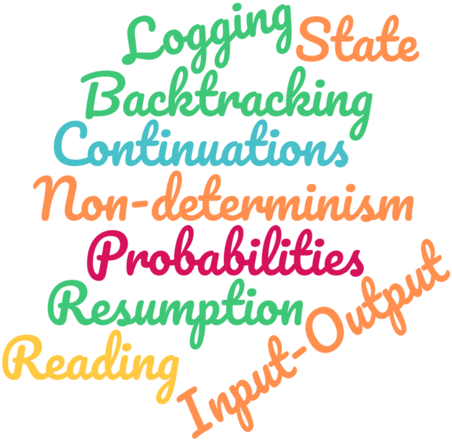
Designing Dijkstra Monads

Kenji Maillard
j.w.w. D. Ahman, B. Atkey, C.Hrițcu, G.Martinez, E.Tanter, E. Rivas

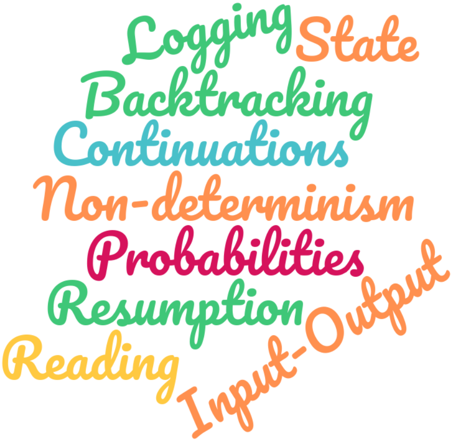
Eutypes'19 WG, Krakow

February 23, 2019

Programming with side effects



Programming with side effects



We need a model to uniformly account for side-effects !

Monads: a nifty algebraic tool

$\mathbb{M}X$ represents a computational context producing values in X

Monads: a nifty algebraic tool

$\mathrm{M}X$ represents a computational context producing values in X

A succint syntactic presentation: a type constructor M

$$\mathtt{ret}^{\mathrm{M}} : X \to \mathrm{M}X \qquad \mathtt{bind}^{\mathrm{M}} : \mathrm{M}X \to (X \to \mathrm{M}Y) \to \mathrm{M}Y$$

$+$ 3 equations (monad laws)

Examples of computational monads

State

$\mathrm{St}\, X = \mathcal{S} \to X \times \mathcal{S}$

Examples of computational monads

State

$\mathrm{St}\, X = \mathcal{S} \to X \times \mathcal{S}$

Exceptions

$\mathrm{Exc}\, X = X + \mathcal{E}$

Examples of computational monads

State
$\mathrm{St}\, X = \mathcal{S} \to X \times \mathcal{S}$

Exceptions
$\mathrm{Exc}\, X = X + \mathcal{E}$

Partiality
$\mathrm{Div}\, X = X + \{\Omega\}$

Examples of computational monads

STATE
$\mathrm{St}\, X = \mathcal{S} \to X \times \mathcal{S}$

EXCEPTIONS
$\mathrm{Exc}\, X = X + \mathcal{E}$

PARTIALITY
$\mathrm{Div}\, X = X + \{\Omega\}$

INPUT-OUTPUT
$\mathrm{IO}\, X = \mu Z. X + (Z \times \mathcal{O}) + (\mathcal{I} \to Z)$

Examples of computational monads

State
$\mathrm{St}\,X = \mathcal{S} \to X \times \mathcal{S}$

Exceptions
$\mathrm{Exc}\,X = X + \mathcal{E}$

Partiality
$\mathrm{Div}\,X = X + \{\Omega\}$

Input-Output
$\mathrm{IO}\,X = \mu Z.X + (Z \times \mathcal{O}) + (\mathcal{I} \to Z)$

Continuations
$\mathrm{Cont}_{\mathrm{Ans}}\,X = (X \to \mathrm{Ans}) \to \mathrm{Ans}$

Examples of computational monads

STATE
$\mathrm{St}\,X = \mathcal{S} \to X \times \mathcal{S}$

EXCEPTIONS
$\mathrm{Exc}\,X = X + \mathcal{E}$

PARTIALITY
$\mathrm{Div}\,X = X + \{\Omega\}$

INPUT-OUTPUT
$\mathrm{IO}\,X = \mu Z.X + (Z \times \mathcal{O}) + (\mathcal{I} \to Z)$

CONTINUATIONS
$\mathrm{Cont}_{\mathrm{Ans}}\,X = (X \to \mathrm{Ans}) \to \mathrm{Ans}$

NON-DETERMINISM
$\mathrm{NDet}\,X = \mathcal{P}_{fin}(X)$

Examples of computational monads

State
$\mathrm{St}\,X = \mathcal{S} \to X \times \mathcal{S}$

Exceptions
$\mathrm{Exc}\,X = X + \mathcal{E}$

Partiality
$\mathrm{Div}\,X = X + \{\Omega\}$

Input-Output
$\mathrm{IO}\,X = \mu Z.X + (Z \times \mathcal{O}) + (\mathcal{I} \to Z)$

Continuations
$\mathrm{Cont}_{\mathrm{Ans}}\,X = (X \to \mathrm{Ans}) \to \mathrm{Ans}$

Non-determinism
$\mathrm{NDet}\,X = \mathcal{P}_{fin}(X)$

Probabilities
$\mathrm{Prob}\,X = \{\ f : X \to [0;1] \ \mid\ \mathrm{supp}(f) \text{ finite}, \Sigma_{x \in \mathrm{supp}(f)} f\,x \leq 1\ \}$

Monadic programming

Moggi's computational lambda-calculus (λ_C)(Moggi, 1989)

$$
\begin{aligned}
T &::= \mathrm{M}\,T \mid T_1 \to T_2 \mid T_1 \times T_2 \mid T_1 + T_2 && \text{types} \\
v &::= x \mid \lambda x.\, c \mid \langle c_1, c_2 \rangle \mid \iota_i\ v && \text{values} \\
c &::= v && \text{returning values} \\
&\quad \mid \texttt{let}\, x = c_1\, \texttt{in}\, c_2 && \text{sequencing computations} \\
&\quad \mid \texttt{op}(c_1, \ldots, c_n) && \text{effectful operations} \\
&\quad \mid c_1\ c_2 \mid \pi_i\ c \mid \texttt{case}\ c\ x.c_1\ y.c_2
\end{aligned}
$$

