Mathematical Structures in Computer Science, 5(3):419–430, 1995.

Strong Normalization with Non-structural Subtyping

Mitchell Wand[*][†] Patrick O'Keefe[‡] Jens Palsberg[§]

June 1, 1994

Abstract

We study a type system with a notion of subtyping that involves a largest type $\top$, a smallest type $\perp$, atomic coercions between base types, and the usual ordering of function types. We prove that any λ-term typable in this system is strongly normalizing; this solves an open problem of Thatte. We also prove that the fragment without $\perp$ types strictly fewer terms. This demonstrates that $\perp$ adds power to a type system.

1 Introduction

Statically-typed languages are desirable for many reasons, but they are often more restrictive than dynamically-typed languages. In particular, it is desirable to allow strongly-typed languages to have "holes" in the type structure, so that portions of the program that are not fully understood may be written using dynamic typing. There have been several proposals for creating such holes, such as [3, 9, 10]. Typically, one gives the result of such an untyped computation a special type, `untyped`. Such a value can be passed as an ordinary value, but is not manipulable except by a polymorphic procedure, such as `print` [10]. Thatte [9] called this *partial* type inference.

The addition of a type `untyped` allows several different kinds of flexibility. It allows portions of a program to escape the scrutiny of the type-checker [10]; it allows for heterogeneous lists and persistent data [9]; and it can also be used to facilitate binding-time analysis or analysis of type errors [3]. It also serves as a basis for dealing with the "don't care" types for records in [8].

[*]Work supported by the National Science Foundation and DARPA under grants CCR-9002253 and CCR-9014603.

[†]College of Computer Science, Northeastern University, 360 Huntington Avenue, 161CN, Boston, MA 02115, USA. E-mail: `wand@ccs.neu.edu`.

[‡]85 East India Row #39B, Boston, MA 02110, USA. E-mail: `pmo@world.std.com`.

[§]College of Computer Science, Northeastern University, 360 Huntington Avenue, 161CN, Boston, MA 02115, USA. E-mail: `palsberg@ccs.neu.edu`.

In this paper we study a type system with a notion of subtyping that involves a largest type $\top$, a smallest type $\bot$, atomic coercions between base types, and the usual ordering of function types. We prove that any λ-term typable in this system is strongly normalizing. The constant type $\top$ corresponds to `untyped`. The constant type $\bot$ is useful for typing "dead code". Thatte's *partially typed* terms [9] correspond to the fragment without $\bot$. Our result implies that partially typed terms are strongly normalizing as well, solving an open problem in [9]. We also prove that the fragment without $\bot$ types strictly fewer terms.

2 The Formal System

2.1 Types

The set of partial types is parameterized by a partially ordered set of atomic types $(B, \sqsubseteq)$, where B consists of constant type symbols and where $B \cap \{\bot, \top, \rightarrow\} = \emptyset$. For example, we might have $B = \{\mathsf{Int}, \mathsf{Real}\}$, where $\mathsf{Int} \sqsubseteq \mathsf{Real}$, $\mathsf{Int} \sqsubseteq \mathsf{Int}$, and $\mathsf{Real} \sqsubseteq \mathsf{Real}$.

Partial types comprise an ordered set $(T_B, \leq)$, where T_B is the set of well-formed finite terms over $B \cup \{\bot, \top, \rightarrow\}$. (For an extension with recursive types, see the paper by Amadio and Cardelli [1].) Here, $\bot$ and $\top$ are constant symbols, and $\rightarrow$ is a binary type constructor. The order $\leq$ is defined inductively by the rules

$$\frac{b \sqsubseteq b'}{b \leq b'}$$

$$\bot \leq t$$

$$t \leq \top$$

$$\frac{s' \leq s \quad t \leq t'}{s \rightarrow t \leq s' \rightarrow t'}$$

where b, b' range over atomic types, and s, s', t, t' range over types. Intuitively, if $s \leq t$, then we can coerce s to t. The coercions among elements of B have been called *atomic* by Mitchell [5]. We will refer to the fourth rule as the *congruence* rule for function types. Typical inclusions are $\bot \leq \bot \rightarrow \top$, $\top \rightarrow \top \leq \top$, $\top \rightarrow \top \leq (\top \rightarrow \top) \rightarrow \top$.

