One Monad to the Tune of Another

real title: Dijkstra Monads For All

Robert Atkey Strathclyde University, Glasgow

jww Kenji Maillard, Danel Ahman, Guido Martínez, Cătălin Hriţcu, Exequiel Rivas, and Éric Tanter

> Shonan Meeting 146 27th March 2019

Modelling programs:

$$\llbracket P \rrbracket : X \to Y$$

Modelling programs with side effects:

State

$$\llbracket P \rrbracket : X \times S \longrightarrow Y \times S$$

Exceptions

$$\llbracket P \rrbracket : X \to Y + E$$

Non-determinism

$$\llbracket P \rrbracket : X \to \mathcal{P}_{fin}(Y)$$

I/O

$$\llbracket P \rrbracket : X \to IOTree(Y)$$

Monads as *Notions of Computation*:

(Moggi, 1989, 1991)

 $\llbracket P \rrbracket : X \to MY$

Monadic basics:

 $return : A \rightarrow MA$

 $bind \quad : \quad MA \to (A \to MA) \to MB$

Pick your monad:

Nothing: MA = A

State: $MA = S \rightarrow (A \times S)$

Exceptions: MA = A + E

Non-determinism: $MA = \mathcal{P}_{fin}(A)$

I/O: MA = IOTree(A)

Reasoning about monadic computations

Equational Reasoning

- . Monad laws
- . Monads generated from equational theories \leadsto equations for reasoning.
- . (Plotkin & Pretnar, 2008)

Evaluation Logic

(Pitts, 1991)

. only allows reasoning about returned values

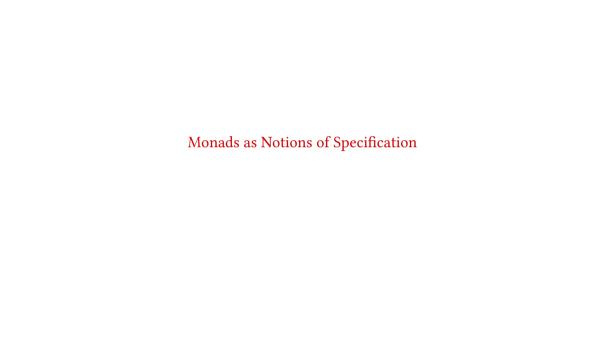
Indexed Monads

. {Parameterised / Graded / Poly}monads

Special Purpose Logics

- . Hoare Type Theory
- . Dynamic Logic

. ...



Many notions of specification are also monads.

Many notions of specification are also monads.

Weakest precondition transformer

$$WP(A) = (A \rightarrow Prop) \rightarrow Prop$$

Many notions of specification are also monads.

Weakest precondition transformer

$$WP(A) = (A \rightarrow Prop) \rightarrow Prop$$

Weakest precondition transformer with state

$$WP_S(A) = (A \to S \to \text{Prop}) \to S \to \text{Prop}$$

Many notions of specification are also monads.

Weakest precondition transformer

$$WP(A) = (A \rightarrow Prop) \rightarrow Prop$$

Weakest precondition transformer with state

$$WP_S(A) = (A \rightarrow S \rightarrow \text{Prop}) \rightarrow S \rightarrow \text{Prop}$$

Weakest precondition transformer with exceptions

$$WP_E(A) = (A + E \to \text{Prop}) \to \text{Prop} \cong (A \to \text{Prop}) \to (E \to \text{Prop}) \to \text{Prop}$$

Many notions of specification are also monads.

Many notions of specification are also monads.

Predicate Monad

$$P(A) = A \rightarrow \text{Prop}$$

Many notions of specification are also monads.

Predicate Monad

$$P(A) = A \rightarrow \text{Prop}$$

Pre-/Post-condition Monad

$$PP(A) = \text{Prop} \times (A \rightarrow \text{Prop})$$

Many notions of specification are also monads.

Predicate Monad

$$P(A) = A \rightarrow \text{Prop}$$

Pre-/Post-condition Monad

$$PP(A) = \text{Prop} \times (A \rightarrow \text{Prop})$$

For now, will use the weakest precondition transformer monads for specifications.