Monadic programming

Moggi's computational lambda-calculus (λ_C)(Moggi, 1989)

$T ::= \mathrm{M}\ T \mid T_1 \to T_2 \mid T_1 \times T_2 \mid T_1 + T_2$	types
$v ::= x \mid \lambda x.\, c \mid \langle c_1, c_2 \rangle \mid \iota_i\ v$	values
$c ::= v$	returning values
$\mid \texttt{let}\ x = c_1\ \texttt{in}\ c_2$	sequencing computations
$\mid \mathtt{op}(c_1, \ldots, c_n)$	effectful operations
$\mid c_1\ c_2 \mid \pi_i\ c \mid \texttt{case}\ c\ x.c_1\ y.c_2$	

Ret

$$\frac{\Gamma \vdash_{val} v : T}{\Gamma \vdash v : \mathrm{M}\ T}$$

Bind

$$\frac{\Gamma \vdash c_1 : \mathrm{M}\ T_1 \qquad \Gamma, x : T_1 \vdash c_2 : \mathrm{M}\ T_2}{\Gamma \vdash \texttt{let}\ x = c_1\ \texttt{in}\ c_2 : \mathrm{M}\ T_2}$$

Operations associated to monads

a.k.a. **generic effects** (Plotkin and Power, 2003)

STATE

$\texttt{get} : \mathbb{1} \to \mathrm{St}\,\mathcal{S}$

$\texttt{put} : \mathcal{S} \to \mathrm{St}\,\mathbb{1}$

INPUT-OUTPUT

$\texttt{read} : \mathbb{1} \to \mathrm{IO}\,\mathcal{I}$

$\texttt{write} : \mathcal{O} \to \mathrm{IO}\,\mathbb{1}$

Operations associated to monads

a.k.a. **generic effects** (Plotkin and Power, 2003)

State

$\mathtt{get} : \mathbb{1} \to \mathrm{St}\,\mathcal{S}$

$\mathtt{put} : \mathcal{S} \to \mathrm{St}\,\mathbb{1}$

Input-Output

$\mathtt{read} : \mathbb{1} \to \mathrm{IO}\,\mathcal{I}$

$\mathtt{write} : \mathcal{O} \to \mathrm{IO}\,\mathbb{1}$

Exceptions

$\mathtt{throw} : \mathcal{E} \to \mathrm{Exc}\,\mathbb{0}$

Partiality

$\mathtt{div} : \mathbb{1} \to \mathrm{Div}\,\mathbb{0}$

Operations associated to monads

a.k.a. **generic effects** (Plotkin and Power, 2003)

STATE

$\texttt{get} : \mathbb{1} \to \mathrm{St}\,\mathcal{S}$

$\texttt{put} : \mathcal{S} \to \mathrm{St}\,\mathbb{1}$

INPUT-OUTPUT

$\texttt{read} : \mathbb{1} \to \mathrm{IO}\,\mathcal{I}$

$\texttt{write} : \mathcal{O} \to \mathrm{IO}\,\mathbb{1}$

EXCEPTIONS

$\texttt{throw} : \mathcal{E} \to \mathrm{Exc}\,\mathbb{0}$

PARTIALITY

$\texttt{div} : \mathbb{1} \to \mathrm{Div}\,\mathbb{0}$

NON-DETERMINISM

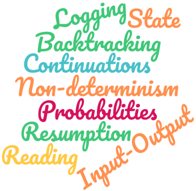
$\texttt{choose} : \mathbb{1} \to \mathrm{NDet}\,\mathbb{B}$

$\texttt{fail} : \mathbb{1} \to \mathrm{NDet}\,\mathbb{0}$

PROBABILITIES

$\texttt{flip} : [0;1] \to \mathrm{Prob}\,\mathbb{B}$

Starting point

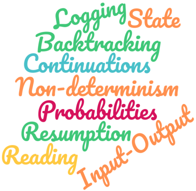
Fix M a computational monad $\in$ 

Take a program $c : \mathrm{M}\mathbb{N}$, for instance

```
let x = read () in
let y = read () in
if y = 0 then throw Div_by_zero else x/y
```

Starting point

Fix M a computational monad $\in$



Take a program $c : \mathrm{M}\mathbb{N}$, for instance

```
let x = read () in
let y = read () in
if y = 0 then throw Div_by_zero else x/y
```

How can we specify and verify such monadic programs ?

Hoare logic Primer

$$\{\ pre\ \}\ code\ \{\ post\ \}$$

$$\{\ \top\ \}\ v\ \{\ x = v\ \}$$

$$\frac{\{\ P\ \}\ c_1\ \{\ Q\ \} \qquad \forall x, \{\ Q(x)\ \}\ c_2\ \{\ R\ \}}{\{\ P\ \}\ \texttt{let}\ x = c_1\ \texttt{in}\ c_2\ \{\ R\ \}}$$

Hoare logic Primer

$$\{\, pre \,\}\ code\ \{\, post \,\}$$

$$\{\, \top \,\}\ v\ \{\, x = v \,\} \qquad \frac{\{\, P \,\}\ c_1\ \{\, Q \,\} \qquad \forall x, \{\, Q(x) \,\}\ c_2\ \{\, R \,\}}{\{\, P \,\}\ \texttt{let}\ x = c_1\ \texttt{in}\ c_2\ \{\, R \,\}}$$

Pure:	$pre : \mathbb{P}$	$post : X \to \mathbb{P}$
Stateful:	$pre : S \to \mathbb{P}$	$post : X \times S \to \mathbb{P}$
With exceptions:	$pre : \mathbb{P}$	$post : X + E \to \mathbb{P}$

Weakest precondition calculi

Dijkstra's insight: there is a **weakest** precondition that can be computed from a program and a postcondition

$$\vdash \{ P \} \; c \; \{ Q \} \qquad \Longleftrightarrow \qquad \vdash P \Rightarrow \mathtt{wp}[c](Q)$$

$$\mathtt{wp}[v](Q) = Q(v) \qquad \mathtt{wp}[\mathtt{let}\, x = c_1 \,\mathtt{in}\, c_2](Q) = \mathtt{wp}[c_1](\lambda x.\, \mathtt{wp}[c_2](Q))$$

