OO in a type system

$List\ \alpha = \mu l.\ 1 + (\alpha \times l)$

$A\ Queue\ \alpha = List\ \alpha \times List\ \alpha$

$enq(a, (f, b)) = (f, a :: b)$: $\alpha \to Q\alpha \to Q\alpha$

$deq((a :: f, b)) = (a, (f, b))$: $Q\alpha \to (\alpha, Q\alpha)$

$deq((\,[\,], b)) = deq((rev\ b, [\,]))$

$mk() = ([\,], [\,])$

$enq(2, (1 :: [\,], [\,]))$

Types don't provide user-abstractions
i.e. no user-enforced invariants

$Bin_N = 1 + (Bin_N, N, Bin_N)$

$BST_N = Bin_N$, where the BST property holds

$(L, N, R) \in BST_N$ if $\forall n \in L,\ n \le N$

$\forall n \in R,\ N \le n$

$ins : num \times BST_N \to BST_N$

$ins(n, tt) = (tt, n, tt)$

$ins(n, (L, v, R)) =$

if $n \le v$, $(ins(n, L), v, R)$

o.w., $(L, v, ins(n, R))$

$messup : num \times BST_N \to BST_N$

$messup(n, T) = (tt, n, T)$

How can types HIDE data?

$M : \forall\alpha.\ \alpha \to \alpha$ implies $M = \Lambda\alpha.\ \lambda(x : \alpha).\ x$

(x: provided by the server; x: client of α)

client is the user of hidden data

server is the provider of hidden data
who can see inside

$$BST\text{-}Server = ((1 \to BIN_N) \quad \text{— newbst}$$
$$\times (N \times BIN_N \to Bool) \quad \text{— member}$$
$$\times (N \times BIN_n \to BIN_N)) \quad \text{— insert}$$

$$BST\text{-}Client = \forall R. ((1 \to R)$$
$$\times (N \times R \to Bool)$$
$$\times (N \times R \to R)) \to Ans$$

example1 = $\Lambda R. \lambda (s:R).$
((2nd s) 5 ((3rd s) 5 ((fst s) tt)))
(member? 5 (ins 5 (new-bst))) $\equiv$ true

Clients are polymorphic over server representations

~~Restrictions~~

Server protection relies on clients using this pattern.

server = ((A → B) * where Rep is in A or B)

client = $\forall R.$ server $[Rep \leftarrow R]$

Workable with pattern enforcement

GOAL: servers protecting themselves

$T = \ldots \mid \exists X.\, T$

$M = \ldots$ (vtable)
$\mid$ pack $[A=T]\ M$ as T' (construct $\exists$)
$\mid$ unpack $[A]\ X$ from M in M'

Server: $\exists$ REP.
((I → REP)
× (N × REP → B)
× (N × REP → REP)

client : server → ans

$V = \ldots \mid$ pack $[A=T]\ V$ as T
$E = \ldots \mid$ pack $[A=T]\ E$ as T'
$\mid$ unpack $[A]\ X$ from E in M

$E[$ unpack $[A]\ X$ from (pack $[A'=T']\ V$ as T) in $M]$
$\longmapsto$ $E[\ m\ [X \leftarrow V]\ [A \leftarrow T']\ ]$
(client body; vtable subst; representation subst)

$\Gamma \vdash M : T[A \leftarrow T']$
$\Gamma \vdash T'$ is valid type
$\Gamma, A \vdash T$ is valid type

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

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

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

$\forall B \ldots \lambda(x : T_B) \ldots \exists \alpha \ldots T_{\alpha, B}$
B is imported abstract types
T_B are the server impls that use B
α are the exported abstract types
$T_{\alpha, B}$ is the client code that uses α and B

```
class NAME {
  NAME (Cargs ...){}
  mIr methI (Cargs ...) {}
  ...
  mNr methN ( mN args...){}
}
```

↓

```
NAME =
  (Cargs ....)
-> ∃R. ( (R, mIargs...)-> mIr)
       ...
       (R, mNargs...)-> mNr))
```