Principles of Program Verification for Arbitrary Monadic Effects

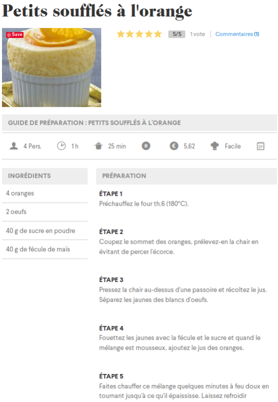
Kenji Maillard
advised by Cătălin Hrițcu

ENS & Inria Paris, team Prosecco

PhD defense
Monday the 25th of November, 2019

The "Programs as Recipes" paradigm

- ▷ A program is a sequence of instructions



Petits soufflés à l'orange

5/5 1 vote | Commentaires (1)

GUIDE DE PRÉPARATION : PETITS SOUFFLÉS À L'ORANGE

4 Pers. 1 h 25 min 5,62 Facile

INGRÉDIENTS

- 4 oranges
- 2 oeufs
- 40 g de sucre en poudre
- 40 g de fécule de maïs

PRÉPARATION

ÉTAPE 1
Préchauffez le four th.6 (180°C).

ÉTAPE 2
Coupez le sommet des oranges, prélevez-en la chair en évitant de percer l'écorce.

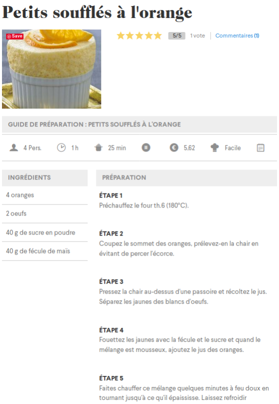
ÉTAPE 3
Pressez la chair au-dessus d'une passoire et récoltez le jus. Séparez les jaunes des blancs d'oeufs.

ÉTAPE 4
Fouettez les jaunes avec la fécule et le sucre et quand le mélange est mousseux, ajoutez le jus des oranges.

ÉTAPE 5
Faites chauffer ce mélange quelques minutes à feu doux en tournant jusqu'à ce qu'il épaississe. Laissez refroidir

The "Programs as Recipes" paradigm

- A program is a sequence of instructions
- A computer evaluates a program in a similar way a cook realizes a recipe, by executing each steps at a time



Petits soufflés à l'orange

Save

5/5 1 vote | Commentaires (1)

GUIDE DE PRÉPARATION : PETITS SOUFFLÉS À L'ORANGE

4 Pers. 1 h 25 min 5,62 Facile

INGRÉDIENTS
4 oranges
2 oeufs
40 g de sucre en poudre
40 g de fécule de maïs

PRÉPARATION

ÉTAPE 1
Préchauffez le four th.6 (180°C).

ÉTAPE 2
Coupez le sommet des oranges, prélevez-en la chair en évitant de percer l'écorce.

ÉTAPE 3
Pressez la chair au-dessus d'une passoire et récoltez le jus. Séparez les jaunes des blancs d'oeufs.

ÉTAPE 4
Fouettez les jaunes avec la fécule et le sucre et quand le mélange est mousseux, ajoutez le jus des oranges.

ÉTAPE 5
Faites chauffer ce mélange quelques minutes à feu doux en tournant jusqu'à ce qu'il épaississe. Laissez refroidir

The "Programs as Recipes" paradigm

- A program is a sequence of instructions
- A computer evaluates a program in a similar way a cook realizes a recipe, by executing each steps at a time

Soufflé au fromage. — Faire fondre dans une casserole 50 gr. de beurre, mélanger 40 gr. de farine et mouiller avec 1/4 de lire de lait. Saler, poivrer et faire bouillir en remuant avec le fouet. Au premier bouillon, on obtient comme une Béchamelle très épaisse. Retirer du feu et ajouter une noix de beurre, une pointe de muscade râpée et 4 jaunes d'œufs. Ajouter encore 3 blancs en neige, en même temps que 100 gr. de gruyère râpé. Le mélange doit se faire assez vivement avec la cuillère. Verser cet appareil dans une casserole à soufflé, beurrée et poudrée de fromage. Cuire à four moyen pendant 20 à 22 minutes et servir aussitôt dans la casserole où il a cuit.