Weakest precondition calculi

Dijkstra's insight: there is a **weakest** precondition that can be computed from a program and a postcondition

$$\vdash \{\, P \,\}\; c \;\{\, Q \,\} \qquad \Longleftrightarrow \qquad \vdash P \Rightarrow \mathtt{wp}[c](Q)$$

$$\mathtt{wp}[v](Q) = Q(v) \qquad \mathtt{wp}[\mathtt{let}\, x = c_1 \,\mathtt{in}\, c_2](Q) = \mathtt{wp}[c_1](\lambda x.\, \mathtt{wp}[c_2](Q))$$

Pure: $\mathtt{wp}[c](\cdot) : (X \to \mathbb{P}) \to \mathbb{P}$

Stateful: $\mathtt{wp}[c](\cdot) : (X \times S \to \mathbb{P}) \to S \to \mathbb{P}$

With exceptions: $\mathtt{wp}[c](\cdot) : (X + E \to \mathbb{P}) \to \mathbb{P}$

Wait. . . That's a monad !

Pure: $\mathrm{W}^{\mathrm{Id}} : (X \to \mathbb{P}) \to \mathbb{P}$

$$\mathtt{ret}^{\mathrm{W}^{\mathrm{Id}}} : X \to \mathrm{W}^{\mathrm{Id}} X \quad \mathtt{bind}^{\mathrm{W}^{\mathrm{Id}}} : \mathrm{W}^{\mathrm{Id}} X \to (X \to \mathrm{W}^{\mathrm{Id}} Y) \to \mathrm{W}^{\mathrm{Id}} Y$$

$$\mathtt{ret}^{\mathrm{W}^{\mathrm{Id}}}\ x\ Q = Q(x) \qquad \mathtt{bind}^{\mathrm{W}^{\mathrm{Id}}}\ w_1\ w_2\ Q = w_1(\lambda x.\, w_2(x)(Q))$$

Wait... That's a monad !

Pure: $\mathrm{W}^{\mathrm{Id}} : (X \to \mathbb{P}) \to \mathbb{P}$

$$\texttt{ret}^{\mathrm{W}^{\mathrm{Id}}} : X \to \mathrm{W}^{\mathrm{Id}} X \quad \texttt{bind}^{\mathrm{W}^{\mathrm{Id}}} : \mathrm{W}^{\mathrm{Id}} X \to (X \to \mathrm{W}^{\mathrm{Id}} Y) \to \mathrm{W}^{\mathrm{Id}} Y$$

$$\texttt{ret}^{\mathrm{W}^{\mathrm{Id}}}\ x\ Q = Q(x) \quad \texttt{bind}^{\mathrm{W}^{\mathrm{Id}}}\ w_1\ w_2\ Q = w_1(\lambda x.\, w_2(x)(Q))$$

Continuation monad with answer type $\mathbb{P}$:

$$\mathrm{W}^{\mathrm{Id}} X = \mathrm{Cont}_{\mathbb{P}}(X) = (X \to \mathbb{P}) \to \mathbb{P}$$

Wait... That's a monad !

$$\textbf{Pure:} \qquad \mathrm{W}^{\mathrm{Id}} : (X \to \mathbb{P}) \to \mathbb{P}$$

$$\texttt{ret}^{\mathrm{W}^{\mathrm{Id}}} : X \to \mathrm{W}^{\mathrm{Id}} X \qquad \texttt{bind}^{\mathrm{W}^{\mathrm{Id}}} : \mathrm{W}^{\mathrm{Id}} X \to (X \to \mathrm{W}^{\mathrm{Id}} Y) \to \mathrm{W}^{\mathrm{Id}} Y$$

$$\texttt{ret}^{\mathrm{W}^{\mathrm{Id}}}\ x\ Q = Q(x) \qquad \texttt{bind}^{\mathrm{W}^{\mathrm{Id}}}\ w_1\ w_2\ Q = w_1(\lambda x.\, w_2(x)(Q))$$

Continuation monad with answer type $\mathbb{P}$:

$$\mathrm{W}^{\mathrm{Id}} X = \mathrm{Cont}_{\mathbb{P}}(X) = (X \to \mathbb{P}) \to \mathbb{P}$$

$$\textbf{Stateful:} \qquad \mathrm{W}^{\mathrm{St}} : (X \times S \to \mathbb{P}) \to S \to \mathbb{P}$$

$$\textbf{With exceptions:} \qquad \mathrm{W}^{\mathrm{Exc}} : (X + E \to \mathbb{P}) \to \mathbb{P}$$

Specification monads

Key idea: **specifications** can also be uniformly captured by monads!

Weakest precondition: $\mathrm{Cont}_{\mathbb{P}}\, X = (X \to \mathbb{P}) \to \mathbb{P}$

Strongest postcondition: $\mathrm{StrPost}\, X = \mathbb{P} \to X \to \mathbb{P}$

Pre/Postconditions: $\mathrm{PrePost}\, X = \mathbb{P} \times (X \to \mathbb{P})$

Specification monads

Key idea: **specifications** can also be uniformly captured by monads!

Weakest precondition:	$\mathrm{Cont}_{\mathbb{P}}\, X = (X \to \mathbb{P}) \to \mathbb{P}$
Strongest postcondition:	$\mathrm{StrPost}\, X = \mathbb{P} \to X \to \mathbb{P}$
Pre/Postconditions:	$\mathrm{PrePost}\, X = \mathbb{P} \times (X \to \mathbb{P})$

What's in a specification monad W ?

