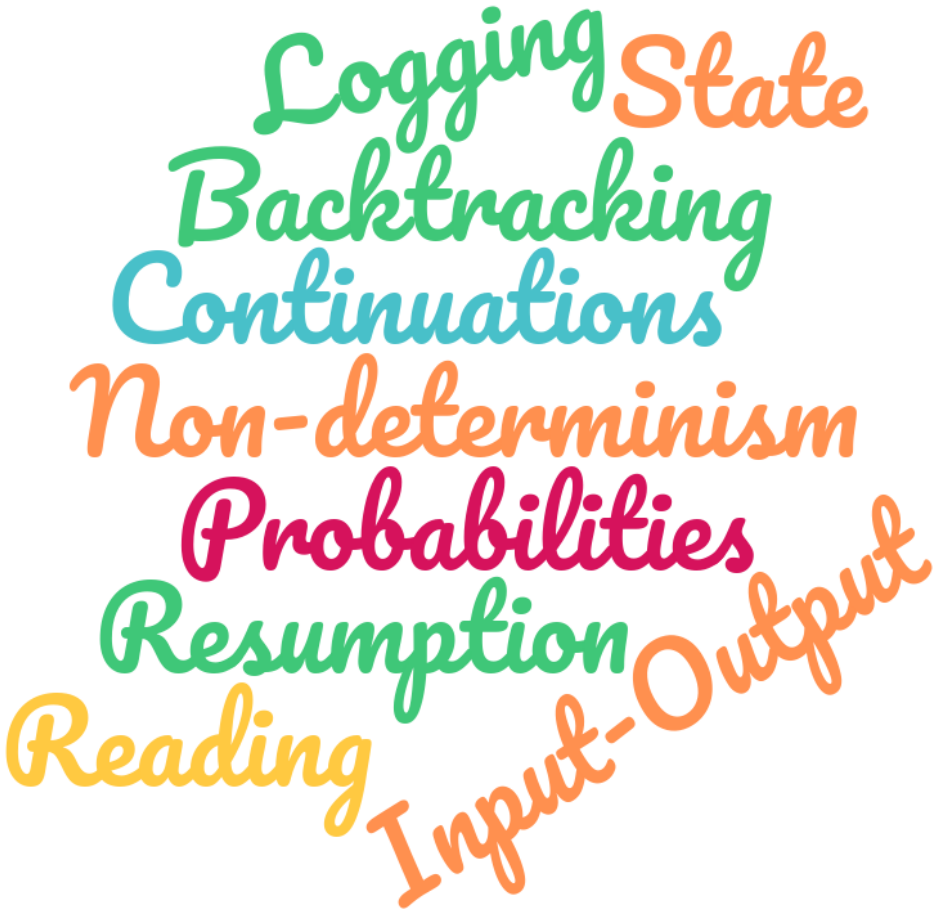
Designing Dijkstra Monads

arXiv: Dijkstra Monads For All

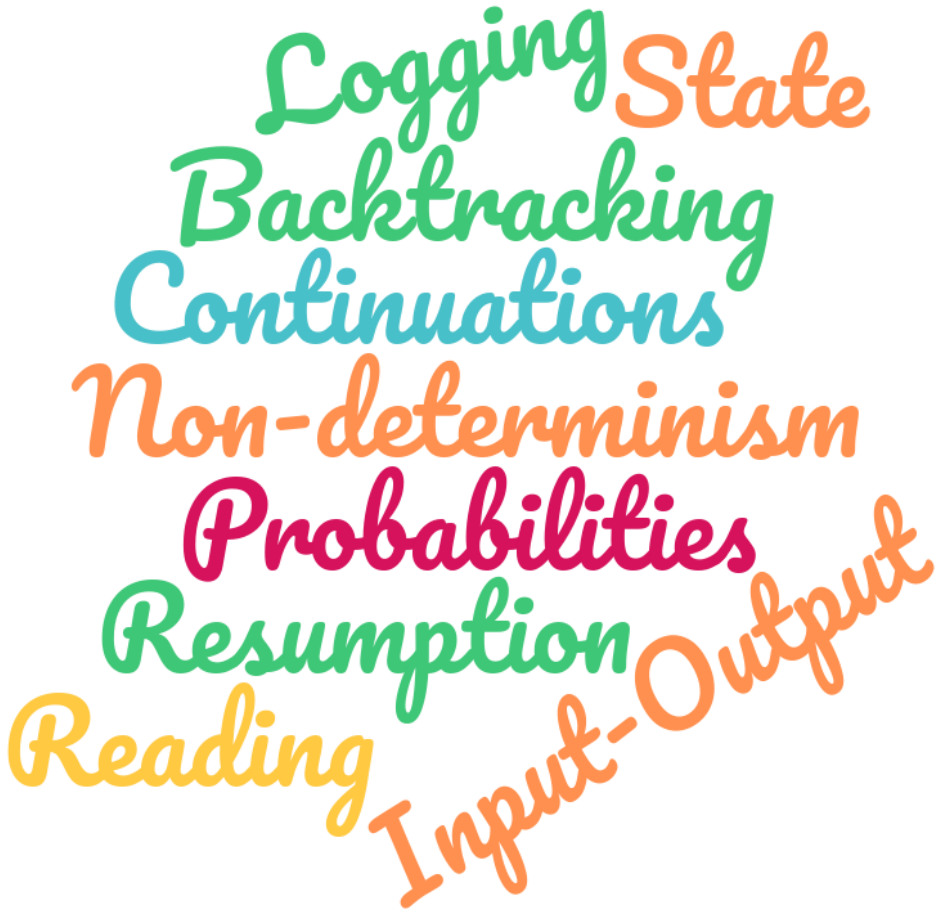
Kenji Maillard
jww D. Ahman, B. Atkey, G.Martinez,
C.Hriţcu, E.Tanter, E. Rivas

