#### Update monads

<u>Danel Ahman</u>, U. of Edinburgh Tarmo Uustalu, Inst. of Cybernetics, Tallinn

EWSCS, 5 March 2014

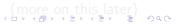
Pure functional programs

$$\frac{f:X\longrightarrow Y}{\lambda x.\,g\,(f\,x):X\longrightarrow Z}$$

$$f: X \longrightarrow 1 + Y$$
  $g: Y \longrightarrow 1 + Z$ 

$$\lambda x$$
. case  $(f x)$  of  $\{\operatorname{inl}(_{-}) \implies \operatorname{inl}() \mid \operatorname{inr}(y) \implies g y\} : X \longrightarrow 1 + Z$ 

- Historically monads have provided such general composition
  - $T : \mathbf{Set} \to \mathbf{Set}$
  - $\eta: \forall \{X\}. X \to TX$
  - $\bullet \ (-)^* : \forall \{X,Y\}. (X \longrightarrow TY) \longrightarrow (TX \longrightarrow TY)$
- Nowadays we often use algebraic presentations



Pure functional programs

$$\frac{f:X\longrightarrow Y \quad g:Y\longrightarrow Z}{\lambda x.\,g\,(f\,x):X\longrightarrow Z}$$

$$f: X \longrightarrow 1 + Y \qquad g: Y \longrightarrow 1 + Z$$

$$\lambda x$$
. case  $(f x)$  of  $\{ \operatorname{inl}(_{-}) \implies \operatorname{inl}() \mid \operatorname{inr}(y) \implies g y \} : X \longrightarrow 1 + Z$ 

- Historically monads have provided such general composition
  - $T : \mathbf{Set} \to \mathbf{Set}$
  - $\eta: \forall \{X\}. X \to TX$
  - $\bullet \ (-)^* : \forall \{X,Y\}. (X \longrightarrow TY) \longrightarrow (TX \longrightarrow TY)$
- Nowadays we often use algebraic presentations

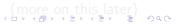


Pure functional programs

$$\frac{f:X\longrightarrow Y}{\lambda x.\,g\,(f\,x):X\longrightarrow Z}$$

$$\frac{f: X \longrightarrow (S \to S \times Y) \qquad g: Y \longrightarrow (S \to S \times Z)}{\lambda x. \ \lambda s. \ g\left(\operatorname{snd}\left(f \times s\right)\right)\left(\operatorname{fst}\left(f \times s\right)\right): X \longrightarrow (S \to S \times Z)}$$

- Historically monads have provided such general composition
  - T : Set → Set
  - $\eta: \forall \{X\}. X \to TX$
  - $\bullet \ (-)^* : \forall \{X,Y\}.(X \longrightarrow TY) \longrightarrow (TX \longrightarrow TY)$
- Nowadays we often use algebraic presentations



Pure functional programs

$$\frac{f: X \longrightarrow Y \quad g: Y \longrightarrow Z}{\lambda x. g(f x): X \longrightarrow Z}$$

$$\frac{f: X \longrightarrow (S \to S \times Y) \qquad g: Y \longrightarrow (S \to S \times Z)}{\lambda x. \ \lambda s. \ g\left(\operatorname{snd}\left(f \times s\right)\right)\left(\operatorname{fst}\left(f \times s\right)\right): X \longrightarrow (S \to S \times Z)}$$

- Historically monads have provided such an interface:
  - $T : \mathbf{Set} \to \mathbf{Set}$
  - $\eta: \forall \{X\}. X \rightarrow TX$
  - $\bullet \ (-)^* : \forall \{X,Y\}. (X \longrightarrow TY) \longrightarrow (TX \longrightarrow TY)$
- Nowadays we often use algebraic presentations



#### Background: Three famous monads

Reader monad

State monad 
$$S$$
 – a set  $T_s X = S \rightarrow S \times X$ 

Ikp:  $(S \rightarrow A) \rightarrow A$ 

upd:  $S \times A \rightarrow A$ 

$$S - a \text{ set} \qquad (P, o, \oplus) - a \text{ monoid}$$

$$T_r X = S \to X \qquad T_w X = P \times X$$

$$|kp: (S \to A) \to A| \qquad upd: P \times A \to A$$

$$S - \text{ states} \qquad A - \text{ carrier of alg. for } T_{\{s,r,w\}}$$

$$P - \text{ updates (alt. "programs")} \qquad + \text{ some equations}$$

Writer monad

### This talk: don't just overwrite. update!

Reader monad 
$$S$$
 – a set  $T_r X = S \rightarrow X$ 

Ikp:  $(S \rightarrow A) \rightarrow A$ 

Writer monad
$$(P, o, \oplus) - \text{a monoid}$$

$$T_w X = P \times X$$

$$\text{upd} : P \times A \to A$$

$$S$$
 – states  $\mathcal{A}$  – carrier of alg. for  $T_{\{s,r,w\}}$  + some equations