- specifications can be compared by strength

$$w_1 \leq^{\mathrm{Cont}_{\mathbb{P}}\, X} w_2 \qquad \Leftrightarrow \qquad \forall p : X \to \mathbb{P}, w_1\, p \implies w_2\, p$$

- $\texttt{bind}^{\mathrm{W}}$ is monotonic in both its arguments

$\rightsquigarrow$ restriction to monotonic elements in $\mathrm{Cont}_{\mathbb{P}}, \mathrm{StrPost}$

Predicate transformers from monad transformers

Examples of **predicate transformer** monads:

Pure: $\mathrm{W}^{\mathrm{Id}} : (X \to \mathbb{P}) \to \mathbb{P}$

Stateful: $\mathrm{W}^{\mathrm{St}} : (X \times \mathcal{S} \to \mathbb{P}) \to \mathcal{S} \to \mathbb{P}$

With exceptions: $\mathrm{W}^{\mathrm{Exc}} : (X + \mathcal{E} \to \mathbb{P}) \to \mathbb{P}$

Predicate transformers from monad transformers

Examples of **predicate transformer** monads:

Pure:	$\mathrm{W}^{\mathrm{Id}} : (X \to \mathbb{P}) \to \mathbb{P}$
Stateful:	$\mathrm{W}^{\mathrm{St}} : (X \times \mathcal{S} \to \mathbb{P}) \to \mathcal{S} \to \mathbb{P}$
With exceptions:	$\mathrm{W}^{\mathrm{Exc}} : (X + \mathcal{E} \to \mathbb{P}) \to \mathbb{P}$

$$\mathrm{W}^{\mathrm{Id}} = \mathcal{T}^{\mathrm{Id}}(\mathrm{Cont}_{\mathbb{P}}) \qquad \mathcal{T}^{\mathrm{Id}}(\mathrm{M}) = \mathrm{M}$$

$$\mathrm{W}^{\mathrm{St}} = \mathcal{T}^{\mathrm{St}}(\mathrm{Cont}_{\mathbb{P}}) \qquad \mathcal{T}^{\mathrm{St}}(\mathrm{M}) = \mathcal{S} \to \mathrm{M}(- \times \mathcal{S})$$

$$\mathrm{W}^{\mathrm{Exc}} = \mathcal{T}^{\mathrm{Exc}}(\mathrm{Cont}_{\mathbb{P}}) \qquad \mathcal{T}^{\mathrm{Exc}}(\mathrm{M}) = \mathrm{M}(- + \mathcal{E})$$

Predicate transformers from monad transformers

Examples of **predicate transformer** monads:

Pure:	$\mathrm{W}^{\mathrm{Id}} : (X \to \mathbb{P}) \to \mathbb{P}$
Stateful:	$\mathrm{W}^{\mathrm{St}} : (X \times \mathcal{S} \to \mathbb{P}) \to \mathcal{S} \to \mathbb{P}$
With exceptions:	$\mathrm{W}^{\mathrm{Exc}} : (X + \mathcal{E} \to \mathbb{P}) \to \mathbb{P}$

$$\mathrm{W}^{\mathrm{Id}} = \mathcal{T}^{\mathrm{Id}}(\mathrm{Cont}_{\mathbb{P}}) \qquad \mathcal{T}^{\mathrm{Id}}(\mathrm{M}) = \mathrm{M}$$

$$\mathrm{W}^{\mathrm{St}} = \mathcal{T}^{\mathrm{St}}(\mathrm{Cont}_{\mathbb{P}}) \qquad \mathcal{T}^{\mathrm{St}}(\mathrm{M}) = \mathcal{S} \to \mathrm{M}(- \times \mathcal{S})$$

$$\mathrm{W}^{\mathrm{Exc}} = \mathcal{T}^{\mathrm{Exc}}(\mathrm{Cont}_{\mathbb{P}}) \qquad \mathcal{T}^{\mathrm{Exc}}(\mathrm{M}) = \mathrm{M}(- + \mathcal{E})$$

Monad transformer $\mathcal{T}$:

- map a monad M to a monad $\mathcal{T}\mathrm{M}$
- $\texttt{lift}^{\mathcal{T}} : \mathrm{M} \to \mathcal{T}\mathrm{M}$

Relating computations & specifications

$\mathbb{M}$

computational monad

code

$c : \mathbb{M}\ \mathbb{N}$

$\mathbb{W}$

specification monad

specification

$w_c : \mathbb{W}\ \mathbb{N}$

Relating computations & specifications

$\mathbb{M}$

computational monad

code

$c : \mathbb{M}\ \mathbb{N}$

$\mathbb{W}$

specification monad

specification

$w_c : \mathbb{W}\ \mathbb{N}$

- ▷ For a fixed $\mathbb{M}$, is there a canonical $\mathbb{W}$?

Relating computations & specifications

$$\mathrm{M} \xleftrightarrow{\theta} \mathrm{W}$$

computational monad

specification monad

code

$c : \mathrm{M}\ \mathbb{N}$

specification

$w_c : \mathrm{W}\ \mathbb{N}$

- For a fixed M, is there a canonical W ?
- What kind of structure θ relate M and W ?

Relating computations & specifications

$$\mathrm{M} \xleftrightarrow{\quad\theta\quad} \mathrm{W}$$

computational monad

specification monad

code

$c : \mathrm{M}\ \mathbb{N}$

specification

$w_c : \mathrm{W}\ \mathbb{N}$

- For a fixed M, is there a canonical W ?
- What kind of structure θ relate M and W ?
- Is θ canonical wrt. M and W ?

Specifying programs with exceptions

$$\theta^{\mathrm{Exc}} \quad : \quad \mathrm{Exc}\, X \quad \longrightarrow \quad \mathrm{W}^{\mathrm{Exc}}\, X = (X + \mathcal{E} \to \mathbb{P}) \to \mathbb{P}$$

$$\theta^{\mathrm{Exc}}(m) = \lambda p.\, p\, m$$

Specifying programs with exceptions

$$\theta^{\mathrm{Exc}} \quad : \quad \mathrm{Exc}\, X \quad \longrightarrow \quad \mathrm{W}^{\mathrm{Exc}}\, X = (X + \mathcal{E} \to \mathbb{P}) \to \mathbb{P}$$

$$\theta^{\mathrm{Exc}}(m) = \lambda p.\, p\, m$$

$$\theta^{\mathrm{Exc}}(\mathtt{ret}^{\mathrm{Exc}}\, v) = \mathtt{ret}^{\mathrm{W}^{\mathrm{Exc}}}\, v$$

$$\theta^{\mathrm{Exc}}(\mathtt{bind}^{\mathrm{Exc}}\, m\ f) = \mathtt{bind}^{\mathrm{W}^{\mathrm{Exc}}}\, (\theta^{\mathrm{Exc}}\, m)\ (\theta^{\mathrm{Exc}} \circ f)$$

