26-1

```
let id = λx. x in
if (id true)
   (id 5)
   (id 6)
```

$\longrightarrow$ $X = Bool$, $X = Num$

$$\text{let } X = \cancel{M}\, M \text{ in } N \;=\; (\lambda X.\, N)\, M$$

"let-polymorphism"

$$\frac{\Gamma \vdash N[X \leftarrow M] : T_N ; C_N ; X_N}{\Gamma \vdash \text{let } X = \underbrace{M}_{} \text{ in } N : T_N ; C_N ; X_n}$$

↳ must be a value if the language has mutation

$T_1 = T_1 \to X$

$T = B \mid 1 \mid 0 \mid T + T \mid T \times T \mid T \to T \mid \mu A.\, T \mid \forall A.\, T \mid A$

$f\, \underbrace{x}_{} = x$

↳ fixed

$nat = 1 + nat$

$nat = \mu A.\, 1 + A$

$\mu A.\, T = \underbrace{tfix}_{} \, (\forall A.\, T)$

↳ computes the fixed-point

$1 + nat = nat?$ $\quad (1 + (\mu A.\, \cancel{A} + A)) = (\mu A.\, 1 + A)$?

$$\frac{\Gamma \vdash M : T_M \qquad \Gamma \vdash N : T_N \qquad \vdash T_M \leftrightarrow T_N \to T_A}{\Gamma \vdash (M\ N) : T_A}$$

$$\frac{\vdash T_A[A \leftarrow C] \leftrightarrow T_B[B \leftarrow C]}{\vdash \mu A.\, T_A \leftrightarrow \mu B.\, T_B}$$

$$\frac{\vdash T_A[A \leftarrow \mu A.\, T_A] \leftrightarrow T'}{\vdash \mu A.\, T_A \leftrightarrow T'}$$

$$\frac{\vdash T' \leftrightarrow T_A[A \leftarrow \mu A.\, T_A]}{\vdash T' \leftrightarrow \mu A.\, T_A}$$

Equi-recursive types

- A recursive type is equal to its unfolding (infinite)
- No algorithm for when to use the unfolding rule
- Type inference is undecidable

Iso-recursive Types

$M = \ldots \mid (\text{fold } M) \mid (\text{unfold } M)$

$V = \ldots \mid (\text{fold } V)$

$E = \ldots \mid (\text{fold } E) \mid (\text{unfold } E)$

$$E[(\text{unfold } (\text{fold } V))] \longrightarrow E[V]$$

$$\frac{\Gamma \vdash M : T[A \leftarrow \mu A. T]}{\Gamma \vdash \text{fold } M : \mu A. T} \qquad \frac{\Gamma \vdash M : \mu A. T}{\Gamma \vdash \text{unfold } M : T[A \leftarrow \mu A. T]}$$

List X = Mt | Cons X (List X)

(Cons 1 (Cons 2 Mt)) $\quad$ Cons : $\forall X.\ X \to \text{List } X \to \text{List } X$

$\text{cons} : \forall \alpha.\ \alpha \to (\mu L.\ 1 + (\alpha \times L)) \to (\mu L.\ 1 + (\alpha \times L))$

$= \Lambda \alpha.\ \lambda a : \alpha.\ \lambda r : (\mu L.\ (1 + (\alpha \times L))).$

$\quad \text{fold } L \ (\text{inr } (\text{pair } a\ r))$

constructors use fold

destructors use unfold

$\text{firstor} : \forall \alpha.\ (\mu L.\ 1 + (\alpha \times L)) \to \alpha \to \alpha$

$\Lambda \alpha.\ \lambda l : (\mu L.\ (1 + (\alpha \times L))).\ \lambda d : \alpha.$

```
case (unfold l) with
  inl unit => d
  inr (pair a r) => a
```

$\forall A, T$ means ...

the provider doesn't know what A is

map : $\forall A. \forall B. \; (L\ A) \to (A \to B) \to (L\ B)$

$\forall$ is for servers of general behavior specialized to consumer

---

data abstraction is when a server of specific behavior hides from the consumer

" $\exists A, T$ " means ...

the consumer doesn't know what A is

server : client $\to$ client's answer

$$\forall ANS. \; (\underbrace{\qquad\qquad\qquad \to ANS}) \to ANS$$

client : server object $\times$ server methods $\to$ client answer

$$\forall SERVER. \; (SERVER \times (SERVER \to Num)) \to ANS$$

$T = \ldots \mid \exists A, T$ $\qquad$ $M = \ldots \mid \text{pack } [A = T]\ M \text{ as } T'$

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

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

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

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

$$\frac{\Gamma \vdash M_1 : \exists A', T' \qquad \Gamma, X : T'[A' \leftarrow A] \vdash M_2 : T \qquad A \notin FV(T)}{\Gamma \vdash \text{unpack } [A]\ X \text{ from } M_1 \text{ in } M_2 : T}$$

26-4

Modular Programs

$$\forall \underbrace{\beta \ldots}_{\text{imports to the module}} \lambda x : \underbrace{T_\beta}_{\text{actual imported funs \& values}} \ldots \exists \underbrace{\alpha,}_{\text{types of exports}} \ldots \underbrace{T\,\alpha, \beta}_{\text{exports value}}$$