An example with type equations instead.

Q What is the type of

Fix $(\lambda x.x)$

?

Recall

$$\frac{}{x :: P, \Gamma \vdash x :: Q \quad \langle P = Q \rangle} \text{ proj}$$

$$\frac{x :: X, \Gamma \vdash \quad t :: Y \langle E \rangle}{\Gamma \vdash \lambda x.t :: Q \quad \langle \exists X, Y. Q = X \to Y, E \rangle} \text{ abst}$$

$$\frac{\Gamma \vdash f :: Z \quad \langle E_1 \rangle \quad \Gamma \vdash t :: X \quad \langle E_2 \rangle}{\Gamma \vdash (ft) :: Q \quad \langle \exists X, Z. Z = X \to Q, E_1, E_2 \rangle} \text{ app}$$

$$\frac{\Gamma \vdash t :: Z \quad \langle E \rangle}{\Gamma \vdash \mathsf{fix}[t] :: Q \quad \langle \exists Z. Z = Q \to Q, E \rangle} \text{ fix}$$

(from your assignment.)

$fix\ (\lambda x.x)$

$$\dfrac{\dfrac{}{x :: X \vdash x :: Y \ \langle X = Y \rangle}\ proj}{\dfrac{}{\emptyset \vdash \lambda x.x :: Z \langle \exists X, Y.\, Z = X \to Y,\ X = Y \rangle}\ abst}$$

$$\dfrac{}{\emptyset \vdash fix\ (\lambda x.x) :: Q \ \langle \exists Z.\, Z = Q \to Q,\ \exists X, Y.\, Z = X \to Y,\ X = Y \rangle}\ fix$$

$$\left[\exists Z.\, Z = Q \to Q,\ \exists X, Y \left(Z = X \to Y,\ X = Y\right)\right]$$

↪ start at right side like a fold

try to eliminate X:
look for rules like "$X = t$", or "$t = X$"
then do substitution $[X/t]$ (here we have $X = Y$)

$$\left[\exists Z.\, Z = Q \to Q,\ \exists Y \left(Z = Y \to Y\right)\right]$$

can you eliminate Y here?
NO, because "$Y = t$" doesn't exist

$$\left[\exists Z, Y.\, \left(Z = Q \to Q,\ Z = Y \to Y\right)\right]$$

eliminate Z since $Z = Q \to Q$

$$\left[\exists Y.\ Q \to Q = Y \to Y\right]$$

← these 2 "$\to$" types match,

$[\exists Y.\ Q=Y,\ Q=Y]$

eliminate Y with $Q=Y$.

$[Q=Q,\ Q=Q]$

$[]$

You know the type of the root is

$fix\ (\lambda x.x) :: Q$

after unifying the constraints (say it produces a list of substitutions "subs"), we can apply "subs" to the variable Q

genTypeEqn (id :: UniqId, term :: Term)

↑ the type id of the current tree

← current term / tree you are on

- If term = A B (an application) then,

  lid = fresh unique type Id
  rid = fresh unique type Id

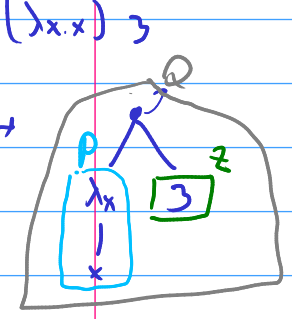
  ↳ left & right id

  ↳ use state monad.

  e1 = genTypeEqn (lid, A)

  e2 = genTypeEqn (rid, B)

  return ∃ rid, lid . id = rid → id
  , e1
  , e2



- If term = λ x. P is an abstraction then,

  argid = fresh unique type id
  bodyid = fresh unique type id

  ↳ new var type id

  symtab = current symbol table
  modify symbol table to include (x, argid)

  e = genTypeEqn(bodyid, P)

  ↳ P is type body id.

  reset the symbol table to symtab.

  return ∃ argid, bodyid . id = argid → bodyid
  , e

- If term = x is a variable, then

  lkup = lookup type id of x in the symbol table

  If lkup fails
  then: fail with free variable error
  else: return lkup = id

symbol table is (String, Uniq)

return pairs for unification.

TyTerm from var annotation

can ignore return (TyTerm () Uniq, TyTerm () Uniq)