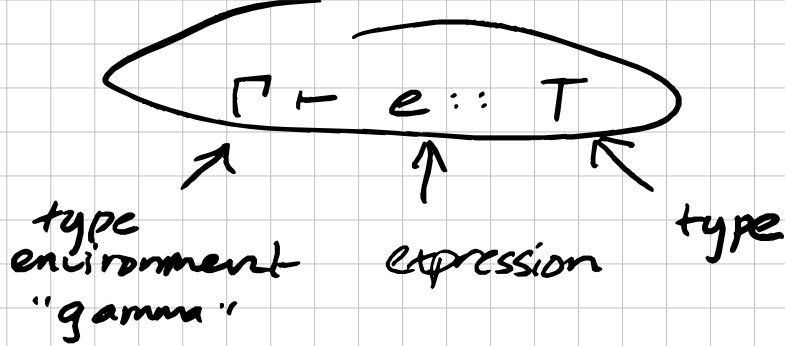


# CSE114A - Lecture 17

agenda:

- get into details of implementing part 3 of hw 5
- look at Nano programs that should type check, but don't with just the rules we've seen so far
  - ↑ polymorphism



$$\frac{n \in \mathbb{Z}}{\Gamma \vdash n :: \text{Int}} \quad \text{T-Num}$$

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

$$\frac{(x, T) \in \Gamma}{\Gamma \vdash x :: T} \quad \text{T-Var}$$

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

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

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

An expression  $e$  is well-typed in  $\Gamma$  if we can derive  $\Gamma \vdash e :: T$  for some  $T$ .

if we can't, the expression is ill-typed (or doesn't type check).

Let's do a typing derivation:

> typeOfString "let f =  $\lambda x \rightarrow x + 1$  in f 3"  
Int

$T_a$  and Int!  
 these must unify!  
 $\downarrow$                        $\downarrow$

$$\frac{(x, \text{Int}) \in [(x, \text{Int})] \quad 1 \in \mathbb{Z}}{\Gamma \vdash x :: \text{Int}} \quad \text{T-Var} \quad \checkmark$$

$$\frac{[(x, \text{Int})] \vdash x :: \text{Int} \quad [(x, \text{Int})] \vdash 1 :: \text{Int}}{[(x, \text{Int})] \vdash x + 1 :: \text{Int}} \quad \text{T-Add} \quad \checkmark$$

$$\frac{[(x, \text{Int})] \vdash x + 1 :: \text{Int}}{[] \vdash \lambda x \rightarrow x + 1 :: \text{Int} \rightarrow \text{Int}} \quad \text{T-Lam}$$

$T_2$  and Int  
 must unify  
 $\downarrow$                        $\downarrow$

$$\frac{(f, \text{Int} \rightarrow \text{Int}) \in [(f, \text{Int} \rightarrow \text{Int})] \quad 3 \in \mathbb{Z}}{[(f, \text{Int} \rightarrow \text{Int})] \vdash f :: \text{Int} \rightarrow \text{Int}} \quad \text{T-Var} \quad \checkmark$$

$$\frac{[(f, \text{Int} \rightarrow \text{Int})] \vdash f :: \text{Int} \rightarrow \text{Int} \quad [(f, \text{Int} \rightarrow \text{Int})] \vdash 3 :: \text{Int}}{[(f, \text{Int} \rightarrow \text{Int})] \vdash f 3 :: \text{Int}} \quad \text{T-App}$$

$$\frac{[] \vdash \lambda x \rightarrow x + 1 :: \text{Int} \rightarrow \text{Int} \quad [(f, \text{Int} \rightarrow \text{Int})] \vdash f 3 :: \text{Int}}{[] \vdash \text{let } f = \lambda x \rightarrow x + 1 \text{ in } f 3 :: \text{Int}} \quad \text{T-Let}$$

parametric  
polymorphism

$\lambda x \rightarrow x$  :: forall a.  $a \rightarrow a$

program inference  
given a type!

"Theorems for free!" - Phil Wadler

type inference

$\lambda x \rightarrow x$

:: forall a.  $a \rightarrow a$

let  $F = \lambda x \rightarrow x$  in  $\text{int} \rightarrow \text{int}$   
let  $b = F\ 3$  in  $\text{Bool} \rightarrow \text{Bool}$   
let  $c = F\ \text{True}$  in  $\text{Bool}$   
 $c \parallel b < 5$   
↑ boolean or      ↑ comparing numbers

let  $F = \lambda x \rightarrow x$  in

let  $y = \boxed{f\ 5}$  in

$f(\lambda z \rightarrow z+y)$

Should have type

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

but how can we make sure to infer that type?

$\boxed{f\ 5}$

would seem to impose a constraint that  $f$  has type  $\text{Int} \rightarrow \text{Int}$

$\boxed{f(\lambda z \rightarrow z+y)}$

would seem to impose a constraint that  $f$  has type  $(\text{Int} \rightarrow \text{Int}) \rightarrow (\text{Int} \rightarrow \text{Int})$

How to deal with this:

$f$ 's real type is  $\forall a. a \rightarrow a$ .  
(a polytype)

we can instantiate a polytype with different types, by replacing the bound type variable with some type.

At a high level, for this example:

- when you have to pick a type for some program variable, don't!  
pick a fresh type variable.
- So the type of  $\lambda x \rightarrow x$  comes out  $a \rightarrow a$ .  
↳ ELet case of infer
- generalize this to  $\forall a. a \rightarrow a$  and put the generalized version of it into the type environment.
- when you use  $f$ , in other words, when you need to 'get the type of  $f$  out of the type environment, instantiate it.  
↳ EVar case of infer

(All of this is going to be useful on 3(b) of HW5.)