Specifying programs with exceptions

$$\theta^{\mathrm{Exc}} \quad : \quad \mathrm{Exc}\, X \quad \longrightarrow \quad \mathrm{W}^{\mathrm{Exc}}\, X = (X + \mathcal{E} \to \mathbb{P}) \to \mathbb{P}$$

$$\theta^{\mathrm{Exc}}(m) = \lambda p.\, p\, m$$

$$\theta^{\mathrm{Tot}} \quad : \quad \mathrm{Exc}\, X \quad \longrightarrow \quad \mathrm{W}^{\mathrm{Id}}\, X = (X \to \mathbb{P}) \to \mathbb{P}$$

$$\theta^{\mathrm{Tot}}(\mathtt{inr}\, v) = \lambda p.\, p\, v \qquad\qquad \theta^{\mathrm{Tot}}(\mathtt{inl}\, e) = \lambda p.\, \bot$$

Specifying programs with exceptions

$$\theta^{\mathrm{Exc}} \quad : \quad \mathrm{Exc}\, X \quad \longrightarrow \quad \mathrm{W}^{\mathrm{Exc}}\, X = (X + \mathcal{E} \to \mathbb{P}) \to \mathbb{P}$$

$$\theta^{\mathrm{Exc}}(m) = \lambda p.\, p\, m$$

$$\theta^{\mathrm{Tot}} \quad : \quad \mathrm{Exc}\, X \quad \longrightarrow \quad \mathrm{W}^{\mathrm{Id}}\, X = (X \to \mathbb{P}) \to \mathbb{P}$$

$$\theta^{\mathrm{Tot}}(\mathtt{inr}\, v) = \lambda p.\, p\, v \qquad \theta^{\mathrm{Tot}}(\mathtt{inl}\, e) = \lambda p.\, \bot$$

$$\theta^{\mathrm{Part}} \quad : \quad \mathrm{Exc}\, X \quad \longrightarrow \quad \mathrm{W}^{\mathrm{Id}}\, X = (X \to \mathbb{P}) \to \mathbb{P}$$

$$\theta^{\mathrm{Part}}(\mathtt{inr}\, v) = \lambda p.\, p\, v \qquad \theta^{\mathrm{Part}}(\mathtt{inl}\, e) = \lambda p.\, \top$$

Effect observation

An effect observation θ (Katsumata, 2014)

$$\mathrm{M} \xrightarrow{\quad\theta\quad} \mathrm{W}$$

is a **monad morphism**, that is

$$\theta(\mathtt{ret}^{\mathrm{M}}\ v) = \mathtt{ret}^{\mathrm{W}}\ v \qquad \theta(\mathtt{bind}^{\mathrm{M}}\ m\, f) = \mathtt{bind}^{\mathrm{W}}\ (\theta\ m)\ (\theta \circ f)$$

Relating stateful computations & specifications

$$\theta^{\mathrm{St}} : \begin{cases} \mathcal{S} \to X \times \mathcal{S} & \longrightarrow & \mathrm{W}^{\mathrm{St}}\, X = (X \times \mathcal{S} \to \mathbb{P}) \to \mathcal{S} \to \mathbb{P} \\ m & \longmapsto & \lambda p\, s_0.\, \texttt{let}\ \langle r,\ s_1 \rangle = m\, s_0\ \texttt{in}\ p\ \langle r,\ s_1 \rangle \end{cases}$$

Relating stateful computations & specifications

$$\theta^{\mathrm{St}} : \begin{cases} \mathcal{S} \to X \times \mathcal{S} & \longrightarrow & \mathrm{W}^{\mathrm{St}}\, X = (X \times \mathcal{S} \to \mathbb{P}) \to \mathcal{S} \to \mathbb{P} \\ m & \longmapsto & \lambda p\, s_0.\, \mathtt{let}\ \langle r,\ s_1 \rangle = m\, s_0\ \mathtt{in}\ p\ \langle r,\ s_1 \rangle \end{cases}$$

$$\frac{\mathrm{Id} \xrightarrow{\quad\mathtt{ret}\quad} \mathrm{Cont}_{\mathbb{P}}}{\theta^{\mathrm{St}} : \mathcal{T}^{\mathrm{St}}(\mathrm{Id}) \xrightarrow{\quad\mathcal{T}^{\mathrm{St}}(\mathtt{ret})\quad} \mathcal{T}^{\mathrm{St}}(\mathrm{Cont}_{\mathbb{P}})}$$

Relating stateful computations & specifications

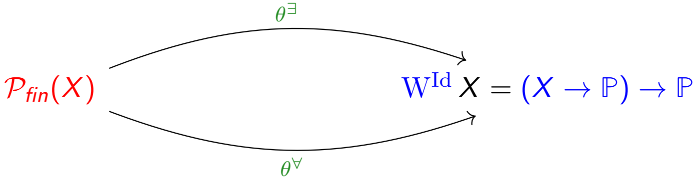
$$\theta^{\mathrm{St}} : \begin{cases} \mathcal{S} \to X \times \mathcal{S} & \longrightarrow & \mathrm{W}^{\mathrm{St}}\, X = (X \times \mathcal{S} \to \mathbb{P}) \to \mathcal{S} \to \mathbb{P} \\ m & \longmapsto & \lambda p\, s_0.\, \mathtt{let}\ \langle r,\ s_1 \rangle = m\, s_0\ \mathtt{in}\ p\ \langle r,\ s_1 \rangle \end{cases}$$

$$\frac{\mathrm{Id} \xrightarrow{\quad\mathtt{ret}\quad} \mathrm{Cont}_{\mathbb{P}}}{\theta^{\mathrm{St}} : \mathcal{T}^{\mathrm{St}}(\mathrm{Id}) \xrightarrow{\quad\mathcal{T}^{\mathrm{St}}(\mathtt{ret})\quad} \mathcal{T}^{\mathrm{St}}(\mathrm{Cont}_{\mathbb{P}})}$$

We can do that whenever we have a monad transformer $\mathcal{T}$

$$\mathcal{T}^{\mathrm{Exc}} \qquad \mathcal{T}^{\mathrm{St}} \circ \mathcal{T}^{\mathrm{Exc}} \qquad \mathcal{T}^{\mathrm{Exc}} \circ \mathcal{T}^{\mathrm{St}} \qquad \ldots$$

