An Alternate Presentation

$$A ::= A \to B \mid \Box A \mid \odot A \mid 1 \mid A + B \mid A \times B \mid \underbrace{\mu\alpha.\, A}_{\text{recursive types!}}$$

$$q ::= \text{now} \mid \text{stable} \mid \text{later}$$

$$\Theta ::= \cdot \mid \Theta, x : A_q$$

$$\Theta \vdash \underbrace{e : A_q}_{\text{Typing judge parameterized by mode}}$$

Modal FRP

$$\frac{x : A_q \in \Theta \quad q \in \{now, stable\}}{\Theta \vdash x : A \text{ ok}}$$

$$\frac{\Theta, x : A \text{ ok} \vdash e : B \text{ ok}}{\Theta \vdash \lambda x. e : A \to B \text{ ok}} \qquad \frac{\Theta \vdash e_1 : A \to B \text{ ok} \quad \Theta \vdash e_2 : A \text{ ok}}{\Theta \vdash e_1\, e_2 : B \text{ ok}}$$

$$\frac{\Theta \vdash e : A \text{ stable}}{\Theta \vdash \text{box}(e) : \Box A \text{ ok}} \qquad \frac{\Theta \vdash e_1 : \Box A \text{ ok} \quad \Theta, x : A \text{ stable} \vdash e_2 : B \text{ ok}}{\Theta \vdash \text{let box}(x) = e_1 \text{ in } e_2 : B \text{ ok}}$$

$$\frac{\Theta^{\Box} \vdash e : A \text{ ok}}{\Theta \vdash e : A \text{ stable}}$$

$$\left.\begin{aligned}
(\cdot)^{\Box} &= \cdot \\
(\Theta, x : A \text{ ok})^{\Box} &= \Theta^{\Box} \\
(\Theta, x : A \text{ stable})^{\Box} &= \Theta^{\Box}, x : A \text{ stable} \\
(\Theta, x : A \text{ later})^{\Box} &= \Theta^{\Box}
\end{aligned}\right\}$$

Delete the unstable variables

Sums and Products

$$\frac{\Theta \vdash e : A_i \text{ ok}}{\Theta \vdash in_i(e) : A_1 + A_2 \text{ ok}} \qquad \frac{\Theta \vdash e_1 : A \text{ ok} \quad \Theta \vdash e_2 : B \text{ ok}}{\Theta \vdash (e_1, e_2) : A \times B \text{ ok}}$$

$$\frac{}{\Theta \vdash () : 1 \text{ ok}} \qquad \frac{\Theta \vdash e : A_1 \times A_2 \text{ ok}}{\Theta \vdash \pi_i(e) : A_i \text{ ok}}$$

$$\frac{\Theta \vdash e : A_1 + A_2 \text{ ok} \qquad \Theta, x_1 : A_1 \text{ ok} \vdash e_1 : B \text{ ok} \qquad \Theta, x_2 : A_2 \text{ ok} \vdash e_2 : B \text{ ok}}{\Theta \vdash \text{case}(e, \ in_1 x_1 \to e_1, in_2 x_2 \to e_2) : B}$$

The Later Modality

$$\frac{\Theta \vdash e : A \text{ later}}{\Theta \vdash \delta(e) : \bullet A}$$

$$\frac{\Theta \vdash e_1 : \bullet A \text{ ok} \qquad \Theta, x : A \text{ later} \vdash e_2 : B \text{ ok}}{\Theta \vdash \text{let } \delta(x) = e_1 \text{ in } e_2 : B \text{ ok}}$$

$$\frac{\Theta^{\bullet} \vdash e : A \text{ ok}}{\Theta \vdash e : A \text{ later}}$$

$$\boxed{\begin{aligned}
(\cdot)^{\bullet} &= \cdot \\
(\Theta, x : A \text{ stable})^{\bullet} &= \Theta^{\bullet}, x : A \text{ stable} \\
(\Theta, x : A \text{ later})^{\bullet} &= \Theta^{\bullet}, x : A \text{ now} \\
(\Theta, x : A \text{ now})^{\bullet} &= \Theta^{\bullet}
\end{aligned}}$$

Recursive Types

$$\frac{\Theta \vdash e : [\bullet\mu\alpha.A/\alpha]A \ \text{ok}}{\Theta \vdash \mathrm{in}(e) : \mu\alpha.A \ \text{ok}} \qquad \frac{\Theta \vdash e : \mu\alpha.A \ \text{ok}}{\Theta \vdash \mathrm{out}(e) : [\bullet\mu\alpha.A/\alpha]A \ \text{ok}}$$

$$\frac{\Theta^{\square}, x : A \ \text{later} \vdash e : A \ \text{ok}}{\Theta \vdash \mathrm{fix}\ x{:}A.\ e : A \ \text{ok}}$$

$$\mu\alpha.\ A\times\alpha \rightsquigarrow A\times\bullet(\mu\alpha.\ A\times\alpha) \rightsquigarrow A\times\bullet(A\times\bullet(\mu\alpha.A\times\alpha))$$