June 12, 2019

Programming with side effects



Programming with side effects



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

Monads: a nifty algebraic tool

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

Monads: a nifty algebraic tool

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

A succint syntactic presentation: a type constructor $\mathcal{M}$

$\texttt{ret}^{\mathcal{M}} : X \to \mathcal{M}X \qquad \texttt{bind}^{\mathcal{M}} : \mathcal{M}X \to (X \to \mathcal{M}Y) \to \mathcal{M}Y$

$+$ 3 equations (monad laws)

Monads: a nifty algebraic tool

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

A succint syntactic presentation: a type constructor $\mathcal{M}$

$$\texttt{ret}^{\mathcal{M}} : X \to \mathcal{M}X \qquad \texttt{bind}^{\mathcal{M}} : \mathcal{M}X \to (X \to \mathcal{M}Y) \to \mathcal{M}Y$$

$+$ 3 equations (monad laws)

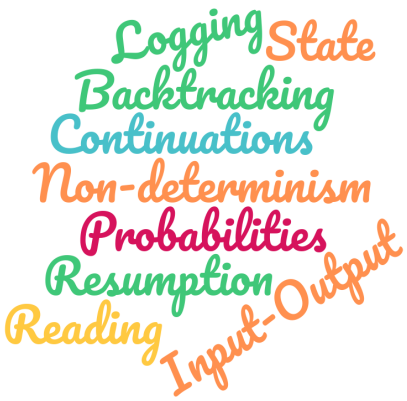
State	Exceptions	Non-determinism
$\mathrm{St}\,X = \mathcal{S} \to X \times \mathcal{S}$	$\mathrm{Exc}\,X = X + \mathcal{E}$	$\mathrm{NDet}\,X = \mathcal{P}_{fin}(X)$

Input-Output

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

Starting point

Pick $\mathcal{M}$ a computational monad $\in$

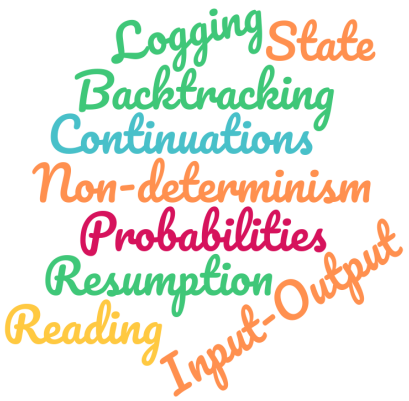


Consider the following monadic program

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

Starting point

Pick $\mathcal{M}$ a computational monad $\in$



Consider the following monadic program

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

How can we specify and verify such monadic programs ?

Hoare logic Primer

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

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

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

Hoare logic Primer

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

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

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

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

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

Wait... That's a monad !

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

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

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

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

Wait... That's a monad !