On peut aussi cuire ce soufflé dans de petites caissettes en porcelaine. Dans ce cas, 8 minutes de cuisson sont suffisantes. Le four doit être plus chaud dessous que dessus.

140

Petits soufflés à l'orange

5/5 1 vote | Commentaires (1)

GUIDE DE PRÉPARATION : PETITS SOUFFLÉS À L'ORANGE

4 Pers. 1 h 25 min 5,62 Facile

INGRÉDIENTS

- 4 oranges
- 2 oeufs
- 40 g de sucre en poudre
- 40 g de fécule de maïs

PRÉPARATION

ÉTAPE 1
Préchauffez le four th.6 (180°C).

ÉTAPE 2
Coupez le sommet des oranges, prélevez-en la chair en évitant de percer l'écorce.

ÉTAPE 3
Pressez la chair au-dessus d'une passoire et récoltez le jus. Séparez les jaunes des blancs d'oeufs.

ÉTAPE 4
Fouettez les jaunes avec la fécule et le sucre et quand le mélange est mousseux, ajoutez le jus des oranges.

ÉTAPE 5
Faites chauffer ce mélange quelques minutes à feu doux en tournant jusqu'à ce qu'il épaississe. Laissez refroidir

How to make sure that the program achieve some properties ?

Program verification: what's my program doing?

Formal specification

A logical formula describing the action of my program

Given a natural number n, compute the n^{th} Fibonnacci number

Formal verification

Proving that a program validates a given specification.

Program verification: what's my program doing?

Formal specification

A logical formula describing the action of my program

Given a natural number n, compute the n^{th} Fibonnacci number

Formal verification

Proving that a program validates a given specification.

```
let rec fibonacci n =
  if n = 0 || n = 1 then 1
  else fibonacci (n-1) + fibonacci (n-2)
```

Program verification: what's my program doing?

Formal specification

A logical formula describing the action of my program

Given a natural number n, compute the n^{th} Fibonnacci number

Formal verification

Proving that a program validates a given specification.

```
val fibonacci : ℕ → ℕ
let rec fibonacci n =
  if n = 0 || n = 1 then 1
  else fibonacci (n-1) + fibonacci (n-2)
```

Program verification: what's my program doing?

Formal specification

A logical formula describing the action of my program

Given a natural number n, compute the n^{th} Fibonnacci number

Formal verification

Proving that a program validates a given specification.

```
val fibonacci : n:ℤ → Pure ℤ
                          (requires _)
                          (ensures _)
let rec fibonacci n =
  if n = 0 || n = 1 then 1
  else fibonacci (n-1) + fibonacci (n-2)
```

Program verification: what's my program doing?

Formal specification

A logical formula describing the action of my program

Given a natural number n, compute the n^{th} Fibonnacci number

Formal verification

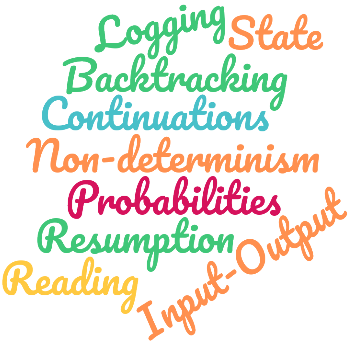
Proving that a program validates a given specification.

```
val fibonacci : n:ℤ → Pure ℤ
                          (requires (n ≥ 0))
                          (ensures (λ r → r ≥ max 1 n))
let rec fibonacci n =
  if n = 0 ‖ n = 1 then 1
  else fibonacci (n-1) + fibonacci (n-2)
```

