Type Inference for Domain-Free $\lambda 2$

Ken-etsu Fujita

Department of Artificial Intelligence
Kyushu Institute of Technology
Iizuka 820-8502, Japan
fujiken@dumbo.ai.kyutech.ac.jp

July 1, 1999

Abstract

We will prove that type checking, typability, and type inference for domain-free $\lambda 2$ are undecidable. The type checking problem for domain-free $\lambda 2$ was posed by Barthe and Sørensen (1997). A certain second order unification problem is reduced to the problem of type inference for domain-free $\lambda 2$. The restricted second order unification has been proven undecidable by Schubert. The reduction method can be obtained from a simplification of Pfenning's reduction from the general problem of second order unification to the partial type inference problem. An analysis of the undecidability proof reveals that the typability problem is still undecidable even for a predicative fragment of domain-free $\lambda 2$, called the rank 2 fragment.

Technical Report in Computer Science and Systems Engineering
CSSE-5, Kyushu Institute of Technology, July 1, 1999

ISSN 1344-8803

1 Introduction

There are known three styles of (typed) λ-terms, called Curry-style, Church-style (for instance, see [Bare92]), and domain-free style [BHS96]. For some systems such as simply typed λ-calculus and ML [Mil78, DM82], it is well-known that the Curry-style and the corresponding Church-style are essentially equivalent [Bare92, HM93]. Hence, the Curry system serves as a short-hand for the Church system. On the other hand, recently, Barthe, Sørensen, and Hatcliff [BHS96, BS97] introduced the notion of domain-free pure type system. Terms in domain-free style have domain-free λ-abstraction. There exist two styles of domain-free pure type system, that is, a single syntactic category of expressions (non-sorted variables) and explicit distinction between objects, constructors, and kinds (sorted variables). Barthe and Sørensen posed a question to know whether the problem of type checking is decidable for domain-free $\lambda 2$ and $\lambda\omega$ (page 18 [BS97]). In this paper, we will show that type checking, typability, and type inference are, in general, undecidable for domain-free $\lambda 2$. In order to prove this, we reduce a certain second order unification problem to the problem of strong type inference for domain-free $\lambda 2$. The restricted second order unification problem has been proven undecidable by Schubert [Schu97, Schu98]. The reduction method can be obtained by a simplification of Pfenning's reduction [Pfen88, Pfen93] from the general problem of second order unification to the partial type inference problem.

Original motivation for domain-free systems comes from a study on classical type system which is an extension of intuitionistic type theory together with classical rules such as double negation elimination. The domain-free systems are useful to give continuation-passing style translations [BHS96] which provide a certain semantics of classical type system. Further, when we construct a polymorphic call-by-value calculus with control operators such as `callcc` or μ-operators [Pari92], the Curry style cannot work for a consistent system. For instance, see the traditional counterexample (ML with `callcc` is unsound) by Harper&Lillibridge [HL91], and see also a proof-theoretical observation in [Fuji99]. Hence, we introduced domain-free $\lambda\mu$-calculus [Fuji99], where the explicit type annotations for polymorphic terms play a role of choosing an appropriate computation under call-by-value. Our result in this paper also gives a negative answer to the problem of type checking for second-order $\lambda\mu$-calculus in domain-free style, which is a variant of Parigot's $\lambda\mu$-calculus in Curry style [Pari92].

Domain-free systems are also useful for a study on partial polymorphic type reconstruction [Boeh85, Pfen88]. Boehm [Boeh85] and Pfenning [Pfen88] have proven that the partial type reconstruction problem is, in general, undecidable for second-order λ-calculus. The typability problem for domain-free $\lambda 2$ can be regarded as a special case of the problem of type reconstruction for partially typed terms. Our result in this paper means that the restricted problem of type reconstruction for partially typed terms is still undecidable. Moreover, the

observation from the undecidability proof reveals that the typability problem is undecidable even for a predicative fragment of domain-free $\lambda 2$, called the rank 2 fragment [Leiv91, KT92]. This analysis also implies that the partial type reconstruction problem is still undecidable for the rank 2 fragment of second-order λ-calculus.

2 Curry-Style, Church-Style, and Domain-free

In Curry-style, terms are essentially type free [Curr34, CFC74, Hind97], and types can be assigned by rules of a type theory if well-formed. Terms in Church-style typed λ-calculus, on the one hand, are originally defined only from variables uniquely type annotated [Chur40]. On the other hand, terms in domain-free style have domain-free λ-abstraction [BS97], and second-order λ-calculus in domain-free style can be regarded, in a sense, as an intermediate representation between à la Curry and à la Church, as shown in the following table.

