

Monads and Effects

Karl Krukow

4th of Nov. 2002
CTfCS - 2002

Plan

- ❖ Syntax of a simple functional PL
- ❖ A Denotational Semantics
- ❖ Adding some effects
 - Exceptions
 - State
- ❖ Some CT
- ❖ Generic Monadic (CBV) Semantics.

Syntax

◆ Types are

– $\tau ::= \text{nat} \mid \tau_1 \rightarrow \tau_2$.

– Natural numbers are in unary-notation

◆ Terms are

x		x		$E ::=$
$\text{fun } f(x^{\tau_1})^{\tau_2}.E$		$\text{fun } f(x^{\tau_1})^{\tau_2}.E$		
$E_1 E_2$		$E_1 E_2$		
0		0		
$\text{succ } E$		$\text{succ } E$		
$\text{ncase}(E_1, E_2, x.E_3)$		$\text{ncase}(E_1, E_2, x.E_3)$		

Intended meaning

- ◆ $\text{ncase}(E_1, E_2, x.E_3)$ corresponds to ML's
case E_1 of | Zero $\Rightarrow E_2$
| Succ $x \Rightarrow E_3$
- ◆ $\text{fun } f(x^{\tau_1})^{\tau_2}.E$ is a recursive λ -abstraction – in ML
let fun f(x : τ_1) : $\tau_2 = E$ in f end
- ◆ the others are as usual

A CVB Denotational Semantics

- ◆ Types, τ are interpreted as a CPO, $\mathcal{V} \llbracket \tau \rrbracket$
 - $\mathcal{V} \llbracket \text{nat} \rrbracket \stackrel{(\text{def})}{=} \mathbb{N}$ - the "flat" natural numbers
 - $\mathcal{V} \llbracket \tau_1 \rightarrow \tau_2 \rrbracket \stackrel{(\text{def})}{=} \mathcal{V} \llbracket \tau_1 \rrbracket \Rightarrow \mathcal{V} \llbracket \tau_2 \rrbracket^\perp$ - the space of continuous functions from $\mathcal{V} \llbracket \tau_1 \rrbracket$ to the lifting $(-\perp)$ of $\mathcal{V} \llbracket \tau_2 \rrbracket$ (which is itself a CPO)
 - finally typing contexts, $\Gamma = \{x_1 \mapsto \tau_1, \dots, x_n \mapsto \tau_n\}$ are denoted by
- $$\mathcal{V} \llbracket \Gamma \rrbracket \stackrel{(\text{def})}{=} \prod_{x \in \text{dom } \Gamma} \mathcal{V} \llbracket \Gamma(x) \rrbracket$$
- We can think of an element, $d \in \prod \mathcal{V} \llbracket \Gamma(x) \rrbracket$, as a function $d : \text{dom } \Gamma \rightarrow \bigcup_{x \in \text{dom } \Gamma} \mathcal{V} \llbracket \Gamma(x) \rrbracket$ such that $d(x) \in \mathcal{V} \llbracket \Gamma(x) \rrbracket$

Denotational Semantics – terms

- Well-typed terms, $\Gamma \vdash E : \tau$ are interpreted as continuous functions, ie. elements $\in \mathcal{V}[\Gamma] \Rightarrow \mathcal{V}[\tau]^\perp$

$$\begin{aligned}
 \mathcal{V}[x] & \stackrel{(\text{def})}{=} \lambda \rho. \mathbf{dn}(\rho x) \\
 \mathcal{V}[0] & \stackrel{(\text{def})}{=} \lambda \rho. \mathbf{dn}(0) \\
 \mathcal{V}[\text{succ } E] & \stackrel{(\text{def})}{=} \lambda \rho. (\lambda n. \mathbf{dn}(n + 1))^\dagger(\mathcal{V}[E]\rho) \\
 \mathcal{V}[\text{ncase}(E_1, E_2, x.E_3)] & \stackrel{(\text{def})}{=} \lambda \rho. \Phi^\dagger(\mathcal{V}[E_1]\rho)
 \end{aligned}$$

- where the continuous function Φ is

$$\Phi \stackrel{(\text{def})}{=} \lambda n. \begin{cases} \mathcal{V}[E_2]\rho & \text{if } n = 0 \\ \mathcal{V}[E_3]\rho[x \mapsto m] & \text{if } n = m + 1 \end{cases}$$

Terms continued . . .