Lemma 1 *The relation $\leq$ is a partial order.*

Proof. Reflexivity: Given t, we need to derive $t \leq t$. This follows by straightforward induction on the structure of t, using that $\sqsubseteq$ is reflexive.

Asymmetry: Assume that $t \leq t'$ and $t' \leq t$ are derivable. We proceed by induction on the sum of the sizes of t and t'. If either t or t' is in $B \cup \{\bot, \top\}$, then clearly $t = t'$. Otherwise, both t and t' are function types and the two derivations of $t \leq t'$ and $t' \leq t$ have the congruence rule as their last step. The result then follows from the induction hypothesis.

Transitivity: Assume $t \leq t'$ and $t' \leq t''$ are derivable. We need to show that also $t \leq t''$ is derivable. We proceed by induction on the structure of t''.

Suppose first that t'' is an atomic type. Then t' is either $\bot$ or an atomic type. If $t' = \bot$, then $t = \bot$, so $t \leq t''$. If t' is an atomic type, then t is either $\bot$ or an atomic type. If $t = \bot$, the $t \leq t''$, and if t is an atomic type, then the result follows from $\sqsubseteq$ being transitive.

Suppose then that $t'' = \bot$. Clearly, $t = t' = \bot$, so $t \leq t''$.

Suppose then that $t'' = \top$. Clearly, $t \leq t''$.

Suppose finally that t'' is a function type. Then t' is either $\bot$ or a function type. If $t' = \bot$, then $t = \bot$, so $t \leq t''$. If t' is a function type, then t is either $\bot$ or a function type. If $t = \bot$, then clearly $t \leq t''$. Thus, we are left with the case where all of t, t', and t'' are function types. Then, both of the derivations of $t \leq t'$ and $t' \leq t''$ have uses of the congruence rule as their last step. The result then follows from the induction hypothesis. □

2.2 Programs

Programs are untyped λ-terms with typed constants. We write constants as c^t where $t \in T_B$. Polymorphic **let** can be treated by using the equivalence **let** $x = M$ **in** N as $(\lambda d.N[d/x])M$, where d is fresh.

If M is a λ-term, t is a type, and A is a type environment, i.e., a partial function assigning types to variables, then the judgement

$$A \vdash M : t$$

means that M has the type t in the environment A. Formally, this holds when the judgement is derivable using the following five rules:

$$A \vdash c^t : t$$

$$A \vdash x : t \quad (A(x) = t)$$

$$\frac{A[x \leftarrow t_1] \vdash M : t_2}{A \vdash (\lambda x.M) : t_1 \to t_2}$$

$$\frac{A \vdash M : t_1 \to t_2 \quad A \vdash N : t_1}{A \vdash (M\ N) : t_2}$$

$$\frac{A \vdash M : t \quad t \leq t'}{A \vdash M : t'}$$

The first four rules are the usual rules for simple types and the last rule is the rule of *subsumption*. We denote this type system by PTB (PTB indicates *Partial Types with Bottom*). Thatte's system of partial types did not include $\bot$; the fragment of the type system without $\bot$ will be denoted PT. Reduction is as usual given by the rewriting rule scheme $(\lambda x.M)M' \Rightarrow M[M'/x]$; Mitchell [5] has shown that subject reduction holds.

This system types terms which are not typable in the simply-typed λ-calculus. For example, consider $\lambda f.(fK(fI))$, where K and I are the usual combinators. This is not typable in the ordinary calculus, since K and I have different types, but it is typable under partial typing: assign f the type $\top \to \top \to \top$. Both the K and I can be coerced to type $\top$, and the result

(fI), of type $\top \to \top$, can be coerced to $\top$ to form the second argument of the first f. Therefore the entire term has type $(\top \to \top \to \top) \to \top$.

Similarly, some self-application is possible: $(\lambda x.xx)$ has type $(\top \to t) \to t$ for all t, since the final x can be coerced to $\top$.