Programs & Side-effects

Beyond computing **pure** mathematical functions, programs can

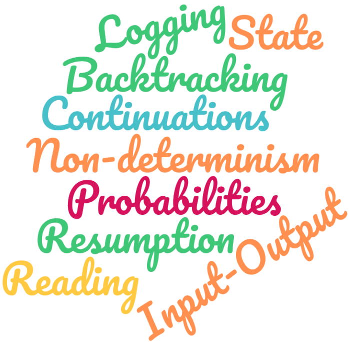
- Read a password from the keyboard
- Send meta-data to a remote server
- Store temporary or persistent data
- Raise signals or exceptions,
  break control flow
- Flip a coin randomly
- Pick an element of a list
  non-deterministically



Programs & Side-effects

Beyond computing **pure** mathematical functions, programs can

- Read a password from the keyboard
- Send meta-data to a remote server
- Store temporary or persistent data
- Raise signals or exceptions,
  break control flow
- Flip a coin randomly
- Pick an element of a list
  non-deterministically



$\rightsquigarrow$ Use monads to represent side-effects uniformly!

Monadic effects [Moggi '89]

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

A monad $\mathcal{M} : \mathrm{Type} \to \mathrm{Type}$ comes with operations and laws:

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

Monadic effects [Moggi '89]

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

A monad $\mathcal{M} : \mathrm{Type} \to \mathrm{Type}$ comes with operations and laws:

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

Examples of computational monads:

- ▷ State state passing computations $\mathrm{St}(X) = \mathcal{S} \to X \times \mathcal{S}$
- ▷ Exceptions $\mathrm{Exc}(X) = X + \mathcal{E}$
- ▷ Non-determinism finite sets of results $\mathrm{NDet}(X) = \mathcal{P}_{fin}(X)$
- ▷ Continuations $\mathrm{Cont}_{\mathfrak{A}}(X) = (X \to \mathfrak{A}) \to \mathfrak{A}$

Also interactive IO, probabilities...

Verifying monadic programs?

- Many tools for program verification

  Hoare logics   Separation logics   . . .

Verifying monadic programs?

- Many tools for program verification

  Hoare logics   Separation logics   ...

- Often effect-specific

  State + Exceptions   State + Probabilities

Verifying monadic programs?

- Many tools for program verification

  Hoare logics   Separation logics   . . .

- Often effect-specific

  State + Exceptions   State + Probabilities

- Distil the common ideas underlying most of these tools

Verifying monadic programs?

- Many tools for program verification

  Hoare logics   Separation logics   ...

- Often effect-specific

  State + Exceptions   State + Probabilities

- Distil the common ideas underlying most of these tools

Unifying principles for reasoning with arbitrary monadic effects

Roadmap

Weakest preconditions as specifications

Hoare logic ['69]: specifying (stateful) code with predicates

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

Weakest preconditions as specifications

Hoare logic ['69]: specifying (stateful) code with predicates

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

Dijkstra's insight ['75]: a **weakest** precondition $\mathtt{wp}[c]$ can be computed compositionally from a program and a postcondition

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

Weakest preconditions as specifications

Hoare logic ['69]: specifying (stateful) code with predicates

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

Dijkstra's insight ['75]: a **weakest** precondition $\mathtt{wp}[c]$ can be computed compositionally from a program and a postcondition

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

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

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

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

Weakest preconditions as monads!

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

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

Weakest preconditions as monads!

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

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

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

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

Weakest preconditions as monads!

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

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

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

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

Also monads:

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

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

Specification monads from monad transformers

Examples of predicate transformers 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 predicate transformers monads:

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

$$
\begin{aligned}
\mathrm{W}^{\mathrm{Id}} &= \mathcal{T}^{\mathrm{Id}}(\mathrm{Cont}_{\mathbb{P}}) & \mathcal{T}^{\mathrm{Id}}(\mathcal{M}) &= \mathcal{M} \\
\mathrm{W}^{\mathrm{St}} &= \mathcal{T}^{\mathrm{St}}(\mathrm{Cont}_{\mathbb{P}}) & \mathcal{T}^{\mathrm{St}}(\mathcal{M}) &= \mathcal{S} \to \mathcal{M}(- \times \mathcal{S}) \\
\mathrm{W}^{\mathrm{Exc}} &= \mathcal{T}^{\mathrm{Exc}}(\mathrm{Cont}_{\mathbb{P}}) & \mathcal{T}^{\mathrm{Exc}}(\mathcal{M}) &= \mathcal{M}(- + \mathcal{E})
\end{aligned}
$$