◆ and

$$\mathcal{V} \llbracket \text{fun } f(x) \text{ in } \text{fix } \Psi \rrbracket \stackrel{(\text{def})}{=} \lambda p. \text{np}(\text{fix } \Psi)$$

◆ where $\Psi : \mathcal{V} \llbracket \tau_1 \rrbracket \rightarrow \mathcal{V} \llbracket \tau_2 \rrbracket \rightarrow \mathcal{V} \llbracket \tau_1 \rrbracket \rightarrow \mathcal{V} \llbracket \tau_2 \rrbracket \rightarrow \dots$

$$\Psi(\phi) \stackrel{(\text{def})}{=} \lambda a. \mathcal{V} \llbracket E \rrbracket (p[f \mapsto \phi, x \mapsto a])$$

◆ finally application is

$$\mathcal{V} \llbracket E_1 E_2 \rrbracket \stackrel{(\text{def})}{=} \lambda p. (\lambda \phi. \phi \dagger (\mathcal{V} \llbracket E_2 \rrbracket p)) \dagger (\mathcal{V} \llbracket E_1 \rrbracket p)$$

Extensions

- ◆ What if we want to add exceptions to our language?
 - $E ::= \dots \mid \text{fail} \mid \text{try } E \text{ catch } E$

◆ Then we should change the interpretation of types

- $\mathcal{V}[\text{nat}] = \mathbb{N}$ (– the same)
- $\mathcal{V}[\tau_1 \rightarrow \tau_2] = \mathcal{V}[\tau_1] \Rightarrow (\mathcal{V}[\tau_2] + \mathbf{1}_{CPO})^\perp$ (– changed)

- ◆ So functions might produce a result, raise an exception or simply diverge

- ◆ Define for any continuous function $f : A \rightarrow (B + \mathbf{1}_{CPO})^\perp$ an “extended” function $f_\diamond : (A + \mathbf{1}_{CPO})^\perp \rightarrow (B + \mathbf{1}_{CPO})^\perp$ by

$$\left. \begin{array}{l} \perp \mapsto \perp \\ \text{np(inl}(a)) \mapsto fa \\ \text{np(inr}(\star)) \mapsto \perp \end{array} \right\} \stackrel{\text{def}}{=} f_\diamond$$

- ◆ also write $\xi(x)$ for $\text{np(inl}(x))$

Redefining the semantics

- ❖ To give semantics to our extended language, we need to redefine the entire semantics, not just add the interpretation of the new constructs!
- ❖ It would be much nicer if we could have a more modular approach so that we don't need to change the meaning of the old constructs, when new are introduced.
- ❖ Later we will see that *Monads* can help with this.

Exception Semantics

◆ The exception-semantics, \mathcal{V}^e of terms $\Gamma \vdash E : \tau$, is a continuous function: $\mathcal{V}^e[\Gamma] \mapsto (\mathcal{V}^e[\tau] + \mathbf{1}_{CPO})^\perp$

$$\mathcal{V}^e[x] \stackrel{(\text{def})}{=} \lambda \rho. \xi(\rho x)$$

$$\mathcal{V}^e[\mathbf{0}] \stackrel{(\text{def})}{=} \lambda \rho. \xi(0)$$

$$\mathcal{V}^e[\text{succ } E] \stackrel{(\text{def})}{=} \lambda \rho. (\lambda n. \xi(n + 1)) \diamond (\mathcal{V}^e[E])[\rho]$$

$$\mathcal{V}^e[\text{ncase}(E_1, E_2, x.E_3)] \stackrel{(\text{def})}{=} \lambda \rho. \Phi \diamond (\mathcal{V}^e[E_1])[\rho]$$

⋮

◆ Φ is as before (but with $\mathcal{V} = \mathcal{V}^e$).

Exception Semantics - continued.

◆ for fail

$$\mathcal{V}^e \llbracket \text{fail} \rrbracket \stackrel{(\text{def})}{=} \lambda p. \text{np}(\text{inr}(\star))$$

◆ for the try-catch

$$\mathcal{V}^e \llbracket \text{try } E_1 \text{ catch } E_2 \rrbracket \stackrel{(\text{def})}{=} \lambda p. \phi^\dagger(\mathcal{V}^e \llbracket E_1 \rrbracket p)$$

◆ where

$$\phi \stackrel{(\text{def})}{=} \begin{cases} \text{inl}(a) \mapsto \xi(a) \\ \text{inr}(\star) \mapsto \mathcal{V}^e \llbracket E_2 \rrbracket p \end{cases}$$

What about state

- ◆ Let's try and add state to the simple language without exceptions
- ◆ Let $\Delta = \{v_1, v_2, \dots, v_n\}$ be a fixed collection of *cell names* - v will now range $v \in \{v_i\}_{i=1}^n$
- ◆ We extend the syntax with

$$E ::= \dots \mid !v \mid v := E$$

- ◆ The interpretation of states will be (set-theoretic) functions from the *cell-names* to natural numbers.

$$\mathcal{V}_s \llbracket \Delta \rrbracket_{(def)} = \Delta \Leftarrow \mathbb{N}$$

State semantics – types

◆ We'll keep the interpretation of `nat` and `take`

– functions from τ_1 to τ_2 as

$$\mathcal{V}_s[\tau_1 \rightarrow \tau_2] \stackrel{\text{(def)}}{=} \mathcal{V}_s[\tau_1] \Rightarrow [\mathcal{V}_s[\Delta]] \Rightarrow (\mathcal{V}_s[\tau_2] \times \mathcal{V}_s[\Delta])^\perp$$

– that is the space of continuous functions taking as input a “ τ_1 ”

and producing a continuous function which takes a state and

returns a result and an updated state (or diverges)

◆ the meaning of a well-typed term, $\Gamma \vdash E : \tau$ will be a continuous

function

$$\mathcal{V}_s[E] : \mathcal{V}_s[\Gamma] \rightarrow [\mathcal{V}_s[\Delta]] \rightarrow (\mathcal{V}_s[\tau] \times \mathcal{V}_s[\Delta])^\perp$$

State semantics – terms

◆ We now define the semantics of terms

$$\begin{aligned}
 \mathcal{V}_s \llbracket x \rrbracket & \stackrel{(\text{def})}{=} \lambda p. \lambda \sigma. \mathbf{np}(px, \sigma) \\
 \mathcal{V}_s \llbracket \mathbf{0} \rrbracket & \stackrel{(\text{def})}{=} \lambda p. \lambda \sigma. \mathbf{np}(0, \sigma) \\
 \mathcal{V}_s \llbracket \text{succ } E \rrbracket & \stackrel{(\text{def})}{=} \lambda p. \lambda \sigma. (\lambda(n, \sigma'). \mathbf{np}(n + 1, \sigma')) \dagger (\mathcal{V}_s \llbracket E \rrbracket) \dagger (p\sigma) \\
 \mathcal{V}_s \llbracket \text{ncase}(E_1, E_2, x.E_3) \rrbracket & \stackrel{(\text{def})}{=} \lambda p. \lambda \sigma. \Phi \dagger (\mathcal{V}_s \llbracket E_1 \rrbracket) \dagger (p\sigma) \\
 & \vdots
 \end{aligned}$$

◆ where the function $\Phi : \mathcal{V}_s \llbracket \text{nat} \rrbracket \times \mathcal{V}_s \llbracket \Delta \rrbracket \rightarrow (\mathcal{V}_s \llbracket \tau \rrbracket \times \mathcal{V}_s \llbracket \Delta \rrbracket)^\perp$ is

$$\Phi(n, \sigma') \stackrel{(\text{def})}{=} \begin{cases} \mathcal{V}_s \llbracket E_2 \rrbracket p\sigma' & \text{if } n = 0 \\ \mathcal{V}_s \llbracket E_3 \rrbracket (p[x \mapsto m]) \sigma' & \text{if } n = m + 1 \end{cases}$$

State semantics - continued

◆ for $i v$

$$\mathcal{V}_s \llbracket i v \rrbracket \stackrel{\text{(def)}}{=} \lambda p. \lambda \sigma. \mathbf{np}(\sigma v, \sigma)$$

◆ for the $v := E$

$$\mathcal{V}_e \llbracket v := E \rrbracket \stackrel{\text{(def)}}{=} \lambda p. \lambda \sigma. (\lambda(n, \sigma'). \mathbf{np}(n, \sigma' [v \mapsto n])) \dagger (\mathcal{V}_s \llbracket E \rrbracket p \sigma)$$

- ◆ This was a more substantial change and hopefully it is a nice surprise that both of these notions of computation (and many more) are instance of a more general CBV - **Monadic** semantics :-)