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

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

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

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

Also monads:

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

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

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

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

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

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

Specification monads from monad transformers

Examples of **specification** 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:	$\mathrm{W}^{\mathrm{Exc}} : (X + \mathcal{E} \to \mathbb{P}) \to \mathbb{P}$

Specification monads from monad transformers

Examples of **specification** 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: $\mathrm{W}^{\mathrm{Exc}} : (X + \mathcal{E} \to \mathbb{P}) \to \mathbb{P}$

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

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

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

Specification monads from monad transformers

Examples of **specification** 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: $\mathrm{W}^{\mathrm{Exc}} : (X + \mathcal{E} \to \mathbb{P}) \to \mathbb{P}$

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

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

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

Monad transformer $\mathcal{T}$: $\begin{cases} \text{map a monad } \mathcal{M} \text{ to a monad } \mathcal{T}\mathcal{M} \\ \texttt{lift}^{\mathcal{T}} : \mathcal{M} \to \mathcal{T}\mathcal{M} \end{cases}$

Bridging the gap

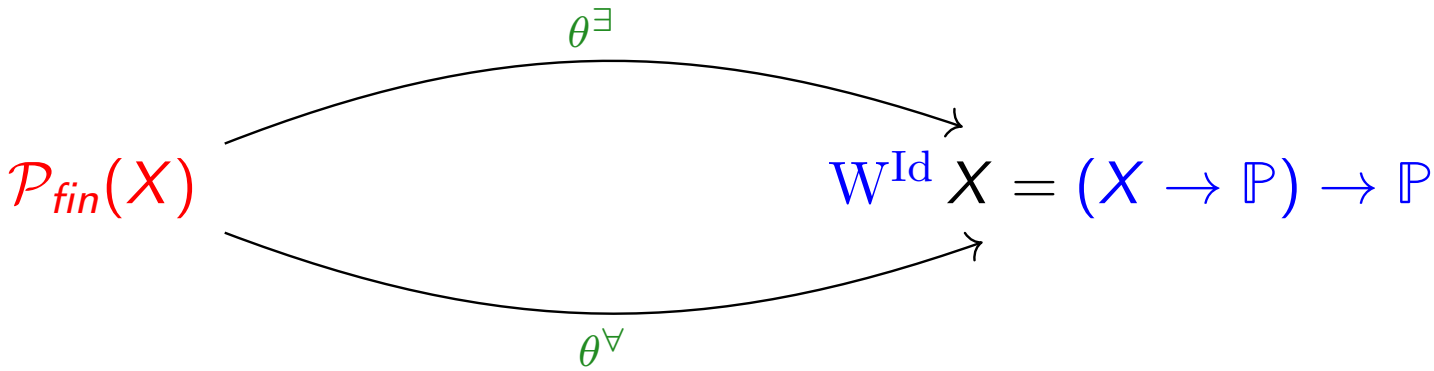
An effect observation θ (Katsumata, 2014)

is a **monad morphism**, that is

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

Provides an **interpretation** of programs as predicate transformers.

Specifying 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 $\theta^{\forall}$

A second bridge between computations and specifications

$\mathcal{M}$

computational monad

W

predicate transformer monad

code

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

specification

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

A second bridge between computations and specifications

$\mathcal{M}$
computational monad

W
predicate transformer monad

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

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

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

A second bridge between computations and specifications

$\mathcal{M}$
computational monad

W
predicate transformer monad

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

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

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

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

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

Program verification with Dijkstra monads (F*)

$$\mathrm{PURE}\ A\ w \qquad w : (A \to \mathbb{P}) \to \mathbb{P}$$

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

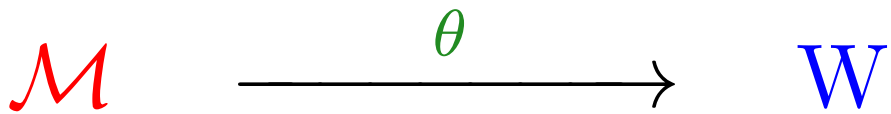
Program verification with Dijkstra monads (F*)

$$\mathrm{PURE}\ A\ w \qquad w : (A \to \mathbb{P}) \to \mathbb{P}$$

$$\mathrm{Pure}\ A\ \mathit{pre}\ \mathit{post} \qquad \mathit{pre} : \mathbb{P} \quad \mathit{post} : A \to \mathbb{P}$$

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

From effect observation to Dijkstra monad



From effect observation to Dijkstra monad

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

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

From effect observation to Dijkstra monad

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

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

Extends to a (categorical) equivalence

$$\begin{array}{ccc} \text{Dijkstra monads} & \cong & \text{Effect observations} \\ (\mathrm{W}, \mathcal{D}) & & \theta : \mathcal{M} \to \mathrm{W} \end{array}$$

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

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

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

We obtain a Dijkstra monad

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

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** $\mathcal{M}$
- ▷ Select a **specification monad** W
- ▷ Define an **effect observation** $\theta : \mathcal{M} \to \mathrm{W}$
- ▷ Obtain a **Dijkstra monad** $\mathcal{D}^{\mathcal{M}}$ indexed by W

$\rightsquigarrow$ Convenient way to verify code in $\mathcal{M}$

Synthesis

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

$\rightsquigarrow$ Convenient way to verify code in $\mathcal{M}$

Moreover combines nicely with **monad transformers**, e.g.