Specification monads from monad transformers

Examples of predicate transformers 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 $\mathcal{T}$: $\left\{ \begin{array}{l} \text{map a monad } \mathcal{M} \text{ to a monad } \mathcal{T}\mathcal{M} \\ \texttt{lift}^{\mathcal{T}} : \mathcal{M} \to \mathcal{T}\mathcal{M} \end{array} \right.$

Specification monads

Beside weakest precondition, other **specification 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

Beside weakest precondition, other **specification monads**!

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

What's in a specification monad W?

- specifications are partially ordered, e.g.

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

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

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

Bridging the specification gap

$\mathcal{M}$ $\mathbb{W}$

Bridging the specification gap

$$\mathcal{M} \xrightarrow{\theta} \mathbb{W}$$

An effect observation θ [Katsumata'14]

Bridging the specification gap

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

An effect observation θ [Katsumata'14] is a *monad morphism* from a computational monad $\mathcal{M}$ to a specification monad W, i.e.

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

$\rightsquigarrow$ An **interpretation/semantics** of programs as specifications.

Interpreting non-deterministic programs

$$\mathcal{P}_{fin}\,X \xrightarrow{\theta^{\forall}} \mathrm{W}^{\mathrm{Id}}\,X = (X \to \mathbb{P}) \to \mathbb{P}$$

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

Demonic non-determinism $\theta^{\forall}$

Interpreting non-deterministic programs

$$
\begin{tikzcd}
\mathcal{P}_{fin}\, X \arrow[rr, bend left, "\theta^{\forall}"] \arrow[rr, bend right, "\theta^{\exists}"'] & & \mathrm{W}^{\mathrm{Id}}\, X = (X \to \mathbb{P}) \to \mathbb{P}
\end{tikzcd}
$$

$$
\begin{array}{rcl}
\theta^{\forall}(\{v_1, \ldots, v_n\}) & = & \lambda post.\, post\ v_1 \wedge \ldots \wedge post\ v_n \\
\theta^{\exists}(\{v_1, \ldots, v_n\}) & = & \lambda post.\, post\ v_1 \vee \ldots \vee post\ v_n
\end{array}
$$

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

Input-Output and the History of events

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

$\mathcal{E}^\star$: list of IO events

Global history specifications θ^{Glob}

Input-Output and the History of events

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

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

$\mathcal{E}^\star$: list of IO events

Global history specifications θ^{Glob} vs **Local history** specifications θ^{Loc}

Contributions

- Specification monads:
  specifications on the same footing as programs
- Effect observations provide great flexibility
  - In the choice of the semantics
  - In the complexity of the specifications
  - Instances for state, exceptions, IO, non-determinism...
- Standard techniques for monads apply:
  Monad Transformers

Presented at ICFP'19 in *Dijkstra Monads for All*

Verification: Dijkstra Monads

What is a Dijkstra monad?

$\mathcal{M}$
computational monad

code

$c : \mathcal{M}\ A$

W
specification monad

specification

$w_c : \mathrm{W}\ A$

What is a Dijkstra monad?

$\mathcal{M}$
computational monad

W
specification monad

code

$c : \mathcal{M}\ A$

specification

$w_c : \mathrm{W}\ A$

Dijkstra monad [Swamy'13]

$c : \mathcal{D}^{\mathcal{M}}\ A\ w_c$

What is a Dijkstra monad?

