Relative Monads in CBPV for Stack-based Effects

Max S. New

University of Michigan

Monads

$T : \mathrm{Type} \to \mathrm{Type}$

$ret : X \Rightarrow TX$

$ext : (X \Rightarrow TY) \Rightarrow (TX \Rightarrow TY)$

(+ equations)

Intuition:

values of TX are first-class values representing effectful computations that return X vals

Monads

$T : \mathrm{Type} \to \mathrm{Type}$

$\mathrm{ret} : X \Rightarrow TX$

$\mathrm{ext} : (X \Rightarrow TY) \Rightarrow (TX \Rightarrow TY)$

(+ equations)

"Exceptions"

$\mathrm{Exn}_E\, A = A + E$

$\mathrm{ret}\; x = \mathrm{inl}$

$\mathrm{ext}\; f\; (\mathrm{inl}\; x) = f\; x$

$\mathrm{ext}\; f\; (\mathrm{inr}\; e) = \mathrm{inr}\; e$

Monads

$T : Type \to Type$

$ret : X \Rightarrow TX$

$ext : (X \Rightarrow TY) \Rightarrow (TX \Rightarrow TY)$

(+ equations)

"State"

$State_S\, A = S \to A \times S$

$ret\ x = \lambda s.\ (x, s)$

$ext\ f\ t = \lambda s_1.$
$\quad let\ (x, s_2) = t\ s_1\ in$
$\quad\quad f\ x\ s_2$

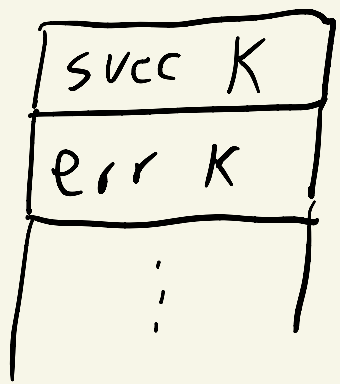
Monads "simulate" effects in high-level languages.

So what about "real" effects implemented in the compiler?

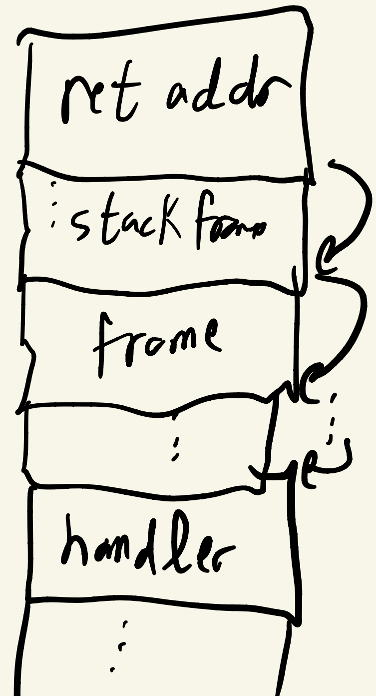
Exceptions in Compilation

A+E

"double barrel"
Kont



stack-walking



State in Compilation

mutable variable x $x := 5$

↳ on STACK   `mov [rsp + off], 5`

↳ or in REGISTER   `mov r15, 5`

Effects as Monads

first-class values representing effectful computation

Effects in Compilation

Continuations,

stack structure,

registers

Generalize

Monad $\longrightarrow$ Adjunction

$T : \mathrm{Type} \to \mathrm{Type}$

first-class computations

$$\begin{tikzcd} \mathrm{ValTy} \arrow[r, bend left, "F"] \arrow[r, phantom, "\perp"] & \mathrm{CompTy} \arrow[l, bend left, "U"] \end{tikzcd}$$

$T := UF$

Monad $\longrightarrow$ Adjunction

Moggi: Computational λ-calculus $\longrightarrow$ Levy: Call-by-push-value

Wadler: Monads for Functional Programming $\longrightarrow$??