Styles of (typed) λ-terms

Style \ System	$\lambda^{\to}$	$\lambda 2$
Church-style	$\lambda x{:}\sigma.M$	$\lambda t.M, \quad M\sigma$
Domain-free	$\lambda x.M$	$\lambda t.M, \quad M\sigma$
Curry-style	$\lambda x.M$	M

We give a definition of domain-free $\lambda 2$-calculus. In terms of domain-free pure type system [BHS96, BS97], this domain-free system is constructed from sorted variables; term variable x and type variable t. Then, on the basis of the sorted variables, type abstraction can be represented by λt rather than the traditional Λt, and we also have explicit distinction between terms and types.

Domain-free $\lambda 2$:

Types $\quad \sigma ::= t \mid \sigma \to \sigma \mid \forall t.\sigma$

Contexts $\quad \Gamma ::= \langle\,\rangle \mid \Gamma, x{:}\sigma$

Terms $\quad M ::= x \mid \lambda x.M \mid MM \mid \lambda t.M \mid M\sigma$

Type Assignment Rules

$$\Gamma \vdash x : \Gamma(x)$$

$$\frac{\Gamma \vdash M_1 : \sigma_1 \to \sigma_2 \quad \Gamma \vdash M_2 : \sigma_1}{\Gamma \vdash M_1 M_2 : \sigma_2}\ (\to E) \qquad \frac{\Gamma, x{:}\sigma_1 \vdash M : \sigma_2}{\Gamma \vdash \lambda x.M : \sigma_1 \to \sigma_2}\ (\to I)$$

$$\frac{\Gamma \vdash M : \forall t.\sigma_1}{\Gamma \vdash M\sigma_2 : \sigma_1[t := \sigma_2]}\ (\forall E) \qquad \frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \lambda t.M : \forall t.\sigma}\ (\forall I)^*$$

where $(\forall I)^*$ denotes the eigenvariable condition.

The introduction rules, $(\rightarrow I)$ and $(\forall I)$ can be coded, respectively, as domain-free λ-abstractions based on the distinction between the sorted variables. The elimination rules, $(\rightarrow E)$ and $(\forall E)$ can also be represented, respectively, by the pairs of two expressions, based on sorted variables. Hence, when well-typed terms of domain-free $\lambda 2$ are given, the type assignment rules are uniquely determined by the shape of the terms. From this syntactical property of terms, we have the natural generation lemma for domain-free $\lambda 2$.

Lemma 1 (Generation lemma) *(1) If $\Gamma \vdash x : \sigma$, then $\Gamma(x) = \sigma$.*
(2) If $\Gamma \vdash M_1 M_2 : \sigma$, then $\Gamma \vdash M_1 : \sigma_1 \rightarrow \sigma$ and $\Gamma \vdash M_2 : \sigma_1$ for some σ_1.
(3) If $\Gamma \vdash \lambda x.M : \sigma$, then $\Gamma, x{:}\sigma_1 \vdash M : \sigma_2$ and $\sigma \equiv \sigma_1 \rightarrow \sigma_2$ for some σ_1 and σ_2.
(4) If $\Gamma \vdash \lambda t.M : \sigma$, then $\Gamma \vdash M : \sigma_1$ and $\sigma \equiv \forall t.\sigma_1$ and $t \notin FV(\Gamma)$ for some σ_1.
(5) If $\Gamma \vdash M\sigma_1 : \sigma$, then $\Gamma \vdash M : \forall t.\sigma_2$ and $\sigma \equiv \sigma_2[t := \sigma_1]$ for some σ_2.

3 Type Checking, Typability, and Type Inference for Domain-Free $\lambda 2$

The problems of type checking, typability, and type inference for Curry and Church $\lambda 2$ are investigated by Jutting [Jutt93], Wells [Well94], and Schubert [Schu97, Schu98], as shown in the following table.

Decidability for type checking, typability, and type inference of second-order λ-calculus ($\lambda 2$)

$\lambda 2$ \ Problem	$\Gamma \vdash M : \sigma$?	$\Gamma \vdash M$:?	$? \vdash M$:?
Church-style	yes [Jutt93]	yes [Jutt93]	no [Schu97]
Domain-free	$??^1$	$??^2$	$??^3$
Curry-style	no [Well94]	no [Well94]	no [Well94]

In this paper, we will prove that in the table above, all of $??^1$, $??^2$, and $??^3$ (case of strong type inference, see below) are "no", i.e., undecidable.