$\mathcal{M}$
computational monad

W
specification monad

code

$c : \mathcal{M}\ A$

specification

$w_c : \mathrm{W}\ A$

Dijkstra monad [Swamy‘13]

$c : \mathcal{D}^{\mathcal{M}}\ A\ w_c$

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

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

$\mathtt{weaken}^{\mathcal{D}^{\mathcal{M}}} : (w \leq^{\mathrm{W}} w') \to \mathcal{D}^{\mathcal{M}}\ A\ w \to \mathcal{D}^{\mathcal{M}}\ A\ w'$

F⋆, Dijkstra monads and Verification Conditions


```
val fibonacci : n:ℤ → Pure ℤ (w_fib n)
let rec fibonacci n =
  if n = 0 ‖ n = 1
  then 1
  else fibonacci (n-1) + fibonacci (n-2)
```

F⋆, Dijkstra monads and Verification Conditions


```
val fibonacci : n:ℤ → Pure ℤ (w_fib n)
let rec fibonacci n =
  if n = 0 || n = 1
  then ret 1
  else
   bind (fibonacci (n-1))
     (λ n_1 → bind (fibonacci (n-2))
       (λ n_2 → ret (n_1 + n_2)))
```

F⋆, Dijkstra monads and Verification Conditions


```
val fibonacci : n:ℤ → Pure ℤ (w_fib n)
let rec fibonacci n =
  if n = 0 ‖ n = 1
  then (ret 1 : Pure ℤ (ret 1))
  else
   (bind (fibonacci (n-1))
     (λ n1 → bind (fibonacci (n-2))
       (λ n2 → ret (n1 + n2)))
       : Pure ℤ (bind (w_fib (n-1)) (λ n1→
                     bind (w_fib (n-2)) ...))
```

F⋆, Dijkstra monads and Verification Conditions


```
val fibonacci : n:ℤ → Pure ℤ (w_fib n)
let rec fibonacci n =
   (if n = 0 ‖ n = 1
   then ret 1
   else
    bind (fibonacci (n-1))
      (λ n1 → bind (fibonacci (n-2))
        (λ n2 → ret (n1 + n2))))
   : Pure ℤ (if n = 0 ‖ n = 1 then ret 1
             else bind (w_fib (n-1)) ...)
```

F⋆, Dijkstra monads and Verification Conditions


```
val fibonacci : n:ℤ → Pure ℤ (w_fib n)
let rec fibonacci n =
   (if n = 0 ∥ n = 1
   then ret 1
   else
    bind (fibonacci (n-1))
      (λ n_1 → bind (fibonacci (n-2))
        (λ n_2 → ret (n_1 + n_2))))
   : Pure ℤ (if n = 0 ∥ n = 1 then ret 1
             else bind (w_fib (n-1)) ...)
```

- **A few fixed Dijkstra monads:**

Pure St Exc Div All

Can Dijkstra monads capture arbitrary monadic effects?

From effect observation to Dijkstra monad

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

From effect observation to Dijkstra monad

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

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

From effect observation to Dijkstra monad

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

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

Extends to a (categorical) equivalence

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

Dijkstra monads from monad transformers

$$\color{red}{\mathrm{Id}} \xrightarrow{\texttt{ret}} \color{blue}{\mathrm{Cont}_{\mathbb{P}}}$$

Dijkstra monads from monad transformers

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

Associates a Dijkstra monad $\mathcal{D}^{\mathcal{T}}$ to any monad $\mathcal{M} = \mathcal{T}(\mathrm{Id})$

Dijkstra monads from monad transformers

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

Associates a Dijkstra monad $\mathcal{D}^{\mathcal{T}}$ to any monad $\mathcal{M} = \mathcal{T}(\mathrm{Id})$

Monad transformers can be derived from monads

State	$C[X] = \mathcal{S} \to \mathbb{M}(X \times \mathcal{S})$
Exceptions	$C[X] = \mathbb{M}(X + \mathcal{E})$
Monotonic State	$C[X] = (s_0 : \mathcal{S}) \to \mathbb{M}(X \times \{s_1 : \mathcal{S} \mid s_0 \leq s_1\})$

as long as they fit the following grammar:

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

Contributions

- An algebraic definition of Dijkstra monads (6 equations)
- Dijkstra monads-Effect observations correspondence
  $\rightsquigarrow$ New examples of Dijkstra monads for non-determinism, IO
- Deriving monad transformers from a metalanguage

Presented at ICFP'19 in *Dijkstra Monads for All*,

Towards Relational Verification

What is relational verification?