Monad $\longrightarrow$ Adjunction

Moggi: Computational λ-calculus $\longrightarrow$ Levy: Call-by-push-value

Wadler: Monads for Functional Programming $\longrightarrow$ <u>relative</u> monads in CBPV

CBPV

Value Types $A ::= \mathrm{Int} \mid \sum_{i\in I} A_i \mid \times_{i\in I} A_i \mid U\underline{B}$

Comp. Types $\underline{B} ::= FA \mid A \to \underline{B} \mid \&_{i\in I} \underline{B}_i$

Levy "A value is,
a computation does"

(first class) Value Types

$A_1 \times A_2$ $\quad$ (V_1, V_2)

$A_1 + A_2$ $\quad$ $\sigma_1 V_1$ or $\sigma_2 V_2$

UB $\quad$ thunk M (closure)

Levy: "A Value is"

Computation Types

$A \to B$ pops an A off stack; does B

$B_1 \& B_2$ pops either π_1 or π_2 off stack; does B_i

FA does effects, return

Levy: "A comp. does"

$$F : \mathrm{ValTy} \rightleftarrows \mathrm{CompTy} : U$$

$$\frac{\Gamma \vdash V : A}{\Gamma \vdash \mathrm{ret}\ V : FA} \qquad \frac{\Gamma \vdash M : FA \quad \Gamma, x : A \vdash N : B}{\Gamma \vdash x \leftarrow M; N : B}$$

$$\frac{\Gamma \vdash M : B}{\Gamma \vdash \mathrm{thunk}\ M : UB} \qquad \frac{\Gamma \vdash V : UB}{\Gamma \vdash \mathrm{force}\ V : B}$$

CBPV decomposes Monad, Eval Order

$$T = UF$$

$$A \to_{cbv} A' := U(A \to FA') \overset{\text{Moggi}}{\cong} A \Rightarrow TA'$$

$$B \to_{cbn} B' := UB \to B' \overset{\text{Girard}}{\cong} !B \multimap B'$$

Computation Types

$A \to B$ pops an A off $\boxed{\text{stack;}}$ does B

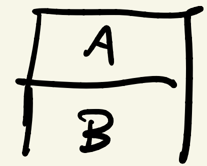
$B_1 \& B_2$ pops either π_1 or π_2 off $\boxed{\text{stack}}$; does B_i

FA does effects, return

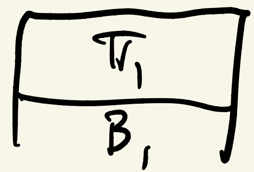
Levy: "A comp. <u>does</u> "

Computation Types are Stack Types

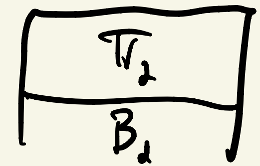
$A \to B$



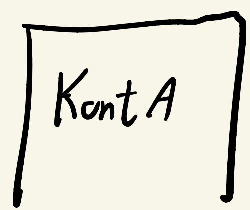
$B_1 \& B_2$



or

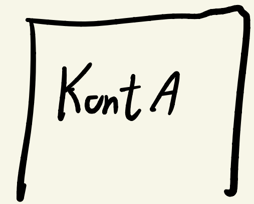


FA

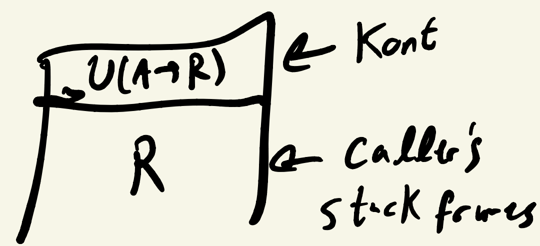


Computation Types are Stack Types

$F A$



$\cong \forall R.\, U(A \to R) \to R$



(Møgelberg-Simpson)

Relative Monad

Altenkirch, Chapman, & Uustalu

$J : \mathcal{C} \to \mathcal{D}$

J-relative monad is

(I) $T : \mathcal{C} \to \mathcal{D}$

(II) $ret : \mathcal{D}(JX, TX)$

(III) $ext : \mathcal{D}(JX, TY) \to \mathcal{D}(TX, TY)$

(+ equations...)

