Idenitity : $f(x) = x$

ISWIM : $\lambda x.\ x$

$(\lambda x. x)\ 5 \longmapsto 5$

$(\lambda x. x)\ (\lambda y.\ y+2) \longmapsto (\lambda y.\ y+2)$

Typed-ISWIM: $\lambda x{:}int.\ x$

$\lambda x{:}bool.\ x$

$\lambda x{:}(int \to bool).\ x$

...

$(5+2)$ — abstract away '5' → $(\lambda y.\ y+2)\ 5$

$(\lambda x{:}int. x)$ — abstact away 'int' → $(\lambda t.\ \lambda x{:}t.\ x)$ int

$\underbrace{\lambda t{:}\omega \quad type_0 : type_1 : \ldots}_{Coq}$

$(\Lambda T.\ \lambda x{:}T. x)[int]$

embed λ-calculus inside of Typed-ISWIM's type system

$T ::= B \mid T \to T$ $\qquad$ $M ::= N \mid M + M$

$\Downarrow$ $\qquad$ $\Downarrow$

$T ::= B \mid T \to T \mid (\forall \alpha. T) \mid \alpha$ $\qquad$ $M ::= N \mid M + M \mid \lambda x. M \mid x \mid (M\ M)$

$M ::= b \mid \lambda x{:}T. M \mid \Lambda \alpha. M \mid x \mid M[T]$

$(\Lambda \alpha.\ \lambda x{:}\alpha.\ x) : \forall \alpha.\ \alpha \to \alpha$

$\beta_T = (\Lambda \alpha. M)[T] \longrightarrow M[\alpha \leftarrow T]$ (a type abstraction is instantiated with subst.)

β_V = normal value rules

23-2

$$\frac{\Gamma, \alpha \vdash M : T}{\Gamma \vdash \Lambda\alpha. M : \forall\alpha. T}$$

← may mention alpha

Polymorphism

substitution at the type-level

$$\frac{\Gamma \vdash M : \forall\alpha. T'}{\Gamma \vdash M[T] : T'[\alpha \leftarrow T]}$$

Java: generics

C++: templates

class $\underbrace{\text{List<X>}}$ { X val; ... }

Λ X.
 class List { ...
  X val; ..
 }

$\underbrace{\text{List<cat> L}}$ = ...

⊗ [cat]

Java:

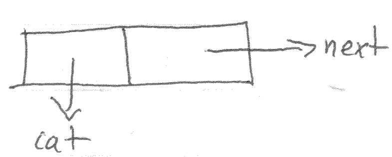
class List < X implents Comparable >

(F-bound polymorphism)

$T ::= \ldots \mid \forall\alpha \underbrace{<: T}_{\text{must be comparable}} . T'$

Java

- $\longrightarrow = (B_T \cup B_V)^*$
- interleaves type app with normal running
- Only 1 copy ever
  ↳ must use a pointer

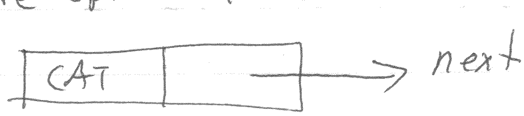


C++

- $\longrightarrow = B_T^* \circ B_V^*$
- puts all type app first
- List<X> includes length

Program uses List<Cat>, List<Dog>
TWO COPIES
IN BINARY + RUNTIME

- More optimizations



23-3

$M = \ldots \mid \lambda x : T.\ M$

explicit type annotation

$$\frac{\Gamma[x:T] \vdash M : T'}{\Gamma \vdash \lambda x:T.\ M : T \to T'}$$

we know T $(O(m))$

$$\frac{\Gamma[x:T] \vdash M : T'}{\Gamma \vdash \lambda x.\ M : T \to T'}$$

guess T $(O(2^{|m|}))$

$\lambda x.\ x + 2$

number

$\Gamma \vdash M : T$

type env. proves a program has type

$\Gamma \vdash M : T \mid C ; V$

usage constraints — variables used

$V = X \ldots$

$C = (T = T) \ldots$

$$\frac{\Gamma[x \mapsto \alpha] \vdash M : T' \mid C ; V}{\Gamma \vdash (\lambda x.m) : \alpha \to T' \mid C ; V \cup \{\alpha\}}$$

