

($\forall A, T$)
→

A is unknown

Set of Numbers

- ↳ Array/List of members
- ↳ Map of (member \rightarrow bool)
- ↳ Tree of members
- ↳ n-bit bitmask

Interface Virtual Abstract Class

- + type of set
- mem : set x num \rightarrow bool
- add : set x num \rightarrow set
- rem : set x num \rightarrow set
- mt : set

class U Σ

void f (SetOfNumbers s) {
s.mem (s.add (s.mt, 1), 1) == true };

Interface (Representation) ^{$\leftarrow R$}

= (R x (R x Num \rightarrow Bool) x (R x Num \rightarrow R))

BitClass : Interface (R = Num)

= (Num x (Num x Num \rightarrow Bool) x (Num x Num \rightarrow Num))
 \emptyset $\left((1 \ll n) \oplus 1 \right)$ $n \mid (1 \ll n)$

MapClass : Interface (R = Map)

= (Map x (Map x Num \rightarrow Bool) x (Map x Num \rightarrow Map))

f : $\forall R$. Interface (R) \rightarrow Bool

= $\forall R$. (R x (R x Num \rightarrow Bool) x (R x Num \rightarrow R))
 \rightarrow Bool

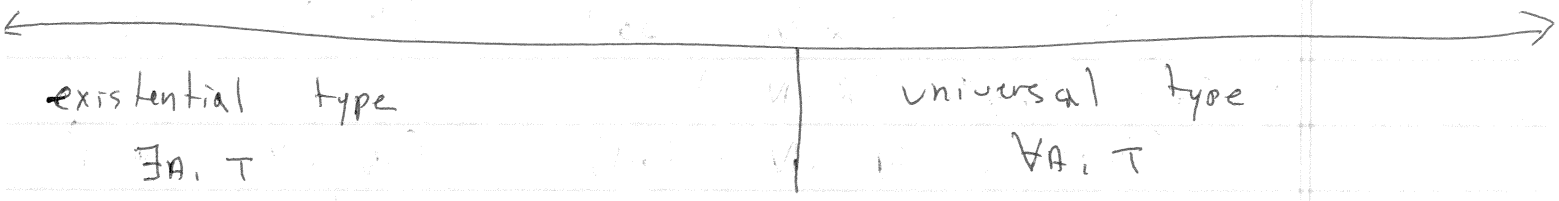
((fst MapClass) map-count) \Rightarrow Num

Library = Interface (concrete R)
 Client = $\forall R. (\text{Interface } (R) \rightarrow \text{Answer})$
 └ trusts clients

Client(A) = $\forall R. (\text{Interface } (R) \rightarrow \text{Answer})$ doesn't trust
 Library' = $\forall A. (\text{Client } (A) \rightarrow A)$ clients

MapClass client :=
 (client [R=Map] (empty-map x map-member? x map-add))

Program :=
 (MapClass [A=Bool] (lambda (si)
 ((snd si) ((third si) (fst si) 1) 1))
)) (set-member? (set-add set-mt 1) 1)



MapClass : $\exists R. \text{Interface } (R)$
 ListClass : $\exists R. \text{Interface } (R)$
 BitClass : $\exists R. \text{Interface } (R)$

unpack [R] SetInstance from MapClass in code that calls get funcs

T = ... | $\exists A. T$

M = ... | pack [A=T] M as T'
 name it this 3
 hide this 2
 run this and get a value 1
 new exposed type 4

pack [A=Map] MapClass as (Ax (AxNum -> Bool) x (AxNum -> A))

unpack [A] X from M in M'
 expose the hidden type as A 2
 name it this 3
 run this 4
 run this with x bound to expose value 4

$V = \dots \mid \text{pack } [A=T] \ V \text{ as } T'$
 $E = \dots \mid \text{pack } [A=T] \ E \text{ as } T'$
 $\quad \mid \text{unpack } [A] \ X \text{ from } E \text{ in } M$

$E \left[(\text{unpack } [A] \ X \text{ from } (\text{pack } [A'=T'] \ V \text{ as } T') \text{ in } M) \right]$
 $\mapsto E \left[M \left[X \leftarrow V \right] \left[A \leftarrow T' \right] \right]$

$\Gamma \vdash M : T \left[A \leftarrow T' \right]$
 $\Gamma \vdash T'$
 $\Gamma, A \vdash T$

$\Gamma \vdash \text{pack } [A=T'] \ M \text{ as } T : \exists A, T$

$\Gamma \vdash M_1 : \exists A', T'$
 $\Gamma, X : T' \left[A' \leftarrow A \right] \vdash M_2 : T$
 $A \notin FV(T)$

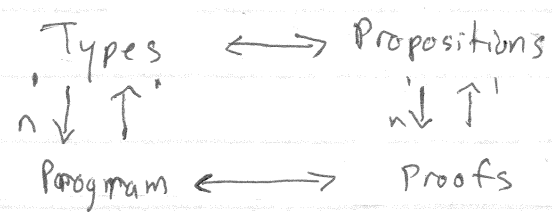
$\Gamma \vdash \text{unpack } [A] \ X \text{ from } M_1 \text{ in } M_2 : T$

$\text{unpack } [A] \ X \text{ from}$
 $\quad (\text{pack } [A' = \text{Num}] \ M \text{ as } A') \quad = \exists A', A'$
 in
 $\quad X : A$

$\forall \beta \dots \lambda x : T_\beta \dots \exists \alpha \dots T_{\alpha, \beta}$

β is the imported abstract types
 T_β is the modules imported that use β
 α are your exported abstract types
 $T_{\alpha, \beta}$ are the exported values (mention α or β)

- T = Num | Bool | base-types
- | $T \rightarrow T$
- | $T \times T$ ($T \wedge T$)
- | $T + T$ ($T \vee T$)
- | $\forall A, T$
- | A
- | $\exists A, T$
- | $\mu A, T$



Curry-Howard Isomorphism

COMP 3010 - 202 Mc Carthy