However, not all terms are typable in this system. Our main result is that any λ-term typable in this system is strongly normalizing. To indicate the flavor of those strongly normalizing λ-term which are not typable in this system, we will prove in Section 5 that $(\lambda x.xxx)(\lambda y.y)$ is not typable in PTB.

In Section 5 we also prove that the λ-term $(\lambda f.f(fx))(\lambda v.vy)$ is typable in PTB but *not* in PT. This demonstrates that $\bot$ adds power to a type system.

While both PTB and System F contain only strongly normalizing λ-terms, they are incomparable because

1. The λ-term $(\lambda x.xxx)(\lambda y.y)$ is not typable in PTB, but it is typable in F, and
2. The λ-term $(\lambda x.\lambda y.y(xI)(xK))\Delta$, where $I = \lambda a.a$, $K = \lambda b.\lambda c.b$, and $\Delta = \lambda d.dd$, is not typable in F [2], but it is typable in PTB and also in PT.

It follows that PT and F are also incomparable. Notice also that the λ-term $(\lambda f.f(fx))(\lambda v.vy)$ is typable in F, but not in PT.

3 Witnesses

A *witness* for either a type derivation or a subtype derivation is an explicitly typed and simply typed λ-term with typed constants. If W is such a λ-term, t is a type, and A is a type environment, then the judgement

$$A \vdash_{ST} W : t$$

means that W has the simple type t in the environment A. Formally, this holds when the judgement is derivable using the following four rules:

$$A \vdash_{ST} c^t : t$$

$$A \vdash_{ST} x : t \quad (A(x) = t)$$

$$\frac{A[x \leftarrow t_1] \vdash_{ST} M : t_2}{A \vdash_{ST} (\lambda x : t_1.M) : t_1 \to t_2}$$

$$\frac{A \vdash_{ST} M : t_1 \to t_2 \quad A \vdash_{ST} N : t_1}{A \vdash_{ST} (M\ N) : t_2}$$

The subscript ST indicates *Simple Typing.*

We assume that the language of witnesses contains the following constants:

- For each $b, b' \in B$, where $b \sqsubseteq b'$, there is a constant $c^{b \to b'}$.

- For each $t \in T_B$, there are constants $c^{\bot \to t}$ and $c^{t \to \top}$.

The judgement $t \leq t'$ $[C^{t \to t'}]$ means that $C^{t \to t'}$ is a witness for $t \leq t'$. Formally, this holds when the judgement is derivable using the following four rules:

$$\frac{b \sqsubseteq b'}{b \leq b' \; [c^{b \to b'}]}$$

$$\bot \leq t \; [c^{\bot \to t}]$$

$$t \leq \top \; [c^{t \to \top}]$$

$$\frac{s' \leq s \; [C^{s' \to s}] \quad t \leq t' \; [C^{t \to t'}]}{s \to t \leq s' \to t' \; [\lambda f : s \to t . \lambda x : s' . C^{t \to t'}(f(C^{s' \to s} x))]}$$

The judgement

$$A \vdash_w M : t \; [M']$$

means that M' is a witness for $A \vdash M : t$. Formally, this holds when the judgement is derivable using the following five rules:

$$A \vdash_w c^t : t \; [c^t]$$

$$A \vdash_w x : t \; [x] \quad (A(x) = t)$$

$$\frac{A[x \leftarrow t_1] \vdash_w M : t_2 \; [M']}{A \vdash_w (\lambda x . M) : t_1 \to t_2 \; [\lambda x : t_1 . M']}$$

$$\frac{A \vdash_w M : t_1 \to t_2 \; [M'] \quad A \vdash_w N : t_1 \; [N']}{A \vdash_w (M \; N) : t_2 \; [M' N']}$$

$$\frac{A \vdash_w M : t \; [M'] \quad t \leq t' \; [C^{t \to t'}]}{A \vdash_w M : t' \; [C^{t \to t'} M]}$$

Thus the witness of a type derivation is the strongly-typed term which is obtained by "inserting the necessary coercions". This intuition is made precise by the following observations.

