Questions questions questions . . .

- What is a partial order over a set $S$?
- What is an equivalence relation over a set $S$?
- What is the greatest lower bound of a set $S$?
- When a system of equations is in solved form?
État Projet
Consider the famous fixed-point combinator

\[ Y \equiv \lambda x. (\lambda y. x(yy))(\lambda y. x(yy)) \]

[...] It is not typable [...] But \( Y \) has considerable practical importance [...] and a theory that excludes it seems rather over restrictive. Can we find a more generous type-theory, one that assigns a type to \( Y \)?

*Types with Intersection: An Introduction*

J.R. Hindley, 1992
Consider the famous fixed-point combinator

\[ Y \equiv \lambda x. (\lambda y. x(yy))(\lambda y. x( yy)) \]

It is not typable

But \( Y \) has considerable practical importance, and a theory that excludes it seems rather over restrictive. Can we find a more generous type-theory, one that assigns a type to \( Y \)?

Types with Intersection: An Introduction
J.R. Hindley, 1992

1979
Intuition

Is term \( \lambda x.xx \) typeable in the simply typed \( \lambda \)-calculus?
Intuition

Is term $\lambda x.x \cdot x$ typeable in the simply typed $\lambda$-calculus?

Simple types not enough

$\frac{x : A \vdash x : A \rightarrow B \quad x : A \vdash x : A}{\vdash xx : B}$

Possible approaches,

- Have a type $A$ such that $A \approx A \rightarrow B$ involved equivalence
- Have a type $C$ that is $A$ and $A \rightarrow B$

$C = A \land (A \rightarrow B)$
Formally

- \( M, N ::= x \mid MN \mid \lambda x. M \) \hspace{1cm} \( \lambda \)-calculus
- \( A, B ::= a \mid \omega \mid A \rightarrow A \mid A \land A \) \hspace{1cm} types

Typing rules,

\[
\begin{align*}
\Gamma, x : A & \vdash x : A \\
\Gamma \vdash \lambda x. M : A \rightarrow B \\
\Gamma \vdash M : A \rightarrow B & \quad \Gamma \vdash N : A \\
\Gamma \vdash MN : B \\
\Gamma \vdash M : \omega \\
\Gamma \vdash M : A \land B & \\
\Gamma \vdash M : A \land B & \quad \Gamma \vdash M : A \\
\Gamma \vdash M : B \\
\Gamma \vdash \lambda x. M(x) : A & \quad x \notin FV(M)
\end{align*}
\]
Examples

Let $C = (A \rightarrow B) \land A$ for some $A$, $B$, and $\Gamma = \{ f : B \rightarrow D \}$

What about $\mathcal{Y}$??
Typing $\mathcal{Y}$

Intuitively

Recall $\mathcal{Y} = \lambda f. ZZ$ where $Z = \lambda x. f(xx)$

for every term $F$

$F(\mathcal{Y}F) =_{\beta} \mathcal{Y}F$

(1) Suppose $F$ function with codomain $A$

(2) $FM$ has type $A$ whenever $M$ “outputs” at all

$F : \omega \to A$

(3) As $\mathcal{Y}F$ in range of $F$ we have $\mathcal{Y}F : A$

(4) Because of (2) and (3)

$\mathcal{Y} : (\omega \to A) \to A$
Typing \( \mathcal{U} \)

Formally

Let \( Z = \lambda x. f(xx) \) and \( B = \omega \to A \) for some \( A \)

\[
\text{1. } f : B, x : B \vdash f : \omega \to A \quad f : \omega \to A, x : \omega \to A \vdash xx : \omega \\
\quad f : B, x : B \vdash f(xx) : A \\
\quad f : B \vdash Z : B \to A \\
\]

\[
\text{2. } f : B, x : \omega \vdash f : \omega \to A \quad f : B, x : \omega \vdash xx : \omega \\
\quad f : B, x : \omega \vdash f(xx) : A \\
\quad f : B \vdash Z : B \\
\]
Properties

Uniqueness false:

\[ \vdash Y : \omega \quad \vdash Y : (\omega \rightarrow A) \rightarrow A \]

Theorem (Characterisation terms with NF)

A \lambda-term \( M \) has a NF if and only if \( \Gamma \vdash M : A \) for some context \( \Gamma \) and type \( A \), neither of which contains \( \omega \).
Towards practice

Suppose we have a function

\[
plus : \text{int} \rightarrow \text{int} \rightarrow \text{int} \land \text{string} \rightarrow \text{string} \rightarrow \text{string}
\]

What is the type of the following function?

\[
mult x y = \text{if } y == 0 \text{ then } 0 \text{ else } plus x (mult x (y - 1))
\]
Towards practice

Suppose we have a function

\[
plus : \text{int} \rightarrow \text{int} \rightarrow \text{int} \land \text{string} \rightarrow \text{string} \rightarrow \text{string}
\]

What is the type of the following function?

\[
mult \ x \ y = \text{if } y == 0 \text{ then } 0 \text{ else } plus \ x \ (mult \ x \ (y - 1))
\]

A compiler for a language with intersection types might even provide two different object-code sequences for the different versions of \(plus\) [\ldots]

– B.C. Pierce, Intersection Types and Bounded Polymorphism

\[\text{C}Duce\]
http://www.cduce.org/

A working programming language
So why recursive types?

Recursive types are not necessary to type \( \mathcal{Y} \ldots \)
So why recursive types?

Recursive types are not necessary to type \( \mathcal{U} \)...

but we still have

Circular definitions

\[
\text{IntList} = [] \mid \text{int} : \text{IntList}
\]

(1)

It is convenient to have a

- finitary object \( A \)
- that satisfies equations as (1).
So why recursive types?

Recursive types are not necessary to type $\mathcal{Y} \ldots$

but we still have

Circular definitions

\[
\text{IntList} = [] \mid \text{int} : \text{IntList}
\]

\[
F(X) = \{\epsilon\} \cup \mathbb{Z} \times X
\]

(1)

It is convenient to have a

- finitary object $A$

- that satisfies equations as (1).
Pour le TP

Projet projet projet!!

implement type inference for recursive types