What the Modalities Mean

$\square A$ are stable values

- They do not change as time passes
- Stable variables are persistent:
  once available, always available

$\bullet A$ are "As tomorrow"

- They are lazy, and scheduled for the next tick
- Later variables cannot be used now,
  but turn into ok variables in one tick

Eta and $\Box$

$\text{bool} = 1 + 1 \qquad \text{true} = in_1() \qquad \text{false} = in_2()$

$\text{promote} : \text{bool} \to \Box\,\text{bool}$

$\text{promote}\ b = \text{case}(b,$
$\quad in_1\ \_ \to \text{box}(\text{true}),$
$\quad in_2\ \_ \to \text{box}(\text{false}))$

$U \to \Box U$ definable for

$U ::= 1 \mid U + U \mid U \times U \mid \Box A$

Guarded Recursion

$\mathrm{fix}\ x : A.\ e$ is guarded recursion

- It is a recursive definition, but the recursive call is only on the next tick
- Works well with guarded recursive types

Guarded Recursive Types

- Each recursive occurence is later

- $S(A) = \mu\alpha.\, A \times \alpha$

$$\frac{\Theta \vdash e : S(A)}{\Theta \vdash \mathrm{out}(e) : A \times \bullet S(A)}$$

$$S(A) = \mu\alpha.\, A \times \alpha$$

$$[\bullet\, \mu\alpha.(A \times \alpha) / \alpha](A \times \alpha)$$

$$= A \times \bullet\, \mu\alpha.(A \times \alpha)$$

$$= A \times \bullet\, S(A)$$

Example: Streams

$S(A) \triangleq \mu\alpha.\, A \times \alpha$

$$\mathrm{cons} : A \to \bullet SA \to SA$$
$$\mathrm{cons}\ a\ as = \mathrm{in}\,(a, as)$$

$\mathrm{head} : S(A) \to A$

$\mathrm{head}\ xs = \pi_1(\mathrm{out}\ xs)$

$\mathrm{tail} : S(A) \to \bullet SA$

$\mathrm{tail}\ xs = \pi_2(\mathrm{out}\ xs)$

$\mathrm{map} : \square(A \to B) \to SA \to SB$

$\mathrm{map}\ (\mathrm{box}(f)) =$

$\quad \mathrm{fix}\ r : SA \to SB.\ \lambda xs : SA.$

$\quad\quad \mathrm{let}\ y = f(\mathrm{head}\ xs)$

$\quad\quad \mathrm{let}\ \delta(xs) = \mathrm{tail}\ xs$

$\quad\quad \mathrm{cons}(y, \delta(r\ xs))$

The recursive call

More Examples

$$\text{accum} : S(\text{int}) \to \text{int} \to S(\text{int})$$

```
accum ns acc =
    let box(a) = promote acc
    let box(n) = promote (head ns)
    let δ(xs') = tail(xs)
    cons(a+n , δ(accum xs' (a+n)))
```

$$\text{accum } [0,1,2,\ldots]\ 0 = [0,1,3,6,10,\ldots]$$

More Examples

accum : S(int) → int → S(int)

accum ns acc =
  let box(a) = promote n
  let box(n) = promote (head ns)
  let δ(xs') = tail(xs)
  cons($\underline{a+n}$, δ(accum xs' $\underline{(a+n)}$))

- a,n used now and later
- must be stable

More Examples

$$\text{tails} : S(A) \to S(S(A))$$

$$
\begin{aligned}
&\text{tails } xs = \\
&\quad \text{let } \delta(xs') = \text{tail}(xs) \\
&\quad \text{cons}(xs, \delta(\text{tails } xs'))
\end{aligned}
$$

$$
\begin{aligned}
\text{tails } [1, 2, 3, 4, \ldots] = [\ &[1, 2, 3, 4, \ldots], \\
&[2, 3, 4, 5, \ldots], \\
&[3, 4, 5, 6, \ldots], \\
&\ldots \\
]&
\end{aligned}
$$

Unfold for Streams

$$\mathrm{unfold} : \square(X \longrightarrow B \times \bullet X) \to X \longrightarrow S(B)$$

$$\begin{aligned}
&\mathrm{unfold}\ (\mathrm{box}\ f)\ x = \\
&\quad \mathrm{let}\ (b, \delta(x')) = f(x) \\
&\quad \mathrm{cons}\,(b, \delta(\mathrm{unfold}\ (\mathrm{box} f)\ x'))
\end{aligned}$$