#### Monoids, monoid actions

• A monoid on a set P is given by

o: 
$$P$$
,  $\oplus$ :  $P \to P \to P$ ,

$$egin{aligned} egin{aligned} eta \oplus \circ &= eta \ \circ \oplus eta &= eta \end{aligned} \ egin{aligned} (eta \oplus eta') \oplus eta'' &= eta \oplus egin{aligned} (eta' \oplus eta'') \end{aligned}$$

• An action of a monoid  $(P, o, \oplus)$  on a set S is given by

$$\downarrow: S \rightarrow P \rightarrow S$$

$$s \downarrow o = s$$
  
 $s \downarrow (p \oplus p') = (s \downarrow p) \downarrow p'$ 



#### Update monads

A set S, monoid  $(P, o, \oplus)$  and action  $\downarrow$  give an update monad:

$$TX = S \to (P \times X)$$

$$\eta : \forall \{X\}. X \to (S \to P \times X)$$

$$\eta x = \lambda s. (o, x)$$

$$(-)^* : \forall \{X, Y\}. (X \to (S \to P \times Y))$$

$$\to (S \to P \times X) \to (S \to P \times Y)$$

$$f^* g = \lambda s. \text{ let } (p, x) = g s;$$

$$(p', y) = f x (s \downarrow p)$$

$$\text{in } (p \oplus p', y)$$

#### Reader and writer monads as instances

Recall update monads:

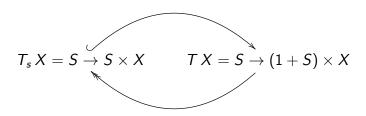
$$TX = S \rightarrow P \times X$$

- Reader monads:  $T_r X = S \to X$  update monads with  $(P, o, \oplus)$  and  $\downarrow$  trivial
- Writer monads:  $T_w X = P \times X$  update monads with S and  $\downarrow$  trivial

#### State monads as canonically related

• State monads:

$$T_s X = S \rightarrow (S \times X)$$
  
embed into and project from update monads



for P the free monoid on the overwrite semigroup  $(S, \bullet)$ 

defined by  $s \bullet s' = s'$ 

### Update monad example: logging state

(set of states)

• 
$$P = S^*$$

(log of states)

• 
$$ss \oplus ss' = ss ++ ss'$$

• 
$$s \downarrow ss = last(s : ss)$$

• 
$$TX = S \rightarrow (S^* \times X)$$

# Update monad example: writing into a buffer

- $S = E^* \times Nat$  (current buffer content and free space)
- $P = E^*$  (new values to write)
- o = []
- $\bullet \ p \oplus p' = p + p'$
- $(s, n) \downarrow p = (s ++ (p|n), n length(p|n))$

(p|n defined as p truncated to length n)

•  $TX = (E^* \times Nat) \rightarrow (E^* \times X)$ 



# Algebras of update monads (cf. algebraic effects)

An algebra of an update monad is a set A with an operation

$$\mathsf{act}: (S \to P \times \mathcal{A}) \to \mathcal{A}$$

$$a = \mathsf{act} \, (\lambda s. \, (\mathsf{o}, \mathsf{a}))$$

$$\mathsf{act} \, (\lambda s. \, (p, \mathsf{act} \, (\lambda s'. \, (p', \mathsf{a}))))$$

$$= \mathsf{act} \, (\lambda s. \, (p \oplus p'[s \downarrow p/s'], a[s \downarrow p/s']))$$

or, equivalently a pair of operations

$$\begin{aligned} \mathsf{lkp} : (S \to \mathcal{A}) &\to \mathcal{A} \\ \mathsf{upd} : P \times \mathcal{A} &\to \mathcal{A} \end{aligned}$$

$$a = \mathsf{lkp} \, (\lambda s. \, \mathsf{upd} (\mathsf{o}, \mathsf{a}))$$

$$\mathsf{upd} \, (p, \mathsf{upd} \, (p', \mathsf{a})) = \mathsf{upd} \, (p \oplus p', \mathsf{a})$$

$$\mathsf{lkp} \, (\lambda s. \, \mathsf{upd} \, (p, \mathsf{lkp} \, (\lambda s'. \, \mathsf{a}))) = \mathsf{lkp} \, (\lambda s. \, \mathsf{upd} \, (p, \mathsf{a}[\mathsf{s} \downarrow p/s']))$$