Lemma 2 *The following are true:*

1. *If $s' \leq s$ $[C^{s' \to s}]$, then $s' \leq s$ and $\emptyset \vdash_{ST} C^{s' \to s} : s' \to s$.*
2. *If $A \vdash_w M : t$ $[M']$, then $A \vdash M : t$ and $A \vdash_{ST} M' : t$.*
3. *If $A \vdash M : t$, then there exists M' such that $A \vdash_w M : t$ $[M']$.*

Proof. Straightforward. □

Lemma 3 *If $A[x \leftarrow t_1] \vdash_w M : t_2$ $[M']$ and $A \vdash_w N : t_1$ $[N']$, then $A \vdash_w M[N/x] : t_2$ $[M'[N'/x]]$.*

Proof. By induction on the structure of the derivation of $A[x \leftarrow t_1] \vdash_w M : t_2$ $[M']$. □

4 The Main Theorem

Theorem 4 *If $A \vdash M\colon t$, then M is strongly normalizing.*

Proof. By Lemma 2, choose a λ-term M' so that $A \vdash_w M\colon t\ [M']$. We will prove the following claim:

> If $A \vdash_w M\colon t\ [M']$ and $M \Rightarrow N$, then there exists N' so that $A \vdash_w N\colon t\ [N']$ and $M' \Rightarrow N'$.

To see that the theorem follows from the claim, consider an infinite reduction sequence starting from M. Then, by the claim, we get an infinite reduction sequence starting from M'. But since M' is typable in the simply-typed λ-calculus, it is strongly normalizing. This gives a contradiction.

To prove the claim, we proceed by induction on the size of M'. All cases but the following are straightforward. Consider the case in the induction step where $M \equiv (\lambda x.M_1)M_2 \Rightarrow M_1[M_2/x]$. The last step in the derivation of $A \vdash_w M\colon t\ [M']$ is

$$\frac{A \vdash_w (\lambda x.M_1)\colon t_1 \to t\ [W] \quad A \vdash_w M_2\colon t_1\ [M_2']}{A \vdash_w (\lambda x.M_1)M_2\colon t\ [WM_2']}$$

So $M' \equiv WM_2'$. There are two cases, depending on how $A \vdash_w (\lambda x.M_1)\colon t_1 \to t\ [W]$ is derived.

In the first case, the last step of the derivation of $A \vdash_w (\lambda x.M_1)\colon t_1 \to t\ [W]$ is:

$$\frac{A[x \leftarrow t_1] \vdash_w M_1\colon t\ [M_1']}{A \vdash_w (\lambda x.M_1)\colon t_1 \to t\ [\lambda x : t_1.M_1']}$$

so $W \equiv \lambda x : t_1.M_1'$. From Lemma 3 we get that

$$A \vdash_w M_1[M_2/x]\colon t\ [M_1'[M_2'/x]]$$

so the λ-term $M_1'[M_2'/x]$ has both the desired properties.

In the second case, the last step in the derivation of $A \vdash_w (\lambda x.M_1)\colon t_1 \to t\ [W]$ is:

$$\frac{A \vdash_w (\lambda x.M_1)\colon u\ [W'] \quad u \leq t_1 \to t\ [C^{u \to (t_1 \to t)}]}{A \vdash_w (\lambda x.M_1)\colon t_1 \to t\ [C^{u \to (t_1 \to t)}W']}$$

so $W \equiv C^{u \to (t_1 \to t)}W'$. Now, $u = u_1 \to u_2$, for some u_1 and u_2, since $\top \not\leq t_1 \to t$, and no abstraction can have a type in $B \cup \{\bot\}$. So we have

$$\begin{aligned} M' &\equiv WM_2' \\ &\equiv C^{(u_1 \to u_2) \to (t_1 \to t)}W'M_2' \\ &\equiv (\lambda f : u_1 \to u_2.\lambda x : t_1.C^{u_2 \to t}(f(C^{t_1 \to u_1}x)))W'M_2' \end{aligned}$$