Useful specification monads are *ordered*:

- 1. partial order structure \sqsubseteq on WX
- 2. compatible with the *bind* operation

For WP, must restrict to monotone predicate transformers:

$$\mathit{MonoWP}(A) = \{ f : (A \to \operatorname{Prop}) \to \operatorname{Prop} \mid \forall q_1, q_2.q_1 \sqsubseteq q_2 \Rightarrow f \ q_1 \sqsubseteq f \ q_2 \}$$

then

$$f \sqsubseteq f' \Leftrightarrow \forall q : A \to \text{Prop. } f \neq f' \neq f'$$



Connecting the Computational and the Specificational

Monad morphisms:

$$\alpha: M \Rightarrow W$$

Families of functions $\alpha_A : MA \to WA$, compatible with *return* and *bind*.

Monad morphisms are "Effect Observations" in (Katsumata, 2014) (used to generate graded monads)

Connecting the Computational and the Specificational

Monad morphisms:

$$\alpha: M \Rightarrow W$$

Families of functions $\alpha_A: MA \to WA$, compatible with *return* and *bind*.

Monad morphisms are "Effect Observations" in (Katsumata, 2014) (used to generate graded monads)

Given a computation
$$m: MA$$
 a specification $w: WA$ m satisfies w iff $\alpha_A(m) \sqsubseteq w$

Example: Non-determinism

The non-determinism monad:

$$ND(A) = \mathcal{P}_{fin}(A)$$

Two morphisms $ND \Rightarrow MonWP$:

Demonic

$$\alpha^D(S) = \lambda post. \ \forall a \in S. \ post(a)$$

Angelic

$$\alpha^{A}(S) = \lambda post. \ \exists a \in S. \ post(a)$$

Demonic and Angelic Specifications

For the specification

$$w = (\lambda post. \ \forall x \ y \ z. \ x^2 + y^2 = z^2 \Rightarrow post(x, y, z)) \in MonWP(\mathbb{N} \times \mathbb{N} \times \mathbb{N})$$

$$\alpha^{D}($$
let $x \leftarrow pick [1, 2, 3, 4, 5]$
let $y \leftarrow pick [1, 2, 3, 4, 5]$
let $z \leftarrow pick [1, 2, 3, 4, 5]$
guard $(x^{2} + y^{2} = z^{2})$
return $(x, y, z)) \sqsubseteq w$

Demonic and Angelic Specifications

For the specification

$$w = (\lambda post. \ \forall x \ y \ z. \ x^2 + y^2 = z^2 \Rightarrow post(x, y, z)) \in MonWP(\mathbb{N} \times \mathbb{N} \times \mathbb{N})$$

$$\alpha^{A}(\text{let } x \leftarrow pick [1, 2, 3, 4, 5]$$

 $\text{let } y \leftarrow pick [1, 2, 3, 4, 5]$
 $\text{let } z \leftarrow pick [1, 2, 3, 4, 5]$
 $\text{guard } (x^{2} + y^{2} = z^{2})$
 $\text{return } (x, y, z)) \sqsubseteq w$

Demonic and Angelic Specifications

For the specification

$$w = (\lambda post. \ \forall x \ y \ z. \ x^2 + y^2 = z^2 \Rightarrow post(x, y, z)) \in MonWP(\mathbb{N} \times \mathbb{N} \times \mathbb{N})$$

$$\alpha^{A}(\text{let } x \leftarrow pick [1, 2, 3, 4, 5] \\ \text{let } y \leftarrow pick [1, 2, 3, 4, 5] \\ \text{let } z \leftarrow pick [1, 2, 3, 4, 5]$$

return
$$(x, y, z)$$
 $\sqsubseteq w$

Example: State

$$\alpha_A : \operatorname{St}(A) \longrightarrow \operatorname{MonWP}_S(A)$$
 $\alpha_A : (S \to (S \times A)) \longrightarrow (A \to S \to \operatorname{Prop}) \to_{mon} S \to \operatorname{Prop}$

$$\alpha_A \qquad m \qquad = \lambda \operatorname{post} s_0 \cdot \operatorname{post}(m s_0)$$

Spec:
$$w = (\lambda post \ s_0. \ \forall s. \ s > s_0 \Rightarrow post(s, *)) \in MonWP_S(1)$$

$$\alpha_1(\mathbf{let}\ x \leftarrow \mathit{get}; \mathit{put}(x+1)) \sqsubseteq w$$

Example: Exceptions

Example: Exceptions

1) "Double-Barrelled" specifications

$$\alpha_A$$
: Exc(A) \rightarrow MonWP_E(A)
 α_A : A + E \rightarrow (A + E \rightarrow Prop) \rightarrow_{mon} Prop
 α_A m = $\lambda post. post(m)$