The problem of type inference is a problem that given a term M, are there a context Γ and a type σ such that $\Gamma \vdash M : \sigma$ is derivable? On the one hand, the problem of strong type inference [Tiur90] is a problem that given a term M and a context Γ_0, are there a context $\Gamma \supseteq \Gamma_0$ and a type σ such that $\Gamma \vdash M : \sigma$ is derivable? The strong type inference problem is naturally considered in the case where the system contains constants. The typability problem is a problem

that given a term M and a context Γ, is there a type σ such that $\Gamma \vdash M : \sigma$ is derivable? Finally, the type checking problem is a problem that given a term M, a type σ, and a context Γ, is the judgement $\Gamma \vdash M : \sigma$ derivable?

By the use of a type forgetful map, the three styles of judgements are equivalent in the following sense, where $|\ |$ is a domain erasing map ($|\lambda x : \sigma.M| = \lambda x.|M|$), and $||\ ||$ is a type erasing map ($||M\sigma|| = ||M||$, $||\lambda t.M|| = ||M||$):
(1) If $\Gamma \vdash M : \sigma$ in Church style, then $\Gamma \vdash |M| : \sigma$ in domain-free style.
(2) If $\Gamma \vdash M : \sigma$ in domain-free style, then $\Gamma \vdash ||M|| : \sigma$ in Curry style.

The inverse directions say that there exists a term whose erasure is the same as the original term [HM93].
(-1) If $\Gamma \vdash M : \sigma$ in domain-free style, then $\Gamma \vdash M_1 : \sigma$ in Church style for some M_1 such that $|M_1| \equiv M$.
(-2) If $\Gamma \vdash M : \sigma$ in Curry style, then $\Gamma \vdash M_2 : \sigma$ in domain-free style for some M_2 such that $||M_2|| \equiv M$.

For the problems above, however, the inverse directions are not straightforward, since the forgetful maps are not one-to-one. Here, we will directly study the problems for domain-free $\lambda 2$.

On the basis of the generation lemma (Lemma 1), we first observe that the strong type inference problem for domain-free $\lambda 2$ is reduced to the typability problem, and the typability problem for domain-free $\lambda 2$ is reduced to the type checking problem.

Lemma 2 $\exists \Gamma. \exists \sigma.\ \Gamma, \Gamma_0 \vdash M : \sigma$ *in domain-free* $\lambda 2$
$\iff$ $\exists \sigma.\ \Gamma_0 \vdash \lambda \vec{x}.M : \sigma$ *in domain-free* $\lambda 2$
$\iff$ $\Gamma_0 \vdash (\lambda xy.y)(\lambda \vec{x}.M) : t \to t$ *in domain-free* $\lambda 2$

4 Type Inference is Undecidable for Domain-Free $\lambda 2$

In this section, we prove that the problem of strong type inference for domain-free $\lambda 2$ is undecidable. To show this, we demonstrate a stronger result such that the problem of strong type inference is undecidable for the fragment of domain-free $\lambda 2$, called domain-free ML.

Domain-free ML:

Monotypes $\tau ::= t \mid \tau \to \tau$

Polytypes $\sigma ::= \tau \mid \forall t.\sigma$

Terms $M ::= x \mid \lambda x.M \mid MM \mid \lambda t.M \mid M\tau \mid \text{let } x = M \text{ in } M$

Contexts $\Gamma ::= \langle\ \rangle \mid \Gamma, x:\sigma$

Type Assignment Rules

$$\Gamma \vdash x : \Gamma(x)$$

$$\frac{\Gamma, x:\tau_1 \vdash M : \tau_2}{\Gamma \vdash \lambda x.M : \tau_1 \to \tau_2}\ (\to I) \qquad \frac{\Gamma \vdash M_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash M_2 : \tau_1}{\Gamma \vdash M_1 M_2 : \tau_2}\ (\to E)$$

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \lambda t.M : \forall t.\sigma}\ (\forall I)^* \qquad \frac{\Gamma \vdash M : \forall t.\sigma}{\Gamma \vdash M\tau : \sigma[t := \tau]}\ (\forall E)$$

$$\frac{\Gamma \vdash M_1 : \sigma \quad \Gamma, x:\sigma \vdash M_2 : \tau}{\Gamma \vdash \text{let } x = M_1 \text{ in } M_2 : \tau}$$

In the discussion of this section, we essentially need the following subsystem of the above domain-free ML:

$M ::= x \mid \lambda x.M \mid MM \mid x\tau_1 \cdots \tau_n$

$$\frac{\Gamma(x) = \forall t_1 \cdots t_n.\tau}{\Gamma \vdash x\tau_1 \cdots \tau_n : \tau[t_1 := \tau_1, \cdots t_n := \tau_n]}\ (n \geq 0)$$

$$\frac{\Gamma, x:\tau_1 \vdash M : \tau_2}{\Gamma \vdash \lambda x.M : \tau_1 \to \tau_2} \qquad \frac{\Gamma \vdash M_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash M_2 : \tau_1}{\Gamma \vdash M_1 M_2 : \tau_2}$$

We first introduce a restricted second-order unification problem for well-formed second-order expressions, which are defined from monotypes τ, binary function constant $\to$, and n-ary second-order function variable X ($n \geq 0$) whose arguments contain no variables. Such terms for the second-order unification are denoted by T or U. A well-formed expression is defined as follows:
(1) A type variable t is a well-formed expression.
(2) If X is an n-ary variable ($n \geq 0$) and τ_i ($1 \leq i \leq n$) are monotypes, then $X\tau_1 \cdots \tau_n$ is well-formed.
(3) If T_1 and T_2 are well-formed, then so is $T_1 \to T_2$.

