

CSE114A lecture 20

Agenda:

- Polymorphism!

Damas-Hindley-Milner type inference

is known for its support for polymorphism

$$\lambda x \rightarrow x :: a \rightarrow a$$

$$\frac{(x, T_1) : \Gamma \vdash e :: T_2}{\Gamma \vdash \lambda x \rightarrow e :: T_1 \rightarrow T_2}$$

$$\lambda x \rightarrow \lambda y \rightarrow x :: a \rightarrow b \rightarrow a$$

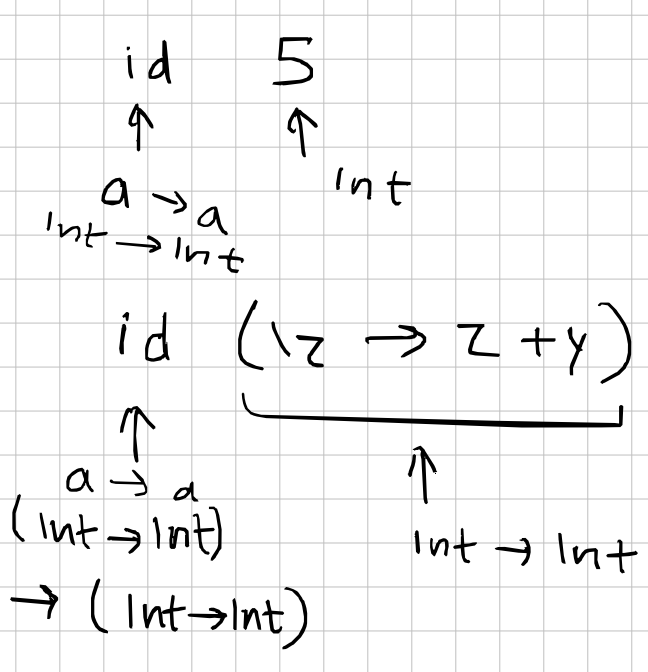
$$P_1 \rightarrow P_2 \rightarrow P_1$$

Think about this program:

```
let id = \x \rightarrow x in
let y = id 5 in
id (\z \rightarrow z + y)
```

:: Int → Int

Quiz question 1: is this program well-typed? If so, what's its type?



How do we deal with the fact that a polymorphic function might be applied to arguments of different types at different places in a program?

1) When we put a type into the type environment for a let-bound variable, we generalize that type.
(This happens in the ELet case of 'infer'.)

$$\underbrace{\text{forall } a.}_{\text{implicit quantifier!}} \underbrace{a \rightarrow a}_{f \text{ should have type forall } a. a \rightarrow a}$$

```
let f = \x \rightarrow x in
let y = f 5 in
f (\z \rightarrow z + y)
```

gets instantiated with a fresh type variable which gets unified with Int → Int
gets its own instantiation

2) When we use a let-bound variable from the type environment we instantiate its type with a fresh type variable specific to that one use!
(This happens in the EVar case of 'infer'.)

(This is called "let-polymorphism")

$$\underbrace{(\lambda x \rightarrow x)}_{}$$