Example: Exceptions

1) "Double-Barrelled" specifications

$$\alpha_A : \operatorname{Exc}(A) \to \operatorname{MonWP}_E(A)$$
 $\alpha_A : A + E \to (A + E \to \operatorname{Prop}) \to_{mon} \operatorname{Prop}$

$$\alpha_A = m = \lambda \operatorname{post.} \operatorname{post}(m)$$

2) "Single-Barrelled" specifications, for a fixed $Q_{exn}: E \to \text{Prop}$

$$\alpha_A$$
: Exc(A) \rightarrow MonWP(A)
 α_A : A + E \rightarrow (A \rightarrow Prop) \rightarrow_{mon} Prop

$$\alpha_A$$
 $m = \lambda post.$ **case** $m \{ inl \ a \mapsto post(a); inr \ e \mapsto Q_{exn}(e) \}$

Example: I/O

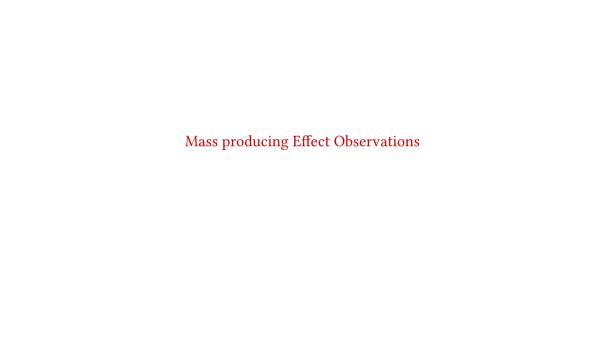
Let
$$Trace = List(I + O)$$
,

$$MonWPTrace(A) = (A \rightarrow Trace \rightarrow Prop) \rightarrow_{mon} Trace \rightarrow Prop$$

$$\begin{array}{lll} \alpha: \mathrm{IOTree}(A) & \longrightarrow & MonWPTrace(A) \\ \alpha(return\ a) & = & \lambda post\ t.\ post\ a\ t \\ \alpha(inp\ k) & = & \lambda post\ t.\ \forall i.\ \alpha(k\ i)\ post\ (\mathrm{In}\ i::\ t) \\ \alpha(out\ o\ k) & = & \lambda post\ t.\ \alpha(k)\ post\ (\mathrm{Out\ }o::\ t) \end{array}$$

Example (assuming I = O):

$$\alpha(\mathbf{let}\ x \leftarrow \mathit{inp}; \mathit{out}\ x; \mathit{out}\ x) \sqsubseteq \lambda p\ t.\ \forall x.\ p\ *\ (\mathrm{Out}\ x :: \mathrm{Out}\ x :: \mathrm{In}\ x :: t)$$



Monad Algebras

$$h: MR \rightarrow R$$

are in 1-1 correspondence with monad morphisms

$$\alpha: M \Longrightarrow (- \longrightarrow R) \longrightarrow R$$

(Kelly, 1980; Kelly and Power, 1993) ... and extends to the ordered case.

$$\alpha(m) = \lambda k. h(bind_M(m, k))$$

Demonic non-determinism:

$$h: \mathcal{P}_{fin}(\text{Prop}) \to \text{Prop}$$

 $h(S) = \bigwedge S$

gives

$$\alpha^D(S) = \lambda post. \ \forall x \in S. \ post(x)$$

Angelic non-determinism:

$$h: \mathcal{P}_{fin}(\text{Prop}) \to \text{Prop}$$

 $h(S) = \bigvee S$

gives

$$\alpha^{A}(S) = \lambda post. \ \exists x \in S. \ post(x)$$

State:

$$h: St(S \to Prop) \to S \to Prop$$

 $h(t) = \lambda s. let (s', p) = t s in p s'$

gives

$$\alpha(t) = \lambda post.\lambda s.let(s', a) = t s in post s'$$

Single-barrelled Exceptions:

$$h : \operatorname{Exc}(\operatorname{Prop}) \to \operatorname{Prop}$$

 $h (\operatorname{inl} p) = p$
 $h (\operatorname{inr} e) = Q_{exn} e$

gives

$$\alpha_A m = \lambda post.$$
 case $m \{ inl \ a \mapsto post(a); inr \ e \mapsto Q_{exn}(e) \}$

