TYPE CHECKING VS TYPE INFERENCE

- Standard type checking

```
int f (int x) { return x+1; }
int g (int y) { return f(y+1) * 2; }
```

examine body of each function

use declared types to check agreement

- type inference

```
~~int~~ f ( ~~int~~ x) { return x+1; }
~~int~~ g ( ~~int~~ y) { return f(y+1) * 2; }
```

examine code without type info.

infer the most general types that could have been declared.

ML, Haskell are designed to make type inference feasible.

THE TYPE INFERENCE PROBLEM

- input:

  a program without types
  (ex- lambda calculus).

- output:

  a program with type for every expression
  (ex- typed lambda calculus).

  every expression is annotated with its most general type.

WHY STUDY TYPE INFERENCE?

- reduces syntactic overhead of expressive types.
- guarenteed to produce most general type.
- widely regarded as important language innovation.
- illustrative example of a flow-insensitive static analysis algorithm.

HISTORY

1958 :

Haskell Curry , Robert Feys invented type inference algorithm for the simply typed lambda calculus.

1969 :

Hindley extended the algorithm to a richer language, and proved it always produced the most general type.

1978 :

Milner, independently developed equivalent algorithm (called W) during his work designing ML.

1982 :

Damas proved the algorithm was complete.

TYPE INFERENCE

fun $x \to 2 + x$
-: $int \to int$ = <fun>

$+$ has type $int \to int \to int$

2 has type int

$\therefore$ $\underset{(2)}{int} \to \underset{(x)}{int} \to \underset{(fun)}{int}$

$\therefore$ $x : int$

fun : $\underset{(x)}{int} \to int$

ex $x := b[z]$

$a[b[y]] := x$

① $b[y] = int$
② $x = int$
③ $y = int$
④ $z = int$
⑤ $b = int[\,]$
⑥ $a = int[\,]$

ex fun f => f 3

① 3: int

② f: int → a

③ fun: (int → a) → a

ex fun f => f (f 3)

① 3: int

② f: int → a

③ f: int → int

④ fun: (int → int) → int

ex fun f => f (f "hi")

① fun: (string → string) → string

ex fun f => f (f 3, f 4)

ex let square = $\lambda z . z * z$
in
$\lambda f . \lambda x . \lambda y$
if (f x y)
then (f (square x) y)
else (f x (f x y))

(1) $* : int \to int \to int$

(2) $z : int$

(3) $square : int \to int$

(4) $f : int \to a \to bool$

(5) $f : int \to bool \to bool$.

(6) $(int \to bool \to bool) \to int \to bool \to bool$