Switching

$$\text{switch} : \square\,\text{int} \to S(A) \to S(A) \to S(A)$$

```
switch (box n) xs ys =
  if n=0 then
     ys
  else
    let (x :: xs') = xs
    let (y :: ys') = ys
    cons (x , δ(switch (box (n-1)) xs' ys')
```

Switching

$$\text{switch} : \square\,\text{int} \to S(A) \to S(A) \to S(A)$$

```
switch (box n) xs ys =
  if n=0 then
     ys
  else
    let (x :: xs') = xs
    let y          = head (ys)
    let δ(ys')  =  tail (ys)
    cons (x , δ(switch (box (n-1)) xs' ys')
```

Events

A stream $S(A)$ yields an infinite stream of As

Some things happen once

$$E(A) = \mu\alpha.\, A + \alpha$$

$$\left(\begin{array}{l} E(A) = \text{Done of } A \\ \quad\quad\;\; \mid \text{Wait of } \bullet E(A) \end{array} \right)$$

Events

$$E(A) = \mu\alpha.\ A + \alpha$$

$$\left(\begin{aligned} E(A) &= \text{Done of } A \\ &\mid \text{Wait of } \bullet E(A) \end{aligned} \right)$$

$$\text{return} : A \to E(A)$$
$$\text{return } x = \text{Done}(x)$$

$$\text{bind} : E(A) \to \square(A \to E(B)) \to E(B)$$
$$\text{bind } (\text{Done } x)\ (\text{box } f) = f\ x$$
$$\text{bind } (\text{Wait } \delta(e'))\ (\text{box } f) = \text{Wait}(\delta(\text{bind } e'\ (\text{box } f)))$$

Events

$$E(A) = \mu\alpha.\ A + \alpha$$

$$\begin{aligned} E(A) &= \text{Done of } A \\ &\mid \text{Wait of } \bullet E(A) \end{aligned}$$

$$\text{map} : \Box(A \to B) \to E(A) \to E(B)$$

$$\text{map}\ (\text{box } f)\ (\text{Done } a) = \text{Done}\ (f\ a)$$

$$\text{map}\ (\text{box } f)\ (\text{Wait } \delta(e)) = \text{Wait } \delta(\text{map}\ (\text{box} f)\ e)$$

Events

$$E(A) = \mu\alpha.\, A + \alpha \qquad\qquad E(A) = \mathrm{Done\ of\ } A \;\mid\; \mathrm{Wait\ of} \bullet E(A)$$

$$\mathrm{select} : E(A) \to E(B) \to E((A \times E(B)) + (E(A) \times B))$$

$$\mathrm{select}\ (\mathrm{Done}\ a)\ e_2 = \mathrm{Done}(in_1(a, e_2))$$

$$\mathrm{select}\ e_1\ (\mathrm{Done}\ b) = \mathrm{Done}(in_2(e_1, b))$$

$$\mathrm{select}\ (\mathrm{Wait}\ \delta(e_1'))\ (\mathrm{Wait}\ \delta(e_2')) = \mathrm{Wait}(\delta(\mathrm{select}\ e_1'\ e_2'))$$

Recursion

Let $X = \mu\alpha.\ \Box(\alpha \to A)$

$\mathrm{selfapp} : (\bullet A \to A) \to X \to A$

$\mathrm{selfapp}\ f\ v =$

$\quad \mathrm{let}\ \mathrm{box}(\omega : X \to A) = \mathrm{out}(v)\ \mathrm{in}$

$\quad f\ \delta(\omega\ (\mathrm{in}(\mathrm{box}\ \omega)))$

$\mathrm{fix} : \Box(\bullet A \to A) \to A$

$\mathrm{fix}\ (\mathrm{box}\ f) =$

$\quad \mathrm{selfapp}\ f\ (\mathrm{in}(\mathrm{box}\ (\mathrm{selfapp}\ f)))$

Variant of the Y combinator

Operational Semantics

- Values of $\bullet A$ are lazily evaluated
- We must preserve sharing for efficiency
- We implement $\bullet A$ with a memoized code pointer

Operational Semantics

Store $\sigma ::= \cdot \mid \sigma, \ell : v \text{ now} \mid \sigma, \ell : e \text{ later} \mid \sigma, \ell : \text{null}$

Values $v ::= () \mid (v, v') \mid \text{in}_i\, v \mid \lambda x : A.\, e \mid \text{in}\, v$
$\mid \text{box}(v)$
$\mid \ell$