Proving that 2 runs of a program,
or 2 different programs share a common specification.

Hyper-properties between multiple executions a single program

- **Non-interference** (NI)
  Public outputs only depend on public inputs
- **Differential Privacy**

Relational properties between two distinct programs

- **Program equivalence**
  Programs exhibit the same behaviours
- **Refinements**
- **Relative cost analysis**

Relational program logics

- Relational Hoare Logic [Benton'04]

$$\vdash \{\, p \,\}\ c \sim c'\ \{\, q \,\}$$

Relational program logics

- Relational Hoare Logic [Benton'04]

$$\vdash \{\, p \,\}\ c \sim c' \ \{\, q \,\}$$

- Triples are derived using inference rules:

$$\textsc{Seq}\ \frac{\vdash \{\, p \,\}\ c_1 \sim c_1' \ \{\, q \,\} \qquad \vdash \{\, q \,\}\ c_2 \sim c_2' \ \{\, r \,\}}{\vdash \{\, p \,\}\ c_1; c_2 \sim c_1'; c_2' \ \{\, r \,\}}$$

Relational program logics

- Relational Hoare Logic [Benton'04]

$$\vdash \{\, p \,\}\ c \sim c' \ \{\, q \,\}$$

- Triples are derived using inference rules:

$$\textsc{Seq}\ \frac{\vdash \{\, p \,\}\ c_1 \sim c_1' \ \{\, q \,\} \qquad \vdash \{\, q \,\}\ c_2 \sim c_2' \ \{\, r \,\}}{\vdash \{\, p \,\}\ c_1; c_2 \sim c_1'; c_2' \ \{\, r \,\}}$$

- Specific to state and non-termination
- Other relational logics for different effects
  (e.g. ($\times$)pRHL [Barthe et al.'09-19] for probabilities)

From unary to relational setting

Unary setting:

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

Relational setting:

$$\mathcal{M}_1, \mathcal{M}_2 \xrightarrow{\quad\theta^{\mathtt{rel}}\quad} \mathrm{W}_{\mathtt{rel}}$$

In the most general case, specifying and verifying

- 2 *distinct* programs,
- with *different* effects,
- at potentially *unrelated* types.

Reconstructing relational program logics

$$\mathcal{M}_1, \mathcal{M}_2 \xrightarrow{\theta^{\mathtt{rel}}} \mathrm{W}_{\mathtt{rel}}$$

Relational judgements:

$$\vdash c_1 \sim c_2 \ \{ \ w \ \}$$

$c_1 : \mathcal{M}_1 \, A_1, \ c_2 : \mathcal{M}_2 \, A_2, \ w : \mathrm{W}_{\mathtt{rel}}(A_1, A_2).$

$$\vDash^{\theta^{\mathtt{rel}}} c_1 \sim c_2 \ \{ \ w \ \} \qquad \Longleftrightarrow \qquad \theta^{\mathtt{rel}}(c_1, c_2) \leq^{\mathrm{W}_{\mathtt{rel}}} w$$

Reconstructing relational program logics

$$\mathcal{M}_1, \mathcal{M}_2 \xrightarrow{\quad\theta^{\mathtt{rel}}\quad} \mathrm{W}_{\mathtt{rel}}$$

Relational judgements:

$$\vdash c_1 \sim c_2 \ \{\ w\ \}$$

$c_1 : \mathcal{M}_1\, A_1,\ c_2 : \mathcal{M}_2\, A_2,\ w : \mathrm{W}_{\mathtt{rel}}(A_1, A_2).$

$$\vDash^{\theta^{\mathtt{rel}}} c_1 \sim c_2 \ \{\ w\ \} \qquad \Longleftrightarrow \qquad \theta^{\mathtt{rel}}(c_1, