CSE 114A Lecture 23

Agenda:
- More typing rules
- Unification

---

mini-Nano:

$e ::= n \mid b \mid x \mid e_1 + e_2 \mid \backslash x \to e \mid e_1\ e_2$

$\quad$ let $x = e_1$ in e_2

$T ::= \text{Int} \mid \text{Bool} \mid T_1 \to T_2$

$$\frac{}{\Gamma \vdash n :: \text{Int}}\ [\text{T-Int}] \qquad \frac{}{\Gamma \vdash b :: \text{Bool}}\ [\text{T-Bool}]$$

$$\frac{\Gamma \vdash e_1 :: \text{Int} \quad \Gamma \vdash e_2 :: \text{Int}}{\Gamma \vdash e_1 + e_2 :: \text{Int}}\ [\text{T-Add}]$$

$$\frac{(x, T) \text{ in } \Gamma}{\Gamma \vdash x :: T}\ [\text{T-Var}] \qquad \frac{\Gamma \vdash e_1 :: T_1 \quad (x, T_1) : \Gamma \vdash e_2 :: T_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 :: T_2}\ [\text{T-Let}]$$

Check the function body's type in an extended type environment.

$$\frac{(x, T_1) : \Gamma \vdash e :: T_2}{\Gamma \vdash \backslash x \to e :: T_1 \to T_2}\ [\text{T-Lam}]$$

$$\frac{\Gamma \vdash e_1 :: T_1 \to T_2 \quad \Gamma \vdash e_2 :: T_1}{\Gamma \vdash e_1\ e_2 :: T_2}\ [\text{T-App}]$$

P → Q P

Q

modus ponens!

Curry-Howard correspondence

types : propositions (formulas) :: programs : proofs

An expression e is well-typed in a type environment Γ if we can write down a typing derivation for

$$\Gamma \vdash e :: T$$

for some type T.

If we can't, the expression is ill-typed.

What if we tried to extend our 'infer' implementation to handle functions?

```
infer gamma (ELam x e) = t1 :=> t2
  where gamma' = (x, t1) : gamma
        t2 = infer gamma' e
        t1 = ... what do we put here? "
```

Spec!

$$\frac{(x, T_1) : \Gamma \vdash e :: T_2}{\Gamma \vdash \backslash x \to e :: T_1 \to T_2}\ [\text{T-Lam}]$$

$\backslash x \to x + \text{False}$

T_1 has to be Int!

$$\frac{\dfrac{(x, \text{Int}) \text{ in } [(x, \text{Int})]}{[(x, \text{Int})] \vdash x :: \text{Int}}\ [\text{T-Var}] \quad \checkmark \quad \dfrac{}{[(x, \text{Int})] \vdash 3 :: \text{Int}}\ [\text{T-Int}]}{[(x, \text{Int})] \vdash x + 3 :: \text{Int}}\ [\text{T-Add}]$$

$$\frac{[(x, \text{Int})] \vdash x + 3 :: \text{Int}}{[] \vdash \backslash x \to x + 3 :: \text{Int} \to \text{Int}}\ [\text{T-Lam}]$$

Unification

type variables: a, b, c

type substitution (sometimes just "substitution"): a list of pairs associating type variables with types

example:

$[(a, \text{Int}), (b, c \to c)]$

(a: type variable; Int: type; b: type variable; $c \to c$: type)

Applying a type substitution to a type means replacing all the type variables in the type with whatever the type substitution maps them to.

Quiz question 1:
What do you get if you apply the above substitution to the type

$$\text{Int} \to b \to b \quad ?$$

Answer: $\text{Int} \to (c \to c) \to (c \to c)$

$\text{Int} \to c \to c \to c \to c$

Unification (for this class, anyway) is finding a substitution that makes two types the same when it's applied to them both

$\text{Int} \to b \to b$ and $a \to d$

$\text{Int} \to (c \to c) \to (c \to c)$ and $\text{Int} \to (c \to c) \to (c \to c)$

$[(a, \text{Int}), (b, c \to c), (d, (c \to c) \to (c \to c))]$ unifies $\text{Int} \to b \to b$ and $a \to d$.

$\text{Int} \to b \to b$ and $a \to d$

$\text{Int} \to b \to b$ and $\text{Int} \to b \to b$

$[(a, \text{Int}), (d, b \to b)]$

a and $\text{Int} \to a$: These don't unify!

In general, you cannot unify a type variable with any type that contains free occurrences of that variable. ("occurs check")