From $A \vdash_w M_2\colon t_1\ [M_2']$ and $t_1 \leq u_1\ [C^{t_1 \to u_1}]$, we get $A \vdash_w M_2\colon u_1\ [C^{t_1 \to u_1}M_2']$. From that and $A \vdash_w (\lambda x.M_1)\colon u\ [W']$ we get $A \vdash_w (\lambda x.M_1)M_2\colon u_2\ [W'(C^{t_1 \to u_1}M_2')]$. By the above equivalence, the term $W'(C^{t_1 \to u_1} M_2')$ is smaller than M' (it is sufficient that it does not include a copy of $C^{u_2 \to t}$). Hence, by the induction hypothesis, we can choose N' so that $A \vdash_w M_1[M_2/x] \colon u_2\ [N']$ and $W'(C^{t_1 \to u_1} M_2') \Rightarrow N'$. So $A \vdash_w M_1[M_2/x] : t\ [C^{u_2 \to t} N']$. We also have

$$M' \Rightarrow C^{u_2 \to t}(W'(C^{t_1 \to u_1} M_2')) \Rightarrow C^{u_2 \to t} N'$$

so the λ-term $C^{u_2 \to t} N'$ has both the desired properties. □

Notice that the claim in the proof of Theorem 4 is a refinement of the subject reduction theorem, taking reduction of witnesses into account.

5 Two λ-terms

In this section we prove that

- The strongly normalizing λ-term $(\lambda x.xxx)(\lambda y.y)$ is not typable, and
- The λ-term $(\lambda f.f(fx))(\lambda v.vy)$ is typable; but it is *not* typable in the fragment of the type system without $\perp$.

To obtain such negative results, we rephrase the type inference in terms of solving a system of type constraints, following for example [4].

5.1 Constraints

Given a λ-term M, assume that M has been α-converted so that all bound variables are distinct. Let X_M be the set of λ-variables x occurring in M, and let Y_M be a set of variables disjoint from X_M consisting of one variable $[\![F]\!]$ for each occurrence of a subterm F of M. (The notation $[\![F]\!]$ is ambiguous because there may be more than one occurrence of F in M. However, it will always be clear from context which occurrence is meant.) We generate the following system of inequalities over $X_M \cup Y_M$ (notice that λ-variables are also used as type variables):

- for every occurrence in M of a subterm of the form c^t, the inequality

$$t \ \leq \ [\![c^t]\!] \ ;$$

- for every occurrence in M of a subterm of the form $\lambda x.F$, the inequality

$$x \to [\![F]\!] \ \leq \ [\![\lambda x.F]\!] \ ;$$

- for every occurrence in M of a subterm of the form FG, the inequality

$$[\![F]\!] \ \leq \ [\![G]\!] \to [\![FG]\!] \ ;$$

- for every occurrence in M of a λ-variable x, the inequality

$$x \;\leq\; [\![x]\!] \,.$$

Denote by $T(M)$ the system of constraints generated from M in this fashion. The solutions of $T(M)$ over T_B correspond to the possible type annotations of M in a sense made precise by Theorem 5.

Let A be a type environment assigning a type to each λ-variable occurring freely in M. If L is a function assigning a type to each variable in $X_M \cup Y_M$, we say that L *extends* A if A and L agree on the domain of A.

Theorem 5 *The judgement $A \vdash M : t$ is derivable if and only if there exists a solution L of $T(M)$ extending A such that $L([\![M]\!]) = t$. In particular, if M is closed, then M is typable with type t if and only if there exists a solution L of $T(M)$ such that $L([\![M]\!]) = t$.*

Proof. Similar to the proof of Theorem 2.1 in the journal version of [4], in outline as follows. Given a solution of the constraint system, it is straightforward to construct a derivation of $A \vdash M{:}\,t$. Conversely, observe that if $A \vdash M{:}\,t$ is derivable, then there exists a derivation of $A \vdash M{:}\,t$ such that each use of one of the ordinary rules is followed by exactly one use of the subsumption rule. The approach in for example [11, 7] then gives a set of inequalities of the desired form. □

5.2 An untypable λ-term

In this subsection we prove that the the strongly normalizing λ-term $(\lambda x.xxx)(\lambda y.y)$ is not typable.