Given a well-formed expression T, a set of (unification) variables in T denoted by $uVar(T)$ and a set of constants in T denoted by $Con(T)$ are defined, respectively, as follows:
$uVar(t) = \emptyset$;
$uVar(X\tau_1 \cdots \tau_n) = \{X\}$ ($n \geq 0$);

$uVar(T_1 \to T_2) = uVar(T_1) \cup uVar(T_2)$.

$Con(t) = \{t\}$;
$Con(X\tau_1 \cdots \tau_n) = \emptyset$ $(n \geq 0)$;
$Con(T_1 \to T_2) = Con(T_1) \cup Con(T_2)$.

Given well-formed T_1 and T_2. Let $uVar(T_1, T_2)$ be $\{X_1, \cdots, X_n\}$. The unification problem $(T_1 \doteq T_2)$ is a problem to find well-formed U_i for each X_i where $1 \leq i \leq n$, such that
(1) Let X_i be $k(i)$-ary variable, and S be a substitution such that
$[X_1 := \lambda t_1 \cdots t_{k(1)}.U_1, \cdots, X_n := \lambda t_1 \cdots t_{k(n)}.U_n]$.
Then $S(T_1) =_\beta S(T_2)$ holds.
(2) We have $uVar(U_i) = \emptyset$ where $1 \leq i \leq n$.

If we have a substitution S such that the above (1) and (2) are satisfied, then we say that T_1 and T_2 are unifiable, and that the unification problem has an answer S. In this case, from the definition, there exists a monotype τ such that $S(T_1) =_\beta \tau =_\beta S(T_2)$.

Theorem 1 (Schubert[Schu97]) *The second-order unification problem on the well-formed expressions is undecidable.*

Schubert [Schu97] has proved that the halting problem for two-counter automata is reduced to the unification problem, where a two-counter automata can simulate an arbitrary Turing machine. On the basis of his result, it is enough to consider one pair of T_1 and T_2, which contain binary function constant $\to$ and at least one constant.

In order to give a reduction from the unification problem to the problem of type inference for domain-free ML, we first define a (pre)context Σ. The context itself may not be an ML-context, but it becomes an ML-context under some substitution if unifiable. This can be justified, since the reduction is formalized as follows:

the unification problem $T_1 \doteq T_2$ has an answer if and
only if there exist Γ and τ such that $\Gamma, \Gamma_0 \vdash M : \tau$ in domain-free ML.

Here, Γ_0 and M are given by T_1 and T_2. Only if T_1 and T_2 are unifiable, say the unifier S, then the ML-context Γ can be obtained as a subcontext of $S(\Sigma)$, such that $S(\Sigma) = \Gamma, \Gamma_0$. Moreover, the monotype τ can also be obtained as a substitution instance (of $ty(\cdot)$ defined below) by S.

Given a well-formed T, then we construct the context $\Sigma[T]$, such that
(1) For each $t \in Con(T)$, t is inhabited in $\Sigma[T]$, i.e., $\Sigma[T](x) = t$ for some x.
(2) For each n-ary variable $X \in uVar(T)$ where $n \geq 0$, the universal closure $\forall t_1 \cdots t_n.(Xt_1 \cdots t_n)$ is inhabited in $\Sigma[T]$.

$\Sigma[T_1, T_2]$ is also defined similarly, and we simply write Σ for $\Sigma[T_1, T_2]$.

Let T_1 and T_2 be well-formed. Given a second-order unification problem $T_1 \doteq T_2$, then, following Pfenning [Pfen93], we construct a term of domain-free ML by the use of the following $\mathcal{UT}$ and $\mathcal{TI}$:

$\mathcal{UT}(\Sigma[T_1, T_2]; T_1 \doteq T_2) =$

$$\lambda z_1.\lambda z_2.\lambda f.fz_1(fz_2(\lambda g.g(\mathcal{TI}(\Sigma; z_1; T_1))(\mathcal{TI}(\Sigma; z_2; T_2)))),$$

where $\mathcal{TI}(\Sigma; z; T)$ is defined by induction on T:

