Fitch-style modal types

Amos Robinson, FP-Syd, June 2025

Modal logic - all possible worlds

$\Box(1 \leq \text{DiceRoll} \leq 6)$

In all worlds, our dice roll is between 1 and 6

$\text{DiceRoll} = 6$

In this world, the dice roll results in 6

$\Box(\text{DiceRoll} = 6 \implies \text{Win})$

In all worlds, if the dice roll is 6, then we win

Win

In this world, we won

Applications of modal types

- Template metaprogramming / staged compilation
  - $\Box A$: compile-time code that produces a runtime value of type A
- Streaming
  - $\Box A$: a stream of values of type A
- Purity in an impure language:
  - $\Box A$: a pure computation of type A with no external dependencies

Common axioms for modal types

K: $\Box(A \to B) \to \Box A \to \Box B$

R: $A \to \Box A$

T: $\Box A \to A$

Staging: axiom K

add : exp int → exp int → exp int

add x y = box (unbox x + unbox y)

Staging: axiom K

```
k : exp (a → b) → exp a → exp b

add (x: exp int) (y: exp int) =
  let add__ : exp (int → (int → int)) = box (+) in
  let addx_ : exp (int → int)         = k add__ x in
  let addxy : exp int                 = k addx_ y in
  addxy
```

Staging: no R (return)

$\text{return} : a \rightarrow \text{exp } a$

```
staged_query (): exp (list user) =
  let db = Db.connect "127.0.0.1" in  (* connection opened at compile-time *)
  box (Db.read_table db)              (* …but used at runtime! *)
```

Staging: no T (eval)

```
eval : exp a → a

staged_args (): list string =
  let read_args = box (System.getArgs ()) in  (* read runtime arguments... *)
  eval read_args                               (* at compile time *)
```

Common axioms for modal types - matrix

	Staging	Streaming	Purity
K: $\Box(A \to B) \to \Box A \to \Box B$	✅	✅	✅
R: $A \to \Box A$	❌	✅	❌
T: $\Box A \to A$	❌	❌	✅

Staging - type system

Staging - grammar

$$e ::= f \mid x \mid \lambda x.\, e \mid e\, e \mid \text{box } e \mid \text{unbox } e$$

$$\tau ::= \text{int} \mid \text{real} \mid \cdots \mid \Box\, \tau$$

$$\Gamma ::= \cdot \mid \Gamma, x : \tau \mid \Gamma, \text{Lock}$$

Staging - typing rules

$$\frac{\Gamma, \text{Lock} \ \vdash \ e : \tau}{\Gamma \ \vdash \ \text{box}\ e : \Box\,\tau}\text{(Box)}$$

$$\frac{\Gamma \ \vdash \ e : \Box\,\tau \qquad \text{Lock} \notin \Gamma'}{\Gamma, \text{Lock}, \Gamma' \ \vdash \ \text{unbox}\ e : \tau}\text{(Unbox)}$$

$$\frac{\text{Lock} \notin \Gamma'}{\Gamma, x : \tau, \Gamma' \ \vdash \ x : \tau}\text{(Var)}$$

$$\frac{f : \tau \in \text{Defs}}{\Gamma \ \vdash \ f : \tau}\text{(Const)}$$

$$\frac{\Gamma, x : \tau \ \vdash \ e : \tau'}{\Gamma \ \vdash \ (\lambda x : \tau\,.\, e) : \tau \to \tau'}\text{(Abs)}$$

$$\frac{\Gamma \ \vdash \ e : (\tau' \to \tau) \qquad \Gamma \ \vdash \ e' : \tau'}{\Gamma \ \vdash \ e\ e' : \tau}\text{(App)}$$

