp. 117–309, Oxford University Press, 1992.
3. H. P. Barendregt, W. Dekkers, R. Statman: *Lambda Calculus with Types*, Cambridge University Press, 2012.
4. G. Barthe, M. H. Sørensen: *Domain-Free Pure Type Systems*, Lecture Notes in Computer Science 1234, pp. 9–20, 1997.
5. H.-J. Boehm: *Partial polymorphic type inference is undecidable*, Proc. IEEE 26th Annual Symposium on Foundations of Computer Science, pp. 339–345, 1985.
6. A. Church: *A formulation of the simple theory of types*, J. Symbolic Logic 5, pp. 56–68, 1940.
7. H. B. Curry: *Functionality in combinatory logic*, Proc. Nat. Acad. Science USA, 20, pp. 584–590, 1934.
8. G. Dowek: *Higher-order unification and matching*, In A. Robinson and A. Voronkov editors, *Handbook of Automated Reasoning*, Elsevier Science Publishers B.V. 2001.
9. K. Fujita: *Domain-free $\lambda\mu$-calculus*, Theoretical Informatics and Applications 34, pp. 433-466, 2000.
10. K. Fujita: *CPS-translation as adjoint*, Theoretical Computer Science 411 (2), pp. 324–340, 2010.
11. K. Fujita, A. Schubert: *Partially typed terms between Church-style and Curry-style*, Lecture Notes in Computer Science 1872, pp. 505-520, 2000.

12 K. Fujita, A. Schubert: *Existential type systems with no types in terms*, Lecture Notes in Computer Science 5608, pp. 112-125, 2009.
13 K. Fujita, A. Schubert: *The undecidability of type related problems in type-free style System* F, Leibniz International Proceedings in Informatics 6, pp. 103–118, 2010.
14 K. Fujita, A. Schubert: *The undecidability of type related problems in the type-free style System* F *with finitely stratified polymorphic types*, Information and Computation 218, pp. 69–87, 2012.
15 W. D. Goldfarb: *The undecidability of the second-order unification problems*, Theoretical Computer Science 13, pp. 225–230, 1981.
16 G. Huet: *The undecidability of unification in third order logic*, Information and Control 22, pp. 257–267, 1973.
17 W. A. Howard: *The formulae-as-types notion of construction*, In J. P. Seldin and J. R. Hindley editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press, 1980.
18 B. Jay, S. Peyton Jones: *Scrap your type applications*, Lecture Notes in Computer Science 5133, pp. 2–27, 2008.
19 A. J. Kfoury, J. Tiuryn: *Type Reconstruction in Finite Rank Fragments of the Second-Order λ-Calculus*, Information and Computation 98, pp. 228–257, 1992.
20 A. J. Kfoury, J. B. Wells: *A direct algorithm for type inference in the rank-2 fragment of the second-order λ-calculus*, Proc. ACM LISP and Functional Programming, pp. 196–207, 1994.
21 D. Leivant: *Polymorphic type inference*, POPL '83: Proc. 10th ACM Symposium on Principles of Programming Languages, pp. 88–98, 1983.
22 D. Le Botlan, D. Rémy: *Recasting* ML^{F}, Information and Computation 207, pp. 726–785, 2009.
23 D. Miller: *A logic programming language with lambda-abstraction, function variables, and simple unification*, J. Logic and Computation 1 (4), pp. 497–536, 1991.
24 K. Nakazawa, M. Tatsuta, Y. Kameyama, H. Nakano: *Type checking and typability in domain-free lambda calculi*, Theoretical Computer Science 412, pp. 6193–6207, 2011.
25 M. Parigot: *$\lambda\mu$-Calculus: An algorithmic interpretation of classical natural deduction*, Lecture Notes in Computer Science 624, pp. 190–201, 1992.
26 F. Pfenning: *Partial polymorphic type inference and higher-order unification*, Proc. ACM Conference on LISP and Functional Programming, pp. 153–163, 1988.
27 F. Pfenning: *On the undecidability of partial polymorphic type reconstruction*, Fundamenta Informaticae 19 (1,2), pp. 185–199, 1993.
28 D. Rémy, B. Yakobowski: *A Church-style intermediate language for ML^F*, Theoretical Computer Science 435, pp. 77–105, 2012.
29 W. Synder, J. H. Gallier: *Higher order unification revisited: Complete sets of transformations*, J. Symbolic Computation 8 (1,2) pp. 101–140, 1989.
30 A. Schubert: *Second-order unification and type inference for Church-style polymorphism*, POPL '98: Proc. 25th ACM Symposium on Principles of Programming Languages, pp. 279–288, 1998.
31 L. S. Van Benthem Jutting: *Typing in Pure Type Systems*, Information and Computation 105, pp. 30–41, 1993.
32 J. B. Wells: *Typability and type checking in system* F *are equivalent and undecidable*, Ann. Pure Appl. Logic 98, pp. 111–156, 1999.