Verifying non-deterministic programs



$$\theta^{\exists}(\{v_1, \ldots, v_n\}) \quad = \quad \lambda p.\, p\, v_1 \vee \ldots \vee p\, v_n$$
$$\theta^{\forall}(\{v_1, \ldots, v_n\}) \quad = \quad \lambda p.\, p\, v_1 \wedge \ldots \wedge p\, v_n$$

Angelic non-determinism $\theta^{\exists}$ vs **Demonic** non-determinism $\theta^{\forall}$

Input-Output in context

$$\mathcal{L} = (\mathcal{I} + \mathcal{O})^*$$

$$\mathrm{IO}\,X \xrightarrow{\theta^{\mathrm{S}}} \mathrm{W}^{\mathrm{Id}}\,X = (X \to \mathbb{P}) \to \mathbb{P}$$

$$\mathrm{IO}\,X \xrightarrow{\theta^{\mathrm{Fr}}} \mathrm{W}^{\mathrm{Fr}}\,X = (X \times \mathcal{L} \to \mathbb{P}) \to \mathbb{P}$$

$$\mathrm{IO}\,X \xrightarrow{\theta^{\mathrm{Hist}}} \mathrm{W}^{\mathrm{Hist}}\,X = (X \times \mathcal{L} \to \mathbb{P}) \to \mathcal{L} \to \mathbb{P}$$

$$\theta^{\mathrm{S}}(\texttt{write}\,o) = \lambda p.\, p\,()$$

$$\theta^{\mathrm{Fr}}(\texttt{write}\,o) = \lambda p.\, p\,\langle (),\ [o] \rangle$$

$$\theta^{\mathrm{Hist}}(\texttt{write}\,o) = \lambda p\, log.\, p\,\langle (),\ (o :: log) \rangle$$

A second bridge between computations and specifications

M	W
computational monad	**predicate transformer** monad
code	specification
$c : \mathrm{M}\ \mathbb{N}$	$w_c : \mathrm{W}\ \mathbb{N}$

A second bridge between computations and specifications

M
computational monad

W
predicate transformer monad

code

$c : \mathrm{M}\ \mathbb{N}$

specification

$w_c : \mathrm{W}\ \mathbb{N}$

Dijkstra monad

$c : \mathcal{D}^{\mathrm{M}}\ \mathbb{N}\ w_c$

A second bridge between computations and specifications

M
computational monad

W
predicate transformer monad

code

$c : \mathrm{M}\ \mathbb{N}$

specification

$w_c : \mathrm{W}\ \mathbb{N}$

Dijkstra monad

$c : \mathcal{D}^{\mathrm{M}}\ \mathbb{N}\ w_c$

$\texttt{ret}^{\mathcal{D}^{\mathrm{M}}} : (x : A) \to \mathcal{D}^{\mathrm{M}}\ A\ (\texttt{ret}^{\mathrm{W}}\ x)$

$$\frac{m : \mathcal{D}^{\mathrm{M}}\ A\ w_1 \qquad f : (x : A) \to \mathcal{D}^{\mathrm{M}}\ B\ w_2(x)}{\texttt{bind}^{\mathcal{D}^{\mathrm{M}}}\ m\ f : \mathcal{D}^{\mathrm{M}}\ B\ (\texttt{bind}^{\mathrm{W}}\ w_1\ w_2)}$$

Monadic program verification in F*

Dijkstra monads are heavily used in F*:

- Dependently-typed, programs and specifications in the same language

Monadic program verification in F*

Dijkstra monads are heavily used in F*:

- Dependently-typed, programs and specifications in the same language
- Multiple primitive effects (from **C**, **OCaml**)

$$\mathrm{Pure}\ A\ (w : W^{\mathrm{Id}} A) \qquad \mathrm{Div}\ A\ (w : W^{\mathrm{Id}} A) \qquad \mathrm{State}\ A\ (w : W^{\mathrm{St}} A)$$

$$\mathrm{Exc}\ A\ (w : W^{\mathrm{Exc}} A) \qquad \mathrm{All}\ A\ (w : W^{\mathrm{StExc}} A)$$

Monadic program verification in F⋆

Dijkstra monads are heavily used in F⋆:

- Dependently-typed, programs and specifications in the same language
- Multiple primitive effects (from **C**, **OCaml**)

$$\mathrm{Pure}\ A\ (w : W^{\mathrm{Id}} A) \qquad \mathrm{Div}\ A\ (w : W^{\mathrm{Id}} A) \qquad \mathrm{State}\ A\ (w : W^{\mathrm{St}} A)$$

$$\mathrm{Exc}\ A\ (w : W^{\mathrm{Exc}} A) \qquad \mathrm{All}\ A\ (w : W^{\mathrm{StExc}} A)$$

```
let rec fib (n:ℕ)
  : PURE ℕ (λ p → ∀ r. r ≥ n ∧ r > 0 ⟹ p r)
  =
  if n ≤ 1 then 1 else fib (n-1) + fib (n-2)
```

Monadic program verification in F⋆

Dijkstra monads are heavily used in F⋆:

- ▷ Dependently-typed, programs and specifications in the same language
- ▷ Multiple primitive effects (from **C**, **OCaml**)

$$\mathrm{Pure}\ A\,(w : W^{\mathrm{Id}} A) \qquad \mathrm{Div}\ A\,(w : W^{\mathrm{Id}} A) \qquad \mathrm{State}\ A\,(w : W^{\mathrm{St}} A)$$

$$\mathrm{Exc}\ A\,(w : W^{\mathrm{Exc}} A) \qquad \mathrm{All}\ A\,(w : W^{\mathrm{StExc}} A)$$

```
let rec fib (n:ℕ)
  : Pure ℕ (requires ⊤)
    (ensures (λ r → r ≥ n ∧ r > 0))
  =
  if n ≤ 1 then 1 else fib (n-1) + fib (n-2)
```

Dijkstra monads in F*

- Region and stack-based low-level model for extraction to **C**, used for implementing cryptographic primitives and protocols: **HACI***, **MiTLS**