(1) $\mathcal{TI}(\Sigma; z; t) = \lambda f.fz(fx(\lambda g.g))$
where $\Sigma(x) = t \in Con(T_1, T_2)$

(2) $\mathcal{TI}(\Sigma; z; X\tau_1 \cdots \tau_n) = \lambda f.fz(f(x\tau_1 \cdots \tau_n)(\lambda g.g))$
where $\Sigma(x) = \forall t_1 \cdots t_n.(Xt_1 \cdots t_n)$

(3) $\mathcal{TI}(\Sigma; z; T_1 \to T_2) =$
$\lambda z_1.\lambda z_2.\lambda f.f(zz_1)(fz_2(\lambda g.g(\mathcal{TI}(\Sigma; z_1; T_1))(\mathcal{TI}(\Sigma; z_2; T_2))))$

Remark 1 *The reduction via $\mathcal{UT}$ and $\mathcal{TI}$ gives a β-normal term.*

The translation $\mathcal{TI}(\Sigma; z; T)$ says that type of z would be a substitution instance of T, see Lemma 3 below.

We next construct $ty(T)$ that is a type of $\mathcal{TI}(\Sigma; z; T)$. Although $ty(T)$ itself may not be a monotype, it becomes a monotype under some substitution if unifiable. Here of course we assume that we have countably many type variables to use a fresh type variable t for each application of the following definition:

(0) $ty(\tau) = (\tau \to (t \to t) \to t \to t) \to t \to t$
for $\tau \in Con(T)$;

(1) $ty(X\tau_1 \cdots \tau_n) = ((X\tau_1 \cdots \tau_n) \to (t \to t) \to t \to t) \to t \to t$
where $n \geq 0$;

(2) $ty(T_1 \to T_2) = T_1 \to T_2 \to (T_2 \to A \to A) \to A$
where $A \equiv (ty(T_1) \to ty(T_2) \to t) \to t$.

Lemma 3 *$S(\Sigma[T]), x{:}\tau \vdash \mathcal{TI}(\Sigma[T]; x; T) : S(ty(T))$ in domain-free* ML *if and only if $S(T) =_\beta \tau$.*

Proof. By induction on T:

(1) $T \equiv t'$

From the definition, in domain-free ML we have

$$S(\Sigma), x{:}\tau \vdash \lambda f.fx(fy(\lambda g.g)) : (t' \to (t \to t) \to t \to t) \to t \to t$$

where $\Sigma(y) = t'$. Here, type of x and y must be equal, i.e., $\tau \equiv t'$.

(2) $T \equiv X\tau_1 \cdots \tau_n$

Let $S(\Sigma(y))$ be $\forall t_1 \cdots t_n.((SX)t_1 \cdots t_n)$. We have

$$S(\Sigma), x{:}\tau \vdash \lambda f.fx(f(y\tau_1 \cdots \tau_n)(\lambda g.g)) : (((SX)\tau_1 \cdots \tau_n) \to (t \to t) \to t \to t) \to t \to t$$

in domain-free ML. Here, type of x and $y\tau_1 \cdots \tau_n$ must be equal. That is, $\tau =_\beta ((SX)\tau_1 \cdots \tau_n)$.

(3) $T \equiv T_1 \to T_2$

From the definition, in domain-free ML we have

$$S(\Sigma), x:\tau \vdash \lambda z_1.\lambda z_2.\lambda f.f(xz_1)(fz_2(\lambda g.g(\mathcal{TI}(\Sigma;z_1;T_1))(\mathcal{TI}(\Sigma;z_2;T_2))))$$
$$: S(T_1) \to S(T_2) \to (S(T_2) \to A \to A) \to A$$

where $A \equiv (S(ty(T_1)) \to S(ty(T_2)) \to t) \to t$.

Then, we also have

$$S(\Sigma), x:\tau, z_1:S(T_1), z_2:S(T_2) \vdash$$
$$\lambda f.f(xz_1)(fz_2(\lambda g.g(\mathcal{TI}(\Sigma;z_1;T_1))(\mathcal{TI}(\Sigma;z_2;T_2))))$$
$$: (S(T_2) \to A \to A) \to A\ .$$

Here, from the induction hypotheses, we have the following:

$$S(\Sigma), z_1:\tau_3 \vdash \mathcal{TI}(\Sigma;z_1;T_1) : S(ty(T_1)) \quad \text{iff} \quad \tau_3 =_\beta S(T_1)$$
$$S(\Sigma), z_2:\tau_4 \vdash \mathcal{TI}(\Sigma;z_2;T_2) : S(ty(T_2)) \quad \text{iff} \quad \tau_4 =_\beta S(T_2)$$

Now, type of (xz_1) and z_2 must be equal, i.e., $\tau =_\beta S(T_1) \to S(T_2)$. □

Lemma 4 (main lemma) *$S(T_1) =_\beta S(T_2)$ if and only if $S(\Sigma) \vdash \mathcal{UT}(\Sigma; T_1 \doteq T_2) : S(ty(T_1 \to T_2))$ in domain-free* ML.

Proof. $S(\Sigma) \vdash \mathcal{UT}(\Sigma; T_1 \doteq T_2) : S(ty(T_1 \to T_2))$

iff (def)

$$S(\Sigma) \vdash \lambda z_1.\lambda z_2.\lambda f.fz_1(fz_2(\lambda g.g(\mathcal{TI}(\Sigma;z_1;T_1))(\mathcal{TI}(\Sigma;z_2;T_2))))$$
$$: S(T_1) \to S(T_2) \to (S(T_2) \to A \to A) \to A$$

where $A \equiv (S(ty(T_1)) \to S(ty(T_2)) \to t) \to t$

iff

$$S(\Sigma), z_1:S(T_1), z_2:S(T_2) \vdash$$
$$\lambda f.fz_1(fz_2(\lambda g.g(\mathcal{TI}(\Sigma;z_1;T_1))(\mathcal{TI}(\Sigma;z_2;T_2))))$$
$$: (S(T_2) \to A \to A) \to A$$

iff (Lemma 3)

$S(T_1) =_\beta S(T_2)$. □

Proposition 1 *The unification problem on the well-formed expressions is reduced to the problem of strong type inference for domain-free* ML. *In other words, $S(T_1) =_\beta S(T_2) \iff \exists\Gamma.\exists\tau.\ \Gamma, \Gamma_0^{T_{1,2}} \vdash M^{T_{1,2}} : \tau$ in domain-free* ML.

Proof. ($\Rightarrow$):

From Lemma 4, $\Gamma_0^{T_{1,2}}$ and $M^{T_{1,2}}$ are determined by T_1 and T_2, such that for each $t \in Con(T_1, T_2)$, we have $\Gamma_0(x) = t$ for some x, and that $M = \mathcal{UT}(\Sigma; T_1 \doteq T_2)$. Then the unifier S gives Γ and τ, respectively, such that $S(\Sigma[T_1, T_2]) = \Gamma, \Gamma_0$ and $S(ty(T_1 \to T_2)) = \tau$.

($\Leftarrow$):

Given $\Gamma_0^{T_{1,2}}$ and $M^{T_{1,2}}$, and assume that there exist τ and Γ such that $\Gamma = \{x_1 : \tau_1, \cdots, x_m : \forall t_1 \cdots t_n.\tau_m\}$. For each $X_i \in uVar(T_1, T_2)$, assume that $\Sigma(x_1) = X_1, \cdots, \Sigma(x_m) = \forall t_1 \cdots t_n.(X_m t_1 \cdots t_n)$. Then an answer to the trivial second-order matching problem such that $X_1 \doteq \tau_1, \cdots, (X_m t_1 \cdots t_n) \doteq \tau_m$ finds a matching S for $ty(T_1 \to T_2) \doteq \tau$, since if $S(\Sigma[T]), x:\tau_0 \vdash \mathcal{TI}(\Sigma[T]; x; T) : \tau$ for

some τ, then $S(ty(T)) =_\beta \tau$. From Lemma 4, the unifier S is an answer to the unification problem $T_1 \doteq T_2$. □

Proposition 2 *The problem of strong type inference is undecidable for domain-free* ML*, even when the given term is in β-normal.*

Proof. From Theorem 1, Proposition 1, and Remark 1. □

Theorem 2 *Type checking, typability, and strong type inference are undecidable for domain-free $\lambda 2$.*

Proof. From Proposition 2 and Lemma 2. Moreover, even in the case where the given term is in β-normal, typability and strong type inference for domain-free $\lambda 2$ are still undecidable. □

5 Related Work and Concluding Remarks

Relating to Proposition 2, the problem of strong type inference is also undecidable for domain-free ML with non-sorted variables [BS97], since the given proof with a slight modification still works for the definition of T, where type variable t is replaced with variable x, and $\tau ::= x \mid \tau \to \tau$, by the use of a single syntactic category of variables x.

Pfenning [Pfen93] has proved that the second-order unification problem can be reduced to the problem of type reconstruction for partially typed terms, such that

$P ::= \ x \mid \lambda x{:}\sigma.P \mid PP \mid \lambda t.P \mid P\sigma \mid \lambda x.P \mid P[\,]$,

where the mark $[\,]$ must be left to show a type has been erased. The typability problem for domain-free $\lambda 2$ can be regarded as a special case of the problem of type reconstruction for partially typed terms with neither $\lambda x{:}\sigma.P$ nor $P[\,]$. Hence, from Theorem 2, the restricted problem of type reconstruction for partially typed terms is still undecidable. This would mean that the difficulty for the partial type reconstruction problem comes from "domain-free" especially with respect to polymorphic abstraction (see the discussion below on typability for domain-free ML_2) rather than from a type-hole $[\,]$.