Example: Free Monad

Assume $\Sigma = \{op_1 : I_1 \leadsto O_1, \cdots, op_n : I_n \leadsto O_n\}.$

$$T_{\Sigma}A = \mu X.A + \coprod_{op:I \sim O \in \Sigma} I \times (O \to X)$$

Example: Free Monad

Assume $\Sigma = \{op_1 : I_1 \leadsto O_1, \cdots, op_n : I_n \leadsto O_n\}.$

$$T_{\Sigma}A = \mu X.A + \coprod_{op:I \sim O \in \Sigma} I \times (O \to X)$$

Operation specifications, for all *op*: $pre_{op}: I \rightarrow \text{Prop and } post_{op}: I \rightarrow O \rightarrow \text{Prop}$

$$\alpha : T_{\Sigma} \text{Prop} \to \text{Prop}$$

$$\alpha(ret \, \phi) = \phi$$

$$\alpha(op \ i \ k) = pre_{op} \ i \land \forall o. \ post_{op} \ i \ o \to k \ o$$

Monad Transformers

Assume that we have a monad transformer:

$$\mathcal{T}: \mathsf{Mon} \to \mathsf{Mon}$$

- . functor from monads to monads
- . equipped with lift : $M \Rightarrow \mathcal{T}M$
- . preserving order

Example:
$$\mathcal{T}(M) = A \mapsto S \to M(S \times A)$$

Given a suitable \mathcal{T} ,

$$\alpha = \mathcal{T}(return) : \mathcal{T}(Id) \to \mathcal{T}(MonCont)$$

DM4Free (POPL'17) presented this idea in a syntactic way.

Monad Transformers

$$\mathcal{T}(M) = A \mapsto M(A + E)$$

Then:
$$\mathcal{T}(\mathrm{Id}) = \mathrm{Exn}$$

 $\mathcal{T}(\mathrm{MonCont}) = A \mapsto (A + E \to \mathrm{Prop}) \to \mathrm{Prop}$
 $\alpha(m) = \lambda post. post(m)$

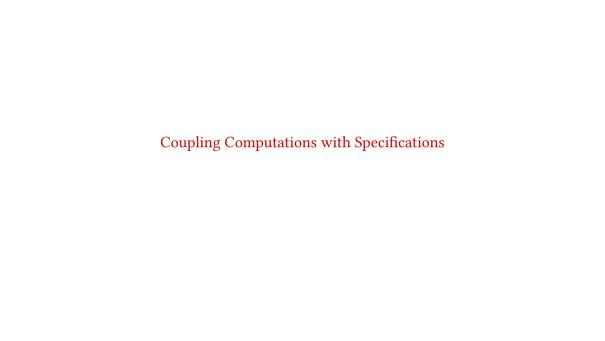
Also works for State, State(Exn), Exn(State), \dots

I/O

$$\mathcal{T}(M) = A \mapsto \mu X.M(A + (O \times X) + (I \to X))$$

$$\mathcal{T}(\text{MonProp}) = A \mapsto \mu X.(A + (O \times X) + (I \to X) \to \text{Prop}) \to \text{Prop}$$

but this doesn't exist in Set; and would make Coq and F# inconsistent...



Coupling Computations with Specifications

The specification $\alpha(m) \sqsubseteq w$ could be tricky to check.

Coupling Computations with Specifications

The specification $\alpha(m) \sqsubseteq w$ could be tricky to check.

Restrict to computations m that satisfy a specification w:

$$D: (A: Set) \to WA \to Set$$
$$DAw = \{m: MA \mid \alpha(m) \sqsubseteq w\}$$

Coupling Computations with Specifications

The specification $\alpha(m) \sqsubseteq w$ could be tricky to check.

Restrict to computations m that satisfy a specification w:

$$D: (A: Set) \to WA \to Set$$
$$DAw = \{m: MA \mid \alpha(m) \sqsubseteq w\}$$

Define:

return:
$$(x:A) \rightarrow DA$$
 (return_W x)
return $x = return_M a$
bind: $DA w_1 \rightarrow ((x:A) \rightarrow DB(w_2 x)) \rightarrow DB$ (bind_W $w_1 w_2$)
bind $m_1 m_2 = bind_M m_1 m_2$
weaken: $(w_1 \sqsubseteq w_2) \rightarrow DA w_1 \rightarrow DA w_2$
weaken $m = m$

Dijkstra Monads over a (specification) monad W:

$$D: (A: Set) \to WA \to Set$$

$$return: (x: A) \to DA(return_W x)$$

$$bind: DA w_1 \to ((x: A) \to DB(w_2 x)) \to DB(bind_W w_1 w_2)$$

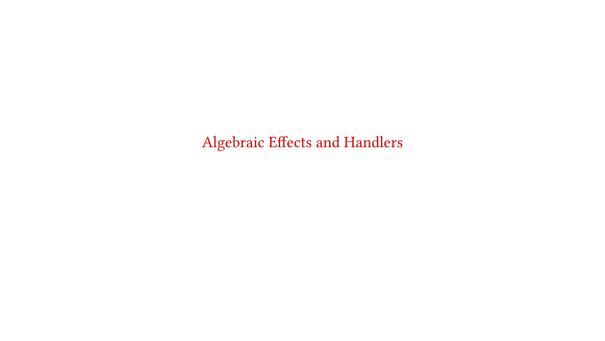
$$weaken: (w_1 \sqsubseteq w_2) \to DA w_1 \to DA w_2$$

with some laws.