Monads and Kleisli Triples

- ◆ **Definition:** A *Kleisli Triple*, K , over a category \mathcal{C} , is a triple, $K = (T, \eta, -^*)$ where
 - $T : \text{Obj}(\mathcal{C}) \rightarrow \text{Obj}(\mathcal{C})$ assigns to each object, A of $\text{Obj}(\mathcal{C})$ an object TA of $\text{Obj}(\mathcal{C})$.
 - η is a family of morphisms, indexed by the objects of \mathcal{C} , so that $\eta_A : A \rightarrow TA$.
 - Finally $-^*$ assigns to each morphism $f : A \rightarrow TB$ a "lifted" morphism, $f^* : TA \rightarrow TB$.
- Such that the following equations hold
 - for any object A of \mathcal{C} ,

$$(\eta_A)^* = 1_{TA}$$

- for any morphism $f : A \rightarrow TB$ of \mathcal{C} ,

$$f = (\eta_A; f^*)$$

- for any morphisms $f : A \rightarrow TB, g : B \rightarrow TC$ of \mathcal{C} ,

$$f^*; g^* = (f; g)^*$$

Examples of useful Kleisli Triples

In the category $CP\mathcal{O}$

◆ **partiality**

$$TA = A_{\perp}$$

$$\eta_A = \lambda a. \mathbf{up}(a)$$

$$f_* = \left\{ \begin{array}{l} \perp \mapsto \perp \\ \mathbf{up} a \mapsto f(a) \end{array} \right.$$

◆ **exceptions**

$$TA = (A + \mathbf{1}_{CPO})$$

$$\eta_A = \lambda a. \mathbf{inl}(a)$$

$$f_* = \left\{ \begin{array}{l} \mathbf{inl}(a) \mapsto f(a) \\ \mathbf{inr}(\ast) \mapsto \mathbf{inr}(\ast) \end{array} \right.$$

◆ **state**

$$TA = \Delta \Leftrightarrow (A \times \Delta)$$

$$\eta_A = \lambda a. \lambda \sigma. (a, \sigma)$$

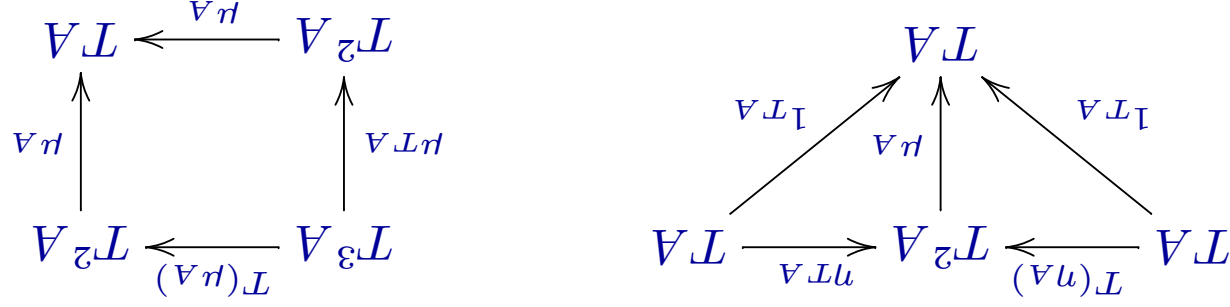
$$f_* c = \lambda \sigma. (\lambda(a, \sigma'). f a \sigma') (c \sigma)$$

What about those Monads?

◆ **Definition:** A *Monad*, \mathbb{T} , over a category \mathcal{C} , is a triple, $\mathbb{T} = (T, \eta, \mu)$ where

- $T : \mathcal{C} \rightarrow \mathcal{C}$ is an endo-functor
- $\eta : 1_{\mathcal{C}} \rightarrow T : \mathcal{C} \rightarrow \mathcal{C}$ is a natural transformation
- $\mu : T^2 \rightarrow T : \mathcal{C} \rightarrow \mathcal{C}$ is a natural transformation

Such that the following diagrams commute



Monadic Semantics

- ◆ In the following let \mathcal{CPO} be the category of CPOs and continuous functions. Let $K = (T, \eta, -^*)$ be a Kleisli Triple over \mathcal{CPO} (we assume further that TA is pointed for every A). We can define a generic CBV, semantics, $\mathcal{V}^K \llbracket - \rrbracket$

◆ on types

– for base type

$$\mathcal{V}^K \llbracket \text{nat} \rrbracket \stackrel{(\text{def})}{=} \mathbb{N}$$

– for function types

$$\mathcal{V}^K \llbracket T_1 \rightarrow T_2 \rrbracket \stackrel{(\text{def})}{=} \mathcal{V}^K \llbracket T_1 \rrbracket \Rightarrow T(\mathcal{V}^K \llbracket T_2 \rrbracket)$$

Monadic Semantics – terms

◆ We assume further that for every $f : A \rightarrow TB$ that $f^* : TA \rightarrow TB$ is

strict.