Relative Monad

$F : ValTy \to CompTy$

F-relative monad is

(I) $Eff : ValTy \to CompTy$

(II) $ret : FX \multimap Eff X$

(III) $ext : (FX \multimap Eff Y) \to Eff X \multimap Eff Y$

Relative Monad

$F : \mathrm{ValTy} \to \mathrm{CompTy}$

F-relative monad is

(I) $\mathrm{Eff} : \mathrm{ValTy} \to \mathrm{CompTy}$

(II) $\mathrm{ret} : FX \multimap \mathrm{Eff}\,X \qquad \cong \; X \to \mathrm{Eff}\,X$

(III) $\mathrm{ext} : (FX \multimap \mathrm{Eff}\,Y) \to \mathrm{Eff}\,X \multimap \mathrm{Eff}\,Y \cong U(X \to \mathrm{Eff}\,Y) \to \mathrm{Eff}\,X \multimap \mathrm{Eff}\,Y$

Relative Monad

$F : \mathrm{ValTy} \to \mathrm{CompTy}$

F-relative monad is

(I) $\mathrm{Eff} : \mathrm{ValTy} \to \mathrm{CompTy}$

$$\boxed{\mathrm{Eff}\ A}$$

(II) $\mathrm{ret} : F X \multimap \mathrm{Eff}\ X$

$$\boxed{\mathrm{Eff}\ X} \multimap \boxed{\mathrm{Kont}\ X}$$

(III) $\mathrm{ext} : U(X \to \mathrm{Eff}\ Y) \to \mathrm{Eff}\ X \multimap \mathrm{Eff}\ Y$

$$\boxed{\begin{array}{c} U(X \to \mathrm{Eff}\ Y) \\ \hline \mathrm{Eff}\ Y \end{array}} \multimap \boxed{\mathrm{Eff}\ X}$$

Exceptions

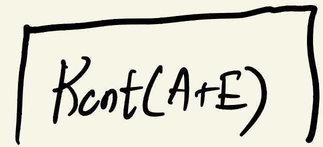
$$\mathrm{Exn}_E\, A = F(A+E) \qquad \boxed{\mathrm{Kcont}(A+E)}$$