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

Further directions

- Implementations: native in F*, quite similar to Hoare Type Theory in Coq (experimental)
- Algebraic effects and handlers (catch for exceptions, general recursion using free monads à la McBride (2015))
- Relational reasoning (RHTT, RF*)

Further directions

- ▷ Implementations: native in F*, quite similar to Hoare Type Theory in Coq (experimental)
- ▷ Algebraic effects and handlers (catch for exceptions, general recursion using free monads à la McBride (2015))
- ▷ Relational reasoning (RHTT, RF*)

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.

C. McBride. Turing-completeness totally free. **MPC**. 2015.

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.

General recursion

Fix domain Dom, $\prec \subseteq \mathrm{Dom} \times \mathrm{Dom}$ well-founded, codomain Cod.

GenRec free monad on one operation

$$\texttt{call} \quad : \quad \mathrm{Dom} \quad \rightsquigarrow \quad \mathrm{Cod}$$

Given a recursion invariant $(\mathit{pre}_d, \mathit{post}_d)_d : (d : \mathrm{Dom}) \to \mathbb{P} \times (\mathrm{Cod} \to \mathbb{P})$ and $d_0 : \mathrm{Dom}$, define

$$\theta^{\mathsf{GenRec}} \quad : \quad \mathsf{GenRec} \quad \longrightarrow \quad \mathrm{Cont}_{\mathbb{P}}$$

$$\begin{array}{lcl} \theta^{\mathsf{GenRec}}(\texttt{ret}\, v) & = & \lambda p.\, p\, v \\ \theta^{\mathsf{GenRec}}(\texttt{call}\, d\, k) & = & \lambda p.\, \mathit{pre}_d \wedge d \prec d_0 \wedge \\ & & \qquad \forall c : \mathrm{Cod}, \mathit{post}_d\, c \Rightarrow \theta^{\mathsf{GenRec}}(k\, c)\, p \end{array}$$

General recursion in Dijkstra monad form

From θ^{GenRec}, derive a dijkstra monad $\mathcal{D}^{\mathsf{GenRec}}_{d_0}$ with

▷ An operation

$$\texttt{call} \quad : \quad (d : \mathrm{Dom}) \quad \longrightarrow \quad \mathcal{D}^{\mathsf{GenRec}}_{d_0}\, \mathrm{Cod}\ (\mathit{pre}_d \wedge d \prec d_0)\ \mathit{post}_d$$

▷ A handler

$$\begin{aligned} \texttt{fix} \quad : \quad & ((d : \mathrm{Dom}) \to \mathcal{D}^{\mathsf{GenRec}}_{d}\, \mathrm{Cod}\ \mathit{pre}_d\ \mathit{post}_d) \to \\ & (d : \mathrm{Dom}) \to \mathrm{Pure}\ \mathrm{Cod}\ \mathit{pre}_d\ \mathit{post}_d \end{aligned}$$

Back to fibonacci

$$\texttt{val}\ \ \mathit{fib}_0 : (n : \mathbb{N}) \to \mathcal{D}_n^{\mathsf{GenRec}}\,\mathbb{N}\ \top\ (\lambda r.\, r \geq 1 \wedge r \geq n))$$

$$\begin{array}{l}
\texttt{let}\ \ \mathit{fib}_0\ n = \\
\qquad \texttt{if}\ \ n \leq 1\ \texttt{then}\ 1 \\
\qquad \texttt{else} \\
\qquad\quad \texttt{let}\ \ r_1 = \texttt{call}\,(n-1)\ \texttt{in} \\
\qquad\quad \texttt{let}\ \ r_2 = \texttt{call}\,(n-2)\ \texttt{in}\ \ r_1 + r_2
\end{array}$$

$$\texttt{val}\ \ \mathit{fib} : (n : \mathbb{N}) \to \mathrm{Pure}\,\mathbb{N}\ \top\ (\lambda r.\, r \geq 1 \wedge r \geq n))$$

$$\texttt{let}\ \ \mathit{fib} = \texttt{fix}\ \mathit{fib}_0$$