◆ Now define our K -semantics on terms, $\Gamma \vdash E : \tau$ as a continuous function $\mathcal{V}^K[[E]] : \mathcal{V}^K[[\Gamma]] \rightarrow T(\mathcal{V}^K[[\tau]])$. Write η_τ as a shorthand for $\eta_{\mathcal{V}^K[[\tau]]}$

$$\stackrel{(\text{def})}{=} \mathcal{V}^K[[x]]$$

$$\lambda p.\eta_\tau(px)$$

$$\stackrel{(\text{def})}{=} \mathcal{V}^K[[0]]$$

$$\lambda p.\eta_{\text{nat}}0$$

$$\stackrel{(\text{def})}{=} \mathcal{V}^K[[\text{succ } E]]$$

$$\lambda p.(\lambda n.\eta_{\text{nat}}(n + 1))^*(\mathcal{V}^K[[E]]p)$$

$$\stackrel{(\text{def})}{=} \mathcal{V}^K[[\text{ncase}(E_1, E_2, x.E_3)]]$$

$$\lambda p.\Phi^*(\mathcal{V}^K[[E_1]]p)$$

$$\stackrel{(\text{def})}{=} \mathcal{V}^K[[\text{fun } f(x)_{\tau_1}.\tau_2.E]]$$

$$\lambda p.\eta_{\tau_1 \rightarrow \tau_2}(\text{fix } (\lambda \psi.\lambda a.\mathcal{V}^K[[E]](p[f \mapsto \psi, x \mapsto a]])))$$

$$\stackrel{(\text{def})}{=} \mathcal{V}^K[[E_1 E_2]]$$

$$(\lambda \phi.\phi^*(\mathcal{V}^K[[E_2]]p))^*(\mathcal{V}^K[[E_1]]p)$$

◆ Note that we obtain the **partiality** semantics with $\eta = \text{np}$ and $f^* = f^\dagger$, and the **exception** semantics with $\eta = \xi$ and $f^* = f_\diamond$. Extra exercise: how do we get the **state** semantics?

Monadic Semantics: Exceptions

◆ Suppose now we want to add exceptions to our language.

◆ To do this we simply use the following Kleisli triple, $K_e = (T, \eta, -^*)$

– for T take

$$TA = (A + \mathbf{1}_{CPO})_{\perp}$$

– for η take

$$\eta^A = a \mapsto a = \eta^A(a)$$

– for $-^*$ take

$$\left. \begin{array}{l} \top \mapsto \top \\ \text{nd}(\text{inl}(a)) \mapsto a \\ \text{nd}(\text{inr}(\star)) \mapsto \star \end{array} \right\} f_{\diamond} = f_{\star}$$

Exception Semantics - revisited.

◆ for the old terms simply use \mathcal{V}^K – !! ii

◆ for fail

$$\mathcal{V}^{K^e} \llbracket \text{fail} \rrbracket \stackrel{(\text{def})}{=} \lambda p. \text{np}(\text{inr}(\star))$$

◆ for the try-catch

$$\mathcal{V}^{K^e} \llbracket \text{try } E_1 \text{ catch } E_2 \rrbracket \stackrel{(\text{def})}{=} \lambda p. \phi^\dagger(\mathcal{V}^e \llbracket E_1 \rrbracket p)$$

◆ where

$$\phi \stackrel{(\text{def})}{=} \left\{ \begin{array}{l} \text{inl}(a) \mapsto \xi(a) = \text{np}(\text{inl}(a)) \\ \text{inr}(\star) \mapsto \mathcal{V}^{K^e} \llbracket E_2 \rrbracket p \end{array} \right.$$

◆ Similarly for the state-semantics – just use the Kleisli triple for state.

Want to know more?

- ◆ You should consider this a (very) short introduction to Monads
- ◆ There is a lot more to learn. In particular take a look at the articles:
 - Andrzej Filinski - Semantics of Functional Programming - preliminary lecture notes fall 98 (available from Mikkel)
 - Nick Benton, John Hughes and Eugenio Moggi - Monads and Effects - a comprehensive article (+ a big list of pointers)