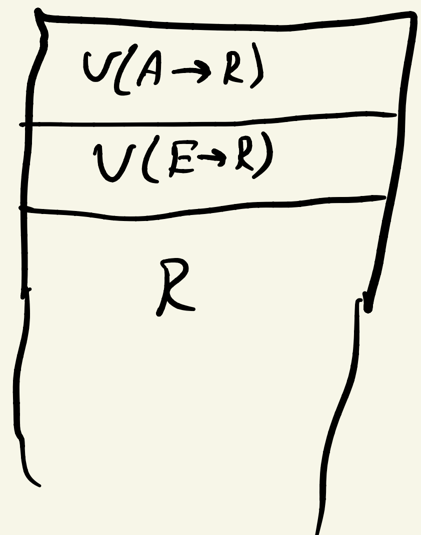
Exceptions

$$\mathrm{Exn}_E\, A = F(A+E)$$

$$\cong \forall R.\, U(A+E \to R) \to R$$

$$\cong \forall R.\, U(A \to R) \to U(E \to R) \to R$$



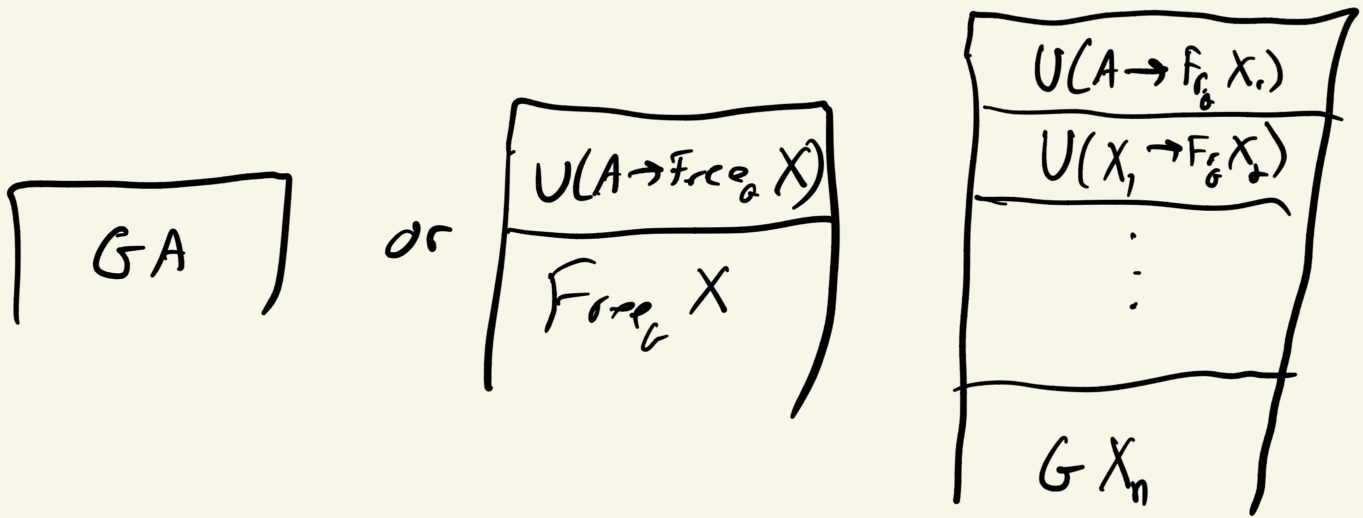


Stack Walking: Free Monad

$$\mathrm{Free}_G A := GA \,\&\, (\forall X.\ U(A \to \mathrm{Free}_G X) \to \mathrm{Free}_G X)$$

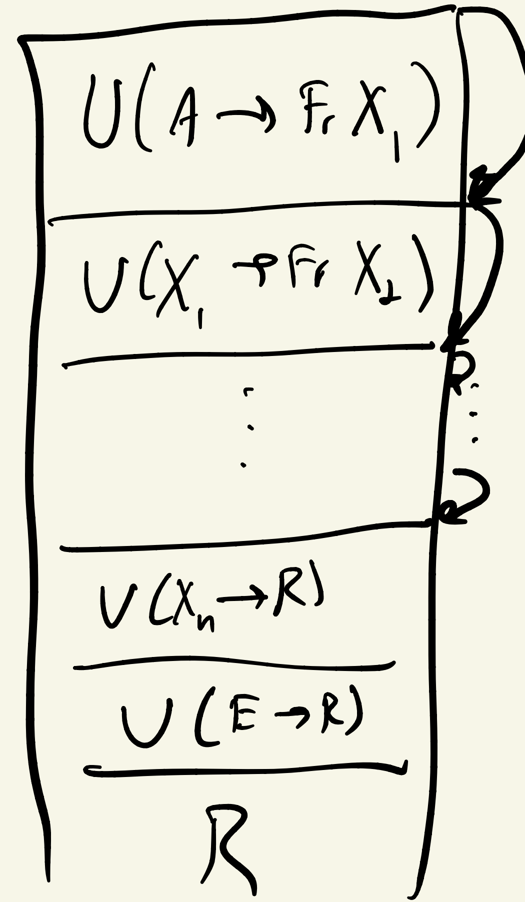
Stack Walking: Free Monad

$$\mathrm{Free}_G\, A := GA \,\&\, (\forall X.\ U(A \to \mathrm{Free}_G X) \to \mathrm{Free}_G X)$$



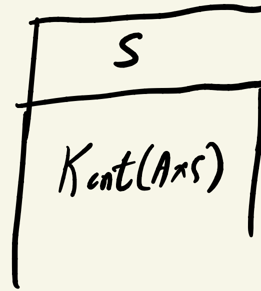
$\mathrm{Free}_{\mathrm{Exn}_E}\ A$

$$\mathrm{raise}\ e := \{\ \pi_1 \mapsto \lambda k_s\, k_e.\ f_c\ k_e\ e$$
$$\mid\ \pi_2 \mapsto \lambda k_s.\ \mathrm{raise}\ e\ \}$$