Dijkstra monads in F*

- ▷ Region and stack-based low-level model for extraction to **C**, used for implementing cryptographic primitives and protocols: **HACL***, **MiTLS**

- ▷ Dijkstra Monads For Free (DM4Free) (Ahman et al., 2017)
  User-defined effects: monads in a DSL elaborate to
  - a specification monad W
  - a Dijkstra monad $\mathcal{D}$ indexed by W

Dijkstra monads in F*

- ▷ Region and stack-based low-level model for extraction to **C**, used for implementing cryptographic primitives and protocols: **HACL***, **MiTLS**

- ▷ Dijkstra Monads For Free (DM4Free) (Ahman et al., 2017)
User-defined effects: monads in a DSL elaborate to
  - a specification monad W
  - a Dijkstra monad $\mathcal{D}$ indexed by W

$\rightsquigarrow$ State, exceptions, State+Exceptions,...
$\rightsquigarrow$ No Input-output, non-determinism, probabilities...

From effect observation to Dijkstra monad

$$\mathrm{M} \xrightarrow{\quad\theta\quad} \mathrm{W}$$

From effect observation to Dijkstra monad

$$\mathrm{M} \xrightarrow{\quad\theta\quad} \mathrm{W}$$

$$\mathcal{D}^{\mathrm{M}} A\,(w : \mathrm{W}\,A) \quad = \quad \{m : \mathrm{M}\,A \mid w \leq^{\mathrm{W}} \theta(m)\}$$

A Dijkstra monad for demonic non-determinism

Recall that there is an effect observation

$$\theta^{\forall} \quad : \quad \mathrm{NDet}\, A \quad \longrightarrow \quad \mathrm{W}^{\mathrm{Id}}\, A = (A \to \mathbb{P}) \to \mathbb{P}$$

$$\theta^{\forall}(m) = \lambda p.\, \forall v \in m, p\, v$$

A Dijkstra monad for demonic non-determinism

Recall that there is an effect observation

$$\theta^{\forall} \quad : \qquad \mathrm{NDet}\, A \quad \longrightarrow \quad \mathrm{W}^{\mathrm{Id}}\, A = (A \to \mathbb{P}) \to \mathbb{P}$$

$$\theta^{\forall}(m) = \lambda p.\, \forall v \in m, p\, v$$

We obtain a Dijkstra monad

$$\begin{aligned} \mathrm{ND} &: (A : \mathrm{Type}) \to (w : \mathrm{W}^{\mathrm{Id}}\, A) \to \mathrm{Type} \\ \texttt{choose} &: (u : \mathbb{1}) \to \mathrm{ND}\ \mathbb{B}\ (\lambda p.\, p\ \texttt{true} \wedge p\ \texttt{false}) \\ \texttt{fail} &: (u : \mathbb{1}) \to \mathrm{ND}\ \mathbb{B}\ (\lambda p.\, \top) \end{aligned}$$

A Dijkstra monad for demonic non-determinism

Recall that there is an effect observation

$$\theta^{\forall} \quad : \quad \mathrm{NDet}\, A \quad \longrightarrow \quad \mathrm{W}^{\mathrm{Id}}\, A = (A \to \mathbb{P}) \to \mathbb{P}$$

$$\theta^{\forall}(m) = \lambda p.\, \forall v \in m, p\, v$$

We obtain a Dijkstra monad

$$\mathrm{ND} : (A : \mathrm{Type}) \to (w : \mathrm{W}^{\mathrm{Id}}\, A) \to \mathrm{Type}$$
$$\texttt{choose} : (u : \mathbb{1}) \to \mathrm{ND}\ \mathbb{B}\ (\lambda p.\, p\ \texttt{true} \wedge p\ \texttt{false})$$
$$\texttt{fail} : (u : \mathbb{1}) \to \mathrm{ND}\ \mathbb{B}\ (\lambda p.\, \top)$$

```
let rec pick #a (l : list a)
  : ND a (λ p → ∀ x. List.memP x l ⟹ p x)
  =
  match l with
  | [] → fail ()
  | x∷xs → if choose true false then x else pick xs
```

Computing pythagorean triples

```
let guard (b:bool)
  : ND unit (λ p → b ⟹ p ())
  =
  if b then () else fail ()

let pyths ()
  : ND (ℤ & ℤ & ℤ)
       (λ p → ∀ x y z. x×x + y×y == z×z ⟹ p (x,y,z))
  =
  let l = [1;2;3;4;5;6;7;8;9;10] in
  let x = pick l in
  let y = pick l in
  let z = pick l in
  guard (x×x + y×y = z×z);
  (x,y,z)
```

Synthesis

- ▷ Start with a **computational monad** M
- ▷ Select a **specification monad** W
- ▷ Define an **effect observation** $\theta : \mathrm{M} \to \mathrm{W}$
- ▷ Obtain a **Dijkstra monad** $\mathcal{D}^{\mathrm{M}}$ indexed by W
- $\rightsquigarrow$ Convenient way to verify code in M

Synthesis

- ▷ Start with a **computational monad** M
- ▷ Select a **specification monad** W
- ▷ Define an **effect observation** $\theta : \mathrm{M} \to \mathrm{W}$
- ▷ Obtain a **Dijkstra monad** $\mathcal{D}^{\mathrm{M}}$ indexed by W
- $\rightsquigarrow$ Convenient way to verify code in M

Further questions:

- ▷ How far is $\mathcal{D}^{\mathrm{M}}$ from θ ?
- ▷ Can we explain the previous awkward restrictions in DM4Free ?

