

CSE114A Lecture 23

Agenda:

- More typing rules
- Unification

mini-Nano:

$$e ::= n \mid b \mid x \mid e_1 + e_2 \mid \lambda x \rightarrow e \mid e_1 \ e_2$$

$$\text{let } x = e_1 \text{ in } e_2$$

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

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

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

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

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

$$\frac{(x, T_1) : \Gamma \vdash e :: T_2}{\Gamma \vdash \lambda x \rightarrow e :: T_1 \rightarrow T_2} \text{ [T-lam]}$$

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

$$\frac{P \rightarrow Q \quad P}{Q} \text{ modus ponens!}$$

Curry-Howard correspondence

types : propositions :: programs : proofs (Formulas)

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 $x \ e$) = $t_1 \Rightarrow t_2$
 where gamma' = $(x, t_1) : \text{gamma}$
 $t_2 = \text{infer gamma}' \ e$
 $t_1 = \dots$ what do we put here? "Spec!"

$$\frac{(x, T_1) : \Gamma \vdash e :: T_2}{\Gamma \vdash \lambda x \rightarrow e :: T_1 \rightarrow T_2} \text{ [T-lam]}$$

T_1 has to be Int!

$\lambda x \rightarrow x + \text{false}$

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

Unification

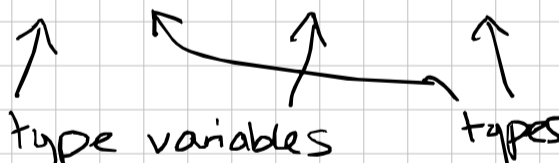
type variables a, b, c

sometimes just "substitution"

type substitution: a list of pairs associating type variables with types

example:

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



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

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

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

$$\text{Int} \rightarrow c \rightarrow c \rightarrow c \rightarrow c$$

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

$$\text{Int} \rightarrow b \rightarrow b \qquad \text{Int} \rightarrow (c \rightarrow c) \rightarrow (c \rightarrow c)$$

$$a \rightarrow d \qquad \text{Int} \rightarrow (c \rightarrow c) \rightarrow (c \rightarrow c)$$

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

unifies $\text{Int} \rightarrow b \rightarrow b$ and $a \rightarrow d$.

$$\text{Int} \rightarrow b \rightarrow b \qquad \text{Int} \rightarrow b \rightarrow b$$

$$a \rightarrow d \qquad \text{Int} \rightarrow b \rightarrow b$$

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

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")