Locations are $\bullet A$ values

$$\boxed{\langle \sigma ; e \rangle \Downarrow \langle \sigma' ; v \rangle}$$

Operational Semantics

$$\frac{}{\langle \sigma; v\rangle \Downarrow \langle \sigma; v\rangle}$$

$$\frac{\langle \sigma; e\rangle \Downarrow \langle \sigma'; (v_1, v_2)\rangle}{\langle \sigma; \pi_i(e)\rangle \Downarrow \langle \sigma'; v_i\rangle} \qquad \frac{\langle \sigma; e\rangle \Downarrow \langle \sigma'; v\rangle}{\langle \sigma; in_i(e)\rangle \Downarrow \langle \sigma'; in_i(v)\rangle}$$

$$\frac{\langle \sigma; e\rangle \Downarrow \langle \sigma'; in_i\, v\rangle \quad \langle \sigma'; [v/x]e_i\rangle \Downarrow \langle \sigma''; v\rangle}{\langle \sigma; case(e, \overrightarrow{in_i\, x \to e_i})\rangle \Downarrow \langle \sigma''; v\rangle}$$

$$\frac{\langle \sigma; e_1\rangle \Downarrow \langle \sigma'; \lambda x.e\rangle \quad \langle \sigma'; e_2\rangle \Downarrow \langle \sigma''; v\rangle \quad \langle \sigma''; [v/x]e\rangle \Downarrow \langle \sigma'''; v'''\rangle}{\langle \sigma; e_1\, e_2\rangle \Downarrow \langle \sigma'''; v'''\rangle}$$

$$\frac{\langle \sigma; e\rangle \Downarrow \langle \sigma'; v\rangle}{\langle \sigma; in(e)\rangle \Downarrow \langle \sigma'; in(v)\rangle} \qquad \frac{\langle \sigma; e\rangle \Downarrow \langle \sigma', in(v)\rangle}{\langle \sigma; out(e)\rangle \Downarrow \langle \sigma'; v\rangle}$$

$$\frac{\langle \sigma; [fix\, x\, e/x]e\rangle \Downarrow \langle \sigma'; v'\rangle}{\langle \sigma; fix\, x.e\rangle \Downarrow \langle \sigma'; v'\rangle}$$

Operational Semantics

$$\frac{\langle \sigma; e \rangle \Downarrow \langle \sigma'; v \rangle}{\langle \sigma; \mathrm{box}(e) \rangle \Downarrow \langle \sigma'; \mathrm{box}(v) \rangle}$$

$$\frac{\langle \sigma; e_1 \rangle \Downarrow \langle \sigma'; \mathrm{box}(v) \rangle \quad \langle \sigma'; [v/x]e_2 \rangle \Downarrow \langle \sigma''; v'' \rangle}{\langle \sigma; \mathrm{let}\ \mathrm{box}(x) = e_1\ \mathrm{in}\ e_2 \rangle \Downarrow \langle \sigma''; v'' \rangle}$$

$$\frac{\cdot}{\langle \sigma; \delta(e) \rangle \Downarrow \langle [\sigma, \ell : e\ \mathrm{later}]; \ell \rangle}$$

$$\frac{\langle \sigma; e_1 \rangle \Downarrow \langle \sigma'; \ell \rangle \quad \langle \sigma'; [!\ell/x]e_2 \rangle \Downarrow \langle \sigma''; v'' \rangle}{\langle \sigma; \mathrm{let}\ \delta(x) = e_1\ \mathrm{in}\ e_2 \rangle \Downarrow \langle \sigma''; v'' \rangle}$$

$$\frac{\ell : v\ \mathrm{now} \in \sigma}{\langle \sigma; !\ell \rangle \Downarrow \langle \sigma; v \rangle}$$

Tick Semantics

$\boxed{\sigma \Rightarrow \sigma'}$

$$\frac{}{\cdot \Rightarrow \cdot}$$

$$\frac{\sigma \Rightarrow \sigma' \quad \langle \sigma'; e \rangle \Downarrow \langle \sigma''; v \rangle}{(\sigma, l : e\ \text{later}) \Rightarrow (\sigma'', l : v\ \text{now}}$$

$$\frac{\sigma \Rightarrow \sigma'}{(\sigma, l : v\ \text{now}) \Rightarrow (\sigma', l : \text{null})}$$

$$\frac{\sigma \Rightarrow \sigma'}{(\sigma, l : \text{null}) \Rightarrow (\sigma', l : \text{null})}$$

Guarantees space leak freedom

Proving type safety needs another logical relation!

Conclusion

1. Modal logic arises from formal logic
2. Same rules can have many applications
3. Even very "purist" type theories can have "applied" utility
4. Many more applications to be discovered

$$\text{button.click} : \Box(\text{Data} \to \text{unit}) \to \text{unit}$$

$$: \Box(\neg\,\text{Data}) \to \text{unit}$$

$$: \neg\,\Box(\neg\,\text{Data})$$

$$: \Diamond\,\text{Data}$$

$$\neg A = A \to p$$