Staging - boxed addition

$$\frac{\Gamma, \mathsf{Lock} \ \vdash \ e : \tau}{\Gamma \ \vdash \ \mathsf{box}\ e : \Box \tau}\,(\mathsf{Box}) \qquad \frac{\Gamma \ \vdash \ e : \Box \tau \qquad \mathsf{Lock} \notin \Gamma'}{\Gamma, \mathsf{Lock}, \Gamma' \ \vdash \ \mathsf{unbox}\ e : \tau}\,(\mathsf{Unbox})$$

$$\frac{\mathsf{Lock} \notin \Gamma'}{\Gamma, x : \tau, \Gamma' \ \vdash \ x : \tau}\,(\mathsf{Var})$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : \Box\,\mathsf{int}, y : \Box\,\mathsf{int} \ \vdash \ x : \Box\,\mathsf{int}}\,(\mathsf{Var})}{x : \Box\,\mathsf{int}, y : \Box\,\mathsf{int}, \mathsf{Lock} \ \vdash \ \mathsf{unbox}\ x : \mathsf{int}}\,(\mathsf{Unbox}) \qquad \cdots}{x : \Box\,\mathsf{int}, y : \Box\,\mathsf{int}, \mathsf{Lock} \ \vdash \ (\mathsf{unbox}\ x + \mathsf{unbox}\ y) : \mathsf{int}}\,(\mathsf{App})}{x : \Box\,\mathsf{int}, y : \Box\,\mathsf{int} \ \vdash \ \mathsf{box}\ (\mathsf{unbox}\ x + \mathsf{unbox}\ y) : \Box\,\mathsf{int}}\,(\mathsf{Box})$$

Staging - invalid variable occurrence

$$\frac{\Gamma, \text{Lock} \ \vdash \ e : \tau}{\Gamma \ \vdash \ \text{box } e : \Box\tau}\text{(Box)}$$

$$\frac{\Gamma \ \vdash \ e : \Box\tau \qquad \text{Lock} \notin \Gamma'}{\Gamma, \text{Lock}, \Gamma' \ \vdash \ \text{unbox } e : \tau}\text{(Unbox)}$$

$$\frac{\text{Lock} \notin \Gamma'}{\Gamma, x : \tau, \Gamma' \ \vdash \ x : \tau}\text{(Var)}$$

$$\dfrac{\dfrac{\dfrac{\cdots}{\text{db} : \text{Database}, \text{Lock} \ \vdash \ \text{Db.read\_table} : \cdots}\text{(Const)} \qquad \dfrac{\text{Lock} \notin \text{Lock}}{\text{db} : \text{Database}, \text{Lock} \ \vdash \ \text{db} : \cdots}\text{(Var)}}{\text{db} : \text{Database}, \text{Lock} \ \vdash \ \text{Db.read\_table db} : \cdots}\text{(App)}}{\text{db} : \text{Database} \ \vdash \ \text{box (Db.read\_table db)} : \cdots}\text{(Box)}$$

box / unbox

$$\begin{array}{c} \Gamma, \mathrm{Lock} \vdash A \\ \hline\hline \Gamma \vdash \Box A \end{array}$$

Icicle in Fitch

$k ::= \text{Element} \mid \text{Aggregate}$

$\Gamma ::= \cdot \mid \Gamma, x : \tau \mid \Gamma, \text{Lock}_k$

$e ::= \text{box}_k\ e \mid \text{unbox}_k\ e \mid \cdots$

$$\frac{}{\Gamma, x : \tau, \Gamma' \ \vdash \ x : \tau}\text{(Var)}$$

$$\frac{\Gamma \ \vdash \ e : \Box_k \tau \qquad \text{Lock}_{A,E} \notin \Gamma'}{\Gamma, \text{Lock}_k, \Gamma' \ \vdash \ \text{unbox}_k\ e : \tau}\text{(Unbox)}$$

$$\frac{\Gamma, \text{Lock}_k \ \vdash \ e : \tau}{\Gamma \ \vdash \ \text{box}_k\ e : \Box_k \tau}\text{(Box)}$$

$\text{fold} : \tau \to (\Box_E \tau \to \Box_E \tau) \to \Box_A \tau$

References

Clouston 2018 - Fitch-style modal lambda calculi

https://arxiv.org/abs/1710.08326v2

Valliappan 2023 - Fitch-style applicative functors (extended abstract)

https://nachivpn.me/r.pdf

https://nachivpn.me/k/

Bahr 2019 - Simply RaTT

https://bahr.io/pubs/entries/bahr19icfp.html