$$\frac{\Gamma \vdash M : T_1 \mid C_1 ; V_1 \qquad \Gamma \vdash N : T_2 \mid C_2 ; V_2}{\Gamma \vdash M + N : num \mid C_1 \cup C_2 \cup \{T_1 = num, T_2 = num\} ; V_1 \cup V_2}$$

$$\frac{(x, T) \in \Gamma}{\Gamma \vdash x : T \mid \emptyset ; \emptyset}$$

$$\frac{\Delta(b) = B}{\Gamma \vdash b : B \mid \emptyset ; \emptyset}$$

$$\frac{\Gamma \vdash M : T_1 \mid C_1 ; V_1 \qquad \Gamma \vdash N : T_2 \mid C_2 ; V_2}{\Gamma \vdash (M\ N) : \alpha \mid C_1 \cup C_2 \cup \{T_1 = T_2 \to \alpha\} ; V_1 \cup V_2 \cup \{\alpha\}}$$

23-4

After type-checking :

T - the result type

C - the constraints

V - the variables

$(\alpha, \{ T_1 = T_2 \to \alpha, T_1 = \underset{\to num}{num}, T_2 = num \},$
$\{T_1, T_2, \alpha\})$

$\rightsquigarrow$ $\alpha = num$, your program returns a number

A solution is called a substitution (σ)
$\sigma = (X = T) \ldots$

$U : C \times \sigma \to \sigma$ or FAIL

$U(\emptyset, \sigma) = \sigma$
$U((C, X=T), \sigma) = \cancel{U(C, (X=T) \cup \sigma)}$
$U(C \cup (T=X), \sigma) = \cancel{U(C, (X=T) \cup \sigma)}$
$U(C[X \leftarrow T],$
$(X=T) \cup \sigma[X \leftarrow T])$

$U(C \cup (T_1 \to T_2 = T_3 \to T_4), \sigma) =$
$U(C \cup (T_1 = T_3) \cup (T_2 = T_4), \sigma)$

$U(C \cup (T=T), \sigma) = U(C, \sigma)$

$U(C \cup (T_1 = T_2), \sigma) = FAIL$
T_1 and T_2 are concrete types (not X)
Unification that are not equal (Num = Bool)
(Num = X → Y)

$X = T$

23-5

$\mathcal{U}(\{T_1 = T_2 \to \alpha,\ T_1 = num \to num,\ T_2 = num\},\ \emptyset)$

$= \mathcal{U}(\{T_2 \to \alpha = n \to n,\ T_2 = N\},\ \{T_1 = T_2 \to \alpha\})$

$= \mathcal{U}(\{T_2 = N,\ \alpha = N,\ T_2 = N\},\ \{T_1 = T_2 \to \alpha\})$

$= \mathcal{U}(\{\alpha = N,\ N = N\},\ \{T_1 = N \to \alpha,\ T_2 = N\})$

$= \mathcal{U}(\{N = N\},\ \{T_1 = N \to N,\ T_2 = N,\ \alpha = N\})$

$= \mathcal{U}(\ \emptyset\ ,\ \text{''}\ )$

$= \{T_1 = N \to N,$
$\quad T_2 = N,$
$\quad \alpha = N\}$

constraint generation

$$\text{type-check}(M) = \frac{\boxed{\emptyset \vdash M : T \mid C ; V}}{\mathcal{U}(C, \emptyset)[T]} = \text{the type}$$

constraint solving

Type Inference

What if $X = X \to X \in C$?
we must add a rule that say

$\mathcal{U}(C \cup (X = T),\ \sigma)$
X must not occur inside of T

$\mathcal{U}(\Omega,\ \emptyset) =$ diverges