Finally, we summarize decidability of type checking, typability, and strong type inference for domain-free $\lambda 2$ and ML with sorted variables.

Decidability for type checking, typability, and strong type inference of domain-free $\lambda 2$ and ML

Domain-free	$\Gamma_0 \vdash M : \sigma$?	$\Gamma_0 \vdash M$:?	?, $\Gamma_0 \vdash M$:?
$\lambda 2$	*no*	*no* (nf)	*no* (nf)
ML	yes	yes	*no* (nf)

In the above table, *no* (nf) means that even when the given term M is in β-normal, the problem is undecidable. The context Γ_0 cannot be empty here, since the undecidability result used in this paper requires at least one constant [Schu97, Schu98]. The strong type inference with no predefined contexts is still open.

The result obtained here finds a negative answer to the question posed by Barthe and Sørensen [BS97] to know whether the problem of type checking is decidable for domain-free $\lambda 2$ and $\lambda\omega$. Moreover, the type checking problem for domain-free $\lambda\mu$-calculus introduced in [Fuji99] also becomes undecidable.

On the one hand, the typability for domain-free ML can be obtained from the well-known $\mathcal{W}$ [Mil78, DM82]. This algorithm can find the principal type for the closed M in domain-free style. In the same way, the type reconstruction for partially typed terms of ML can be solved. On the one hand, we cannot have a principal type inference algorithm for Damas-Milner ML in Curry style, such that $PTI(M)$ gives principal type τ and context Γ which satisfy $\Gamma \vdash M : \tau$ in Damas-Milner ML (here, M may not be closed, and Γ can contain polymorphic types σ). Otherwise, we could obtain a type inference algorithm for domain-free ML, which is a contradiction to Proposition 2.

On the other hand, the problem of typability becomes undecidable for some predicative extension of the domain-free ML. We introduce a predicative fragment of domain-free $\lambda 2$, called domain-free ML_2. This extension allows us to abstract a term variable with a polymorphic type σ (polymorphic abstraction), but not to apply a polymorphic function to a polymorphic type (i.e., only to a monomorphic type τ). For this purpose, an extension of type schemes is introduced as follows:

$$\rho ::= \ \sigma_1 \to \cdots \to \sigma_n \to \tau \ (n \geq 0)$$

This type ρ belongs to the so-called S(2)-class in [KT92], which is a special form of restrict types of rank 2 [Leiv91].

Domain-free ML_2:

$$\frac{\Gamma(x) = \sigma}{\Gamma \vdash x : \sigma}$$

$$\frac{\Gamma \vdash M_1 : \sigma \to \rho \quad \Gamma \vdash M_2 : \sigma}{\Gamma \vdash M_1 M_2 : \rho} \ (\to E) \qquad \frac{\Gamma, x{:}\sigma \vdash M : \rho}{\Gamma \vdash \lambda x.M : \sigma \to \rho} \ (\to I)$$

$$\frac{\Gamma \vdash M : \forall t t_1 \cdots t_n.\rho}{\Gamma \vdash M\tau : \forall t_1 \cdots t_n.\rho[t := \tau]} \ (\forall E) \qquad \frac{\Gamma \vdash M : \forall t_1 \cdots t_n.\rho}{\Gamma \vdash \lambda t.M : \forall t t_1 \cdots t_n.\rho} \ (\forall I)^*$$

The problem of strong type inference in domain-free ML can be reduced to the typability problem in domain-free ML_2. That is, let $\{x_1, \cdots, x_n\}$ be a set of free variables in M;

$\exists\sigma_1 \cdots \sigma_n.\exists\tau.\ \Gamma_0, x_1 : \sigma_1, \cdots, x_n : \sigma_n \vdash M : \tau$ in domain-free ML if and only if $\exists\sigma_1 \cdots \sigma_n.\exists\tau.\ \Gamma_0 \vdash \lambda x_1 \cdots x_n.M^* : \sigma_1 \to \cdots \to \sigma_n \to \tau$ in domain-free ML_2,

since let-expression can be coded in ML_2, such that $(\texttt{let}\ x = M_1\ \texttt{in}\ M_2)^* = (\lambda x.M_2^*)M_1^*$. (Strictly speaking, we need no `let`-expression for the undecidability of domain-free ML.) From Proposition2, we can obtain the undecidable typability with respect to a certain predicative fragment of domain-free $\lambda 2$;

"the typability problem is undecidable for the rank 2 fragment [Leiv91, KT92] of domain-free $\lambda 2$".