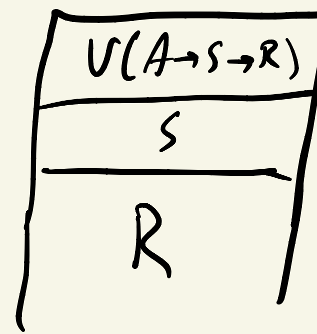


Example: State

$$\mathrm{State}_S\,A := S \to F(A \times S)$$



$$\cong \forall R.\, U(A \to S \to R) \to S \to R$$



Relation to F, U

(I) F is the "identity" rel. monad

(II) If $T : V_{ty} \to V_{ty}$, then FT is rel monad
monad

(III) If Eff rel monad $U\mathrm{Eff}$ is monad

Composing Effects

Any Rel monad <u>definable</u> in CBPV determines a relative monad transformer

$$F(A+E) \quad \mapsto \quad Eff\,(A+E)$$

$$\forall R.\ U(A \to R) \to U(E \to R) \to R \quad \mapsto \quad \forall R.\ Eff\text{-}Alg\ R \to U(A \to R) \to U(E \to R) \to R$$

Limitations of CBPV

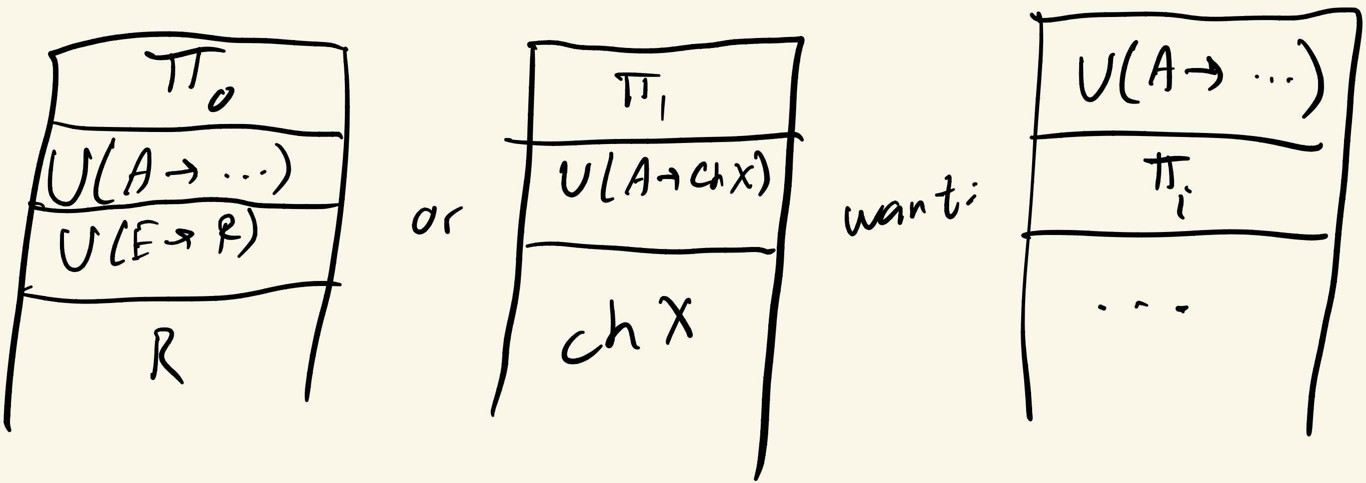
- Stack machine, no registers

- CBPV+res:

$$\mathrm{State}^{r}_{s}\, A := \forall R \perp r.\ U(A \to S \to_r R) \to S \to_r R$$

Goal: ret, bind for exn have no branching

$$\mathrm{CheapExn}\ A := (\forall R.\ U(A \to U(E \to R) \to R) \to U(E \to R) \to R)$$
$$\&\ (\forall X.\ U(A \to \mathrm{Cheap}\ X) \to \mathrm{Cheap}\ X)$$



Relative Comonads?

$F \sim \mathrm{Kont} \mapsto \mathrm{Eff} \sim \mathrm{Kont}{+}{+}$

$U \sim \mathrm{Closure} \mapsto \mathrm{Clo} \sim \mathrm{Closure}{+}{+}$?

$\mathrm{Destructor}\ B = UB \times UF1$

$\mathrm{Debuggable}\ B = UB \times \mathrm{Info}$

Applications?

- CBPV as low-level lang

- CBPV as shared IR
  - FFIs using relative monad morphisms?

I. CBPV is a metalang for stack machines

II. Use CBPV to model effects as implemented in compilers using Relative monads

III. CBPV (and extensions): combine low-level details + hi-level abstractions

impl in progress . . .