## Update monads as compatible compositions

The update monad for S,  $(P, o, \oplus)$ ,  $\downarrow$  is the compatible composition of the

reader monads and writer monads 
$$T_r X = S \to X \qquad \qquad T_w X = P \times X$$

for the distributive law

$$\theta: \forall \{X\}. \ P \times (S \to X) \to (S \to P \times X)$$
$$\theta(p, f) = \lambda s. (p, f(s \downarrow p))$$

**Thm.** There is a bijection between update monads and distributive laws between reader and write monads.



### Update monad algebras as compat. compositions

An algebra of the update monad for S,  $(P, o, \oplus)$ ,  $\downarrow$  is a set  $\mathcal{A}$  carrying both the

satisfying an additional compatibility condition

$$\operatorname{\mathsf{upd}}(p,\operatorname{\mathsf{lkp}}(\lambda s'.a)) = \operatorname{\mathsf{lkp}}(\lambda s.\operatorname{\mathsf{upd}}(p,a[s\downarrow p/s']))$$



#### Buffers and truncation revisited

- $S = E^* \times Nat$  (current buffer content and free space)
- $P = E^*$  (new values to write)
- o = []
- $\bullet \ p \oplus p' = p + p'$
- $(s, n) \downarrow p = (s ++ (p|n), n length(p|n))$

 $(p|n \ defined \ as \ p \ truncated \ to \ length \ n)$ 

- $TX = (E^* \times Nat) \rightarrow (E^* \times X)$
- How to avoid truncation?



## A finer dependently-typed version

Rather than

$$S$$
 – a set  $(P, o, \oplus)$  – a monoid  $\downarrow$  – an action  $TX = S \rightarrow P \times X$ 

consider a *directed container*  $(S, P, \downarrow, o, \oplus)$ 

P a S-indexed family,

$$\downarrow: \Pi s: S. P s \rightarrow S$$
o:  $\Pi \{s: S\}. P s$ 

$$\oplus: \Pi \{s: S\}. \Pi p: P s. P (s \downarrow p) \rightarrow P s$$

$$TX = \Pi s : S. Ps \times X$$

S – states

Ps – updates *enabled* (or *safe*) in state s

#### Monads from directed containers

The def. of monad is the same (but with dependent typing):

$$TX = \Pi s : S.Ps \times X$$

$$\eta : \forall \{X\}.X \to \Pi s : S.Ps \times X$$

$$\eta x = \lambda s. (o, x)$$

$$(-)^* : \forall \{X, Y\}. (X \to \Pi s : S.Ps \times Y)$$

$$\to (\Pi s : S.Ps \times X) \to (\Pi s : S.Ps \times Y)$$

$$(f)^*(g) = \lambda s. \text{ let } (p, x) = g s;$$

$$(p', y) = f x (s \downarrow p)$$

$$\text{in } (p \oplus p', y)$$

Formally, it is the co-interpretation of directed containers

$$\langle\!\langle - \rangle\!\rangle^{\mathrm{dc}} : \mathsf{DCont}^{\mathrm{op}} \longrightarrow \mathsf{Monads}(\mathsf{Set})$$



# Example: writing into a buffer (a finer version)

- $S = E^* \times Nat$  (current buffer content and free space)
- $P(s,n) = E^{\leq n}$  (new values to write)
- o = []
- $\bullet \ p \oplus p' = p + p'$
- $(s, n) \downarrow p = (s ++ p, n length(p))$ 
  - no additional truncation needed!

•  $TX = \Pi(s, n) : E^* \times \text{Nat. } E^{\leq n} \times X$ 

#### Conclusion

- Update monads  $TX = S \rightarrow (P \times X)$  are a natural combination of the reader and writer monads
  - from a programming perspective
  - from a monadic perspective
  - from an algebraic perspective

- They are also a special case of a more general dependently-typed version
  - the co-interpretation of directed containers

### Connection to Kammar-Plotkin generalization

For a set S, a monoid  $(P, o, \oplus)$ , an action  $\downarrow$ , Kammar and Plotkin defined a generalized state monad as:

$$T_{KP} X = \Pi s : S.(s \downarrow P) \times X$$

 $T_{KP} X$  is the middle monad in the epi-mono factorization

$$TX = S \rightarrow (P \times X)$$

$$T_s X = S \rightarrow (S \times X)$$

$$T_{KP} X = \Pi s : S.(s \downarrow P) \times X$$

of the mon. morphism  $\tau = \lambda f. \lambda s. \operatorname{let}(p, x) = f s$  in  $(s \downarrow p, x)$