# CSE114A - lecture 17!

- Today we will write some code
- unification

the challenging case from last time:

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

:: Int -> Int

$\Gamma \vdash e :: T$

Type environment

[(f, forall a. a -> a), (y, Int)]

"Poly" encompasses both of these

$$\dfrac{\dfrac{\dfrac{(x,\boxed{\text{Int}}) \text{ is in } \Gamma(x,\boxed{\text{Int}})}{\Gamma(x,\boxed{\text{Int}}) \vdash x :: \boxed{\text{Int}}}\text{[T-Var]} \qquad \dfrac{1 \in \mathbb{Z}}{\Gamma(x,\boxed{\phantom{I}}) \vdash 1 :: \boxed{\text{Int}}}\text{[T-Num]}}{\Gamma(x,\boxed{\text{Int}}) \vdash x+1 :: \boxed{\text{Int}}}\text{[T-Add]}}{\Gamma \vdash \backslash x \to \underbrace{x+1}_{} :: \boxed{\text{Int}} \to \boxed{\text{Int}}}\text{[T-Lam]}$$

To do constraint-based type inference:

- whenever you need to guess a type, don't!
  - Just use a fresh type variable.
- whenever a rule imposes a constraint, try to find the right substitution for the free type variables to satisfy the constraint.

→ this step is called unification!

unification means:

may contain type variables

given two types $T_1$ and $T_2$, find a substitution that makes them equal.

maps type variables to types.

This substitution is called a unifier of $T_1$ and $T_2$.

| $T_1$ | $T_2$ | unifier |
|---|---|---|
| "a" | Int | [("a", Int)] |
| "a" → "a" | Int → Int | [("a", Int)] |
| "a" → Int | Int → "b" | [("a", Int), ("b", Int)] (or [("a", Int), ("b", Int), ("c", Bool)]) |
| "a" | "a" | [] |
| Int | Int | [] |
| Int | Int → Int | can't unify. |
| "a" | "a" → "a" | can't unify. |
| Int | "a" → "a" | can't unify. |
| "a" → Int → Int | "b" → "c" | [("a", "b") ("c", Int → Int)] or [("b", "a") ("c", Int → Int)] |