First we present a small system of constraints which is unsolvable. Let S be the set consisting of the two constraints

$$\begin{aligned} y \to y &\leq y \\ y &\leq (y \to y) \to \top \end{aligned}$$

Proposition 6 *S is not solvable over T_B.*

Proof. Suppose that S is solvable. Clearly, no solution can assign an element of $B \cup \{\bot, \top\}$ to y. So choose a type $s \to t$ so that it solves S when assigned to y and so that no type of smaller height has this property. From $y \to y \leq y$ and $y \leq (y \to y) \to \top$ we get

$$\begin{aligned} (s \to t) \to (s \to t) &\leq s \to t \\ s \to t &\leq ((s \to t) \to (s \to t)) \to \top \end{aligned}$$

so

$$\begin{aligned} s &\leq s \to t \\ s \to t &\leq t \\ (s \to t) \to (s \to t) &\leq s \end{aligned}$$

Clearly, s cannot be an element of $B \cup \{\bot, \top\}$. So write $s = a \to b$. Thus,

$$\begin{array}{rcl} a \to b & \leq & (a \to b) \to t \\ (a \to b) \to t & \leq & t \\ ((a \to b) \to t) \to ((a \to b) \to t) & \leq & a \to b \end{array}$$

From this we derive

$$\begin{array}{rcl} a \to b & \leq & a \\ b & \leq & t \\ (a \to b) \to t & \leq & t \\ a & \leq & (a \to b) \to t \\ (a \to b) \to t & \leq & b \end{array}$$

So

$$a \leq (a \to b) \to t \leq b$$

thus $a \leq b$. This means that also

$$a \to a \leq a \to b \leq a$$

and

$$a \leq (a \to b) \to t \leq (a \to a) \to \top$$

So assigning a to y yields a solution of S. This is a contradiction since a has smaller height than $s \to t$ (because $s \to t = (a \to b) \to t$). □

Proposition 7 *The λ-term $(\lambda x.xxx)(\lambda y.y)$ is not typable in* PTB.

Proof. To distinguish the three occurrences of x, we denote them x_1, x_2, and x_3. The constraint system $T((\lambda x.xxx)(\lambda y.y))$ is as follows.

$$\begin{array}{rcll} x & \leq & [\![x_1]\!] & (1) \\ x & \leq & [\![x_2]\!] & (2) \\ x & \leq & [\![x_3]\!] & (3) \\ y & \leq & [\![y]\!] & (4) \\ [\![\lambda x.xxx]\!] & \leq & [\![\lambda y.y]\!] \to [\![(\lambda x.xxx)(\lambda y.y)]\!] & (5) \\ y \to [\![y]\!] & \leq & [\![\lambda y.y]\!] & (6) \\ x \to [\![xxx]\!] & \leq & [\![\lambda x.xxx]\!] & (7) \\ [\![x_1]\!] & \leq & [\![x_2]\!] \to [\![x_1 x_2]\!] & (8) \\ [\![x_1 x_2]\!] & \leq & [\![x_3]\!] \to [\![xxx]\!] & (9) \end{array}$$

From these constraints we derive

$$\begin{array}{rcll} x & \leq & [\![x_1]\!] & \text{(using 1)} \\ & \leq & [\![x_2]\!] \to [\![x_1 x_2]\!] & \text{(using 8)} \\ & \leq & [\![x_2]\!] \to ([\![x_3]\!] \to [\![xxx]\!]) & \text{(using 9)} \\ & \leq & x \to (x \to \top) & \text{(using 2, 3, and the rule for } \top) \end{array}$$

so $x \leq x \to (x \to \top)$. We also derive

$$\begin{array}{rcll} x \to [\![xxx]\!] & \leq & [\![\lambda x.xxx]\!] & \text{(using 7)} \\ & \leq & [\![\lambda y.y]\!] \to [\![(\lambda x.xxx)(\lambda y.y)]\!] & \text{(using 5)} \\ & \leq & (y \to [\![y]\!]) \to [\![(\lambda x.xxx)(\lambda y.y)]\!] & \text{(using 6)} \\ & \leq & (y \to y) \to [\![(\lambda x.xxx)(\lambda y.y)]\!] & \text{(using 4)} \end{array}$$