Effect Observations and Dijkstra Monads

$$\mathrm{DMon}(X) \simeq \mathrm{Mon}/W$$

an equivalence of categories.



Algebraic Effects

If $op^M : I \times (O \rightarrow MA) \rightarrow MA$ is algebraic

and $\alpha: M \Rightarrow W$ is an effect observation,

then

$$op^{W}(i, w) = \mu^{W}(\alpha(op^{M}(i, \lambda o. return^{M}(w o))))$$

is algebraic, and serves to be the specification for the operation op^{M} :

$$op^D: (i:I) \rightarrow (c:(o:O) \rightarrow DA(wo)) \rightarrow DA(op^W(i,w))$$

Handlers, attempt 1

handle :
$$T_{\Sigma}A \rightarrow (I \times (O \rightarrow MB) \rightarrow MB)_{op:I \sim O} \rightarrow (A \rightarrow MB) \rightarrow MB$$

Handlers, attempt 1

handle :
$$T_{\Sigma}A \rightarrow (I \times (O \rightarrow MB) \rightarrow MB)_{op:I \rightarrow O} \rightarrow (A \rightarrow MB) \rightarrow MB$$

handle^D:
$$D_{\Sigma} A w_1 \rightarrow (H_{op}^{W'}: I \times (O \rightarrow W'B) \rightarrow W'B)_{op:I \rightsquigarrow O} \rightarrow ((i:I) \rightarrow ((o:O) \rightarrow D'B(wo)) \rightarrow D'B(h_{op}^{W'}(i,w)))_{op:I \rightsquigarrow O} \rightarrow ((a:A) \rightarrow D'B(w_2 a)) \rightarrow D'B(handle w_1 (h_{HW'})_* w_2)$$

where $h_*: WW'B \to W'B$ whenever $h: TW'B \to W'B$.

not automatic; needs to be established for each $\alpha: M \Rightarrow W$ and W'.

Handlers: lifting algebras

For exceptions, $\alpha : \text{Exn} \Rightarrow \text{ExnWP} = ((- + E \rightarrow \text{Prop}) \rightarrow \text{Prop})$

Possible to take $h : \text{Exn}(\text{ExnWP}(B)) \rightarrow \text{ExnWP}(B)$

to $h_* : \text{ExnWP}(\text{ExnWP}(B)) \rightarrow \text{ExnWP}(B)$

Handlers: lifting algebras

For exceptions, $\alpha : \text{Exn} \Rightarrow \text{ExnWP} = ((- + E \rightarrow \text{Prop}) \rightarrow \text{Prop})$

Possible to take
$$h : \text{Exn}(\text{ExnWP}(B)) \rightarrow \text{ExnWP}(B)$$

to $h_* : \text{ExnWP}(\text{ExnWP}(B)) \rightarrow \text{ExnWP}(B)$

For I/O, α : IO \Rightarrow MonWPTrace, not possible to do the lifting.

The specification monad

$$\mathcal{T}(\text{MonProp}) = A \mapsto \mu X.(A + (O \times X) + (I \to X) \to \text{Prop}) \to \text{Prop}$$

"works", but doesn't exist in categories/theories of interest.

Handlers, attempt 2

Problem seems to be:

- trying to get the "most general" specification for the handled computation
- ▶ then try to instantiate that specification with the spec of the handler
- but we get circularity between the handler behaviour and the handled's behaviour

A possible solution

- ightharpoonup Assume some pre_{op} and $post_{op}$ specification for the operations
- ▶ Handler of an operation op(i, k):
 - ightharpoonup Assumes $pre_{op}(i)$
 - ► Must establish $post_{op}(i, o)$ before invoking k o

Sort of works, but only for handling into a Disjktra monad (can't write state handler).

Conclusions

- ► Monads as notions of Specification
- ► Effect observations = monad morphisms
- ► Packaged up as Dijkstra Monads

- ► Algebraic *effects* work well
- ► Handlers are a mess