From Dijkstra monad to effect observation

Given a Dijkstra monad $\mathcal{D}$ over W

From Dijkstra monad to effect observation

Given a Dijkstra monad $\mathcal{D}$ over W

$$\mathrm{M}\,A \quad = \quad (w : \mathrm{W}\;A) \times \mathcal{D}\,A\,w$$

$$\mathrm{M} \xrightarrow{\quad \pi_1 \quad} \mathrm{W}$$

From Dijkstra monad to effect observation

Given a Dijkstra monad $\mathcal{D}$ over W

$$\mathrm{M}\,A \quad = \quad (w : \mathrm{W}\ A) \times \mathcal{D}\,A\,w$$

$$\mathrm{M} \xrightarrow{\quad \pi_1 \quad} \mathrm{W}$$

Extends to a (categorical) equivalence

Dijkstra monads	$\cong$	Effect observations
$(\mathrm{W}, \mathcal{D})$		$\theta : \mathrm{M} \to \mathrm{W}$

A DSL for monad transformers

$$C ::= \mathbb{M}A \mid C_1 \times C_2 \mid (x : A) \to C \mid C_1 \to C_2 \quad A \in Type_{\mathcal{L}}$$

$$t ::= \texttt{ret} \mid \texttt{bind} \mid \langle t_1,\ t_2 \rangle \mid \pi_i\ t \mid x \mid \lambda x.\, t \mid t_1\ t_2 \mid \lambda^{\diamond} x.\ t \mid t\ u$$

A DSL for monad transformers

$$C ::= \mathbb{M}A \mid C_1 \times C_2 \mid (x : A) \to C \mid C_1 \to C_2 \quad A \in Type_{\mathcal{L}}$$

$$t ::= \mathtt{ret} \mid \mathtt{bind} \mid \langle t_1,\ t_2 \rangle \mid \pi_i\ t \mid x \mid \lambda x.\, t \mid t_1\ t_2 \mid \lambda^{\diamond} x.\ t \mid t\ u$$

Observation 1: from monads $C[X]$ and M we can derive a monad $\mathcal{T}^C(\mathrm{M})$
Just substitute $\mathbb{M}$ by M in $C[X]$

A DSL for monad transformers

$$C ::= \mathbb{M}A \mid C_1 \times C_2 \mid (x : A) \to C \mid C_1 \to C_2 \quad A \in Type_{\mathcal{L}}$$

$$t ::= \texttt{ret} \mid \texttt{bind} \mid \langle t_1,\ t_2 \rangle \mid \pi_i\ t \mid x \mid \lambda x.\, t \mid t_1\ t_2 \mid \lambda^{\diamond} x.\ t \mid t\ u$$

Observation 1: from monads $C[X]$ and M we can derive a monad $\mathcal{T}^C(\mathrm{M})$
Just substitute $\mathbb{M}$ by M in $C[X]$

Observation 2: $\mathcal{T}^C(\mathrm{M}) = C[\mathrm{M}/\mathbb{M}]$ comes with an M-algebra structure α

A DSL for monad transformers

$$C ::= \mathbb{M}A \mid C_1 \times C_2 \mid (x : A) \rightarrow C \mid C_1 \rightarrow C_2 \quad A \in Type_{\mathcal{L}}$$

$$t ::= \texttt{ret} \mid \texttt{bind} \mid \langle t_1,\ t_2 \rangle \mid \pi_i\ t \mid x \mid \lambda x.\, t \mid t_1\ t_2 \mid \lambda^{\diamond} x.\ t \mid t\ u$$

Observation 1: from monads $C[X]$ and M we can derive a monad $\mathcal{T}^{C}(\mathrm{M})$
Just substitute $\mathbb{M}$ by M in $C[X]$

Observation 2: $\mathcal{T}^{C}(\mathrm{M}) = C[\mathrm{M}/\mathbb{M}]$ comes with an M-algebra structure α

$$\texttt{lift} \quad : \quad \mathrm{M} \xrightarrow{\mathrm{M}(\texttt{ret}^{\mathcal{T}^{C}(\mathrm{M})})} \mathrm{M}(\mathcal{T}^{C}(\mathrm{M})) \xrightarrow{\alpha} \mathcal{T}^{C}(\mathrm{M})$$

Reinterpreting DM4Free

Under a few conditions on C: $\mathcal{T}^C$ is actually a **monad transformer**

$\rightsquigarrow$ uses a logical relation to extend $\mathcal{T}^C$ on monad morphisms

Reinterpreting DM4Free

Under a few conditions on C: $\mathcal{T}^C$ is actually a **monad transformer**

$\rightsquigarrow$ uses a logical relation to extend $\mathcal{T}^C$ on monad morphisms

Effect observation for DM4Free:

$$\frac{\mathrm{Id} \xrightarrow{\quad \texttt{ret} \quad} \mathrm{Cont}_{\mathbb{P}}}{\theta : \mathcal{T}(\mathrm{Id}) \xrightarrow{\quad \mathcal{T}(\texttt{ret}) \quad} \mathcal{T}(\mathrm{Cont}_{\mathbb{P}})}$$

Reinterpreting DM4Free

Under a few conditions on C: $\mathcal{T}^C$ is actually a **monad transformer**

$\rightsquigarrow$ uses a logical relation to extend $\mathcal{T}^C$ on monad morphisms

Effect observation for DM4Free:

$$\frac{\mathrm{Id} \xrightarrow{\quad \texttt{ret} \quad} \mathrm{Cont}_{\mathbb{P}}}{\theta : \mathcal{T}(\mathrm{Id}) \xrightarrow{\quad \mathcal{T}(\texttt{ret}) \quad} \mathcal{T}(\mathrm{Cont}_{\mathbb{P}})}$$

For a general monad C: $\mathcal{T}^C$ only preserves **monadic relations** not monad morphisms

Further directions

- Monadic relations as effect observations

Further directions

- Monadic relations as effect observations
- Partial implementations in Coq, $F^\star$

Further directions

- Monadic relations as effect observations
- Partial implementations in Coq, $F^\star$
- Algebraic effects and handlers

Further directions

- Monadic relations as effect observations
- Partial implementations in Coq, $F^\star$
- Algebraic effects and handlers
- Relational reasoning

Further directions

- Monadic relations as effect observations
- Partial implementations in Coq, $F^\star$
- Algebraic effects and handlers
- Relational reasoning

Thank you !

Bibliography

D. Ahman, C. Hriţcu, K. Maillard, G. Martínez, G. Plotkin, J. Protzenko, A. Rastogi, and N. Swamy. Dijkstra monads for free. **POPL**. 2017.

S. Katsumata. Parametric effect monads and semantics of effect systems. **POPL**. 2014.

E. Moggi. Computational lambda-calculus and monads. **LICS**. 1989.

G. D. Plotkin and J. Power. Algebraic operations and generic effects. **Applied Categorical Structures**, 11(1):69–94, 2003.