so $y \to y \leq x$. We can now derive

$$\begin{array}{rcll} y \to y & \leq & x & \text{(using the above)} \\ & \leq & x \to (x \to \top) & \text{(using the above)} \\ & \leq & (y \to y) \to ((y \to y) \to \top) & \text{(using the above)} \end{array}$$

so $y \to y \leq (y \to y) \to ((y \to y) \to \top)$. Using the congruence rule on this inequality, we get the two constraints in the constraint system S. Thus, if $T((\lambda x.xxx)(\lambda y.y))$ is solvable, then S is solvable. So, since S is unsolvable over T_B by Proposition 6, we get that $T((\lambda x.xxx)(\lambda y.y))$ is unsolvable over T_B. □

5.3 The power of $\bot$

In this subsection we prove that the λ-term $(\lambda f.f(fx))(\lambda v.vy)$ is typable in PTB; but also that it is *not* typable in PT. The latter result could also be obtained using one of the two known type inference algorithms for PT [6, 4]. To give some intuition about *why* this λ-term is untypable over PT, we give a direct proof. Notice that $(\lambda f.f(fx))(\lambda v.vy)$ is not closed. We might have used the closed λ-term $\lambda x.\lambda y.(\lambda f.f(fx))(\lambda v.vy)$ instead, but we have preferred to use the shorter $(\lambda f.f(fx))(\lambda v.vy)$.

We denote by T'_B set of well-formed finite terms over $B \cup \{\top, \to\}$. Notice that $T'_B \subseteq T_B$. Intuitively, the set T'_B is the subset of T_B "without $\bot$".

We first present a small system of constraints which is unsolvable over T'_B but solvable over T_B. Let S' be the singleton set consisting of the constraint

$$v \leq y \to v$$

Proposition 8 *S' is not solvable over T'_B.*

Proof. First note that we can solve S' over T_B by assigning $\bot$ to v and any type to y. Suppose then that S' is solvable over T'_B. Choose a solution L over T'_B with the property that any other solution over T_B maps v to a type of at least the same height as $L(v)$. Clearly, L does not map v to an element of $B \cup \{\top\}$. So write $L(v) = s \to t$. From $v \leq y \to v$ we get

$$s \to t \leq L(y) \to (s \to t)$$

So

$$t \leq s \to t$$

Thus, by assigning t to v and by assigning s to y, we obtain a solution of S'. This is a contradiction since t has smaller height than $s \to t$. □

Proposition 9 *The λ-term $(\lambda f.f(fx))(\lambda v.vy)$ is typable in* PTB *but not in* PT.

Proof. To distinguish the two occurrences of f, we denote them f_1 and f_2. Moreover, we will denote by M the term $(\lambda f.f(fx))(\lambda v.vy)$. The constraint system $T(M)$ is as follows.

$$\begin{aligned}
[\![\lambda f.f(fx)]\!] &\leq [\![\lambda v.vy]\!] \to [\![M]\!] &\quad (10)\\
f \to [\![f(fx)]\!] &\leq [\![\lambda f.f(fx)]\!] &\quad (11)\\
[\![f_1]\!] &\leq [\![fx]\!] \to [\![f(fx)]\!] &\quad (12)\\
[\![f_2]\!] &\leq x \to [\![fx]\!] &\quad (13)\\
f &\leq [\![f_1]\!] &\quad (14)\\
f &\leq [\![f_2]\!] &\quad (15)\\
v \to [\![vy]\!] &\leq [\![\lambda v.vy]\!] &\quad (16)\\
[\![v]\!] &\leq y \to [\![vy]\!] &\quad (17)\\
v &\leq [\![v]\!] &\quad (18)
\end{aligned}$$

