

TYPE CHECKING VS TYPE INFERENCE

- Standard type checking

int & (int n) & return n+1; }

int g (int y) & return f (y+1) * 2; 3

examine body of each function

use declared types to check agreement

type interence

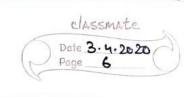
int f (int x) & return x+1; 3

ixt g Cixt y) & return & Cy+1) * 2; }

examine code without type info.

infer the most general types that could have been declared.

ML, Haskell are designed to make type inherence beasible.



THE TYPE INFERENCE PROBLEM

input:

a program without types Cex-lambda calculus).

- output:

a program with type for every expression

every expression is annotated with its

(ex- typed lambda colculus)

WHY STUDY TYPE INFERENCE?

types.

guarenteed to produce most general type.

coidely regarded as important language innovation.

static analysis algorithm.

HISTORY

1958:

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

3(7) -3-1

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 $n \rightarrow 2 + n$ -: int \rightarrow int = $\langle 6un \rangle$

+ has type int > int > int

2 has type int

 $\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}$

int: ont

bun: $int \rightarrow int$ (x)

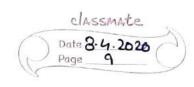
ex x:= b[z]

a[b[y]]:= (ax

② x = int ③ b = int[]

Obsyl = int Oz = int

3 y = int @ a = int []



ex fon 6 => 63

① 8: int

B f: int → a

3 fon: (int -> a) -> a

ex fon f => f (f 3)

(1) 3: int

② f: int → a

3 $f: int \rightarrow int$

(3) bun: (int -> int) -> int

ex bun 6 => 6 (6 "hi")

① fun: (string -> string) -> string

ex fon f => f (63, 64)



ex let square = 2 z . z * z

not. not. noy

else Cfx Cfxy)

1 *: int > lint > int

2 z i cnt

3 Square: int -> int

(i) f: int → a → bool

(5) f: int → book → book.

6 (int > bool > bool) > int > bool > bool