This result also means that the partial type reconstruction problem is still undecidable even for the rank 2 fragment of $\lambda 2$.

Corollary 1 *The problem of typability (with non-empty context) for domain-free* ML_2 *is undecidable.*

Following Pfenning [Pfen93], the partial type reconstruction problem is undecidable for a predicative fragment of $\lambda 2$, and this fragment can be regared as a subsystem of the rank 2 fragment of $\lambda 2$.

Acknowledgements I am grateful to J.Roger Hindley and Horai-Takahashi Masako for helpful discussions. I would like to thank Frank Pfenning and Aleksy Schubert for valuable comments on this work.

References

[Bare92] H.P.Barendregt: Lambda Calculi with Types, Handbook of Logic in Computer Science Vol.II, Oxford University Press, pp. 1–189, 1992.

[Boeh85] Hans-J.Boehm: Partial Polymorphic Type Inference is Undecidable, *Proc. 26th Annual Symposium of Fundations of Computer Science*, pp. 339–345, 1985.

[BHS96] G.Barthe, J.Hatcliff, and M.H.Sørensen: CSP Translations and Applications: The Cube and Beyond, *Proc. the 2nd ACM SIGPLAN Workshop on Continuations*, pp. 1–31, 1996.

[BS97] G.Barthe and M.H.Sørensen: Domain-free Pure Type Systems, Lecture Notes in Computer Science 1234, pp. 9–20, 1997.

[Chur40] A.Church: A Formulation of the Simple Theory of Types, *The Journal of Symbolic Logic*, Vol. 5, pp. 56–68, 1940.

[Curr34] H.B.Curry: Functionality in combinatory logic, *Proc. National Academy of Sciences of the USA* 20, pp. 584–590, 1934.

[CFC74] H.B.Curry, R.Feys, and W.Craig: Combinatory Logic, Volume 1 (Third printing), North-Holland, 1974.

[DM82] L.Damas and R.Milner: Principal type-schemes for functional programs, *Proc. 9th Annual ACM Symposium on Principles of Programming Languages*, pp. 207–212, 1982.

[Fuji99] K.Fujita: Explicitly Typed $\lambda\mu$-Calculus for Polymorphism and Call-by-Value, Lecture Notes in Computer Science 1581, pp. 162–176, 1999.

[HL91] R.Harper and M.Lillibridge: ML with callcc is unsound, The Types Form, 8 July 1991.

[Hind97] J.R.Hindley: Basic Simple Type Theory, Cambridge University Press, 1997.

[HM93] R.Harper and J.C.Mitchell: On The Type Structure of Standard ML, *ACM Transactions on Programming Languages and Systems*, Vol. 15, No.2, pp. 210–252, 1993.

[Tiur90] J.Tiuryn: Type Inference Problems: A Survey, Lecture Notes in Computer Science 452, pp. 105–120, 1990.

[KT92] A.J.Kfoury and J.Tiuryn: Type Reconstruction in Finite Rank Fragments of the Second-Order λ-Calculus, *Information and Computation* 98, pp. 228–257, 1992.

[Leiv91] D.Leivant: Finitely Stratified Polymorphism, *Information and Computation* 93, pp. 93–113, 1991.

[Mil78] R.Milner: A Theory of Type Polymorphism in Programming, *Journal of Computer and System Sciences* 17, pp. 348–375, 1978.

[Pari92] M.Parigot: $\lambda\mu$-Calculus: An Algorithmic Interpretation of Classical Natural Deduction, Lecture Notes in Computer Science 624, pp. 190–201, 1992.

[Pfen88] F.Pfenning: Partial polymorphic type inference and higher-order unification, *Proc. ACM Conference on Lisp and Functional Programming*, pp. 153–163, 1998.

[Pfen93] F.Pfenning: On the undecidability of partial polymorphic type reconstruction, *Fundamenta Informaticae* 19, pp. 185–199, 1993.

[Schu97] A.Schubert: Second-order unification and type inference for Church-style, Tech. Report TR 97-02 (239), Institute of Informatics, Warsaw University, January 1997.

[Schu98] A.Schubert: Second-order unification and type inference for Church-style, *Proc. ACM Symposium on Principles of Programming Languages*, pp. 279–288, 1998.

[Jutt93] L.S. Van B. Jutting: Typing in Pure Type Systems, *Information and Computation* 105, pp. 30–41, 1993.

[Well94] J.B.Wells: Typability and Type Checking in the Second-Order λ-Calculus Are Equivalent and Undecidable, *Proc. IEEE Symposium on Logic in Computer Science*, pp. 176–185, 1994.