From these constraints we derive

$$\begin{aligned}
([\![fx]\!] \to [\![f(fx)]\!]) \to [\![f(fx)]\!] &\leq [\![f_1]\!] \to [\![f(fx)]\!] && \text{(using 12)}\\
&\leq f \to [\![f(fx)]\!] && \text{(using 14)}\\
&\leq [\![\lambda f.f(fx)]\!] && \text{(using 11)}\\
&\leq [\![\lambda v.vy]\!] \to [\![M]\!] && \text{(using 10)}\\
&\leq (v \to [\![vy]\!]) \to [\![M]\!] && \text{(using 16)}
\end{aligned}$$

so $[\![fx]\!] \leq v$ and $v \to [\![vy]\!] \leq f$. We also derive

$$\begin{aligned}
v \to [\![vy]\!] &\leq f && \text{(using the above)}\\
&\leq [\![f_2]\!] && \text{(using 15)}\\
&\leq x \to [\![fx]\!] && \text{(using 13)}
\end{aligned}$$

so $[\![vy]\!] \leq [\![fx]\!]$. We can now derive

$$\begin{aligned}
v &\leq [\![v]\!] && \text{(using 18)}\\
&\leq y \to [\![vy]\!] && \text{(using 17)}\\
&\leq y \to [\![fx]\!] && \text{(using the above)}\\
&\leq y \to v && \text{(using the above)}
\end{aligned}$$

so $v \leq y \to v$.

Thus, if $T(M)$ is solvable over T'_B, then S' is solvable over T'_B. So, since S' is unsolvable over T'_B by Proposition 8, we get that $T(M)$ is not solvable over T'_B, and hence M is not typable in PT.

To see that M is typable in PTB, give f the type $\bot \to \bot$, give v the type $\bot$, and give both x and y the type $\bot$. The subterm $\lambda v.vy$ has type $\bot$ because we can coerce the type of the occurrence of v to be $\bot \to \bot$. □

Acknowledgements

The authors thank Fritz Henglein and Jonathan Young for helpful discussions.

References

[1] Roberto M. Amadio and Luca Cardelli. Subtyping recursive types. *ACM Transactions on Programming Languages and Systems*, 15(4):575–631, 1993. Also in Proc. POPL'91.

[2] Paola Giannini and Simona Ronchi Della Rocca. Characterization of typings in polymorphic type discipline. In *Proc. LICS'88, Third Annual Symposium on Logic in Computer Science*, pages 61–70, 1988.

[3] Carsten K. Gomard. Partial type inference for untyped functional programs. In *Proc. ACM Conference on Lisp and Functional Programming*, pages 282–287, 1990.

[4] Dexter Kozen, Jens Palsberg, and Michael I. Schwartzbach. Efficient inference of partial types. *Journal of Computer and System Sciences*, 49(2):306–324, 1994. Also in Proc. FOCS'92, 33rd IEEE Symposium on Foundations of Computer Science, pages 363–371, Pittsburgh, Pennsylvania, October 1992.

[5] John C. Mitchell. Type inference with simple subtypes. *Journal of Functional Programming*, 1:245–285, 1991.

[6] Patrick M. O'Keefe and Mitchell Wand. Type inference for partial types is decidable. In *Proc. ESOP'92, European Symposium on Programming*, pages 408–417. Springer-Verlag (*LNCS* 582), 1992.

[7] Jens Palsberg and Michael I. Schwartzbach. Safety analysis versus type inference for partial types. *Information Processing Letters*, 43:175–180, 1992.

[8] Didier Rémy. Typechecking records and variants in a natural extension of ML. In *Sixteenth Symposium on Principles of Programming Languages*, pages 77–88, 1989.

[9] Satish Thatte. Type inference with partial types. In *Proc. International Colloquium on Automata, Languages, and Programming 1988*, pages 615–629. Springer-Verlag (*LNCS* 317), 1988.

[10] Mitchell Wand. A semantic prototyping system. In *Proc. ACM SIGPLAN'84 Symposium on Compiler Construction*, pages 213–221. Sigplan Notices, 1984.

[11] Mitchell Wand. Type inference for record concatenation and multiple inheritance. *Information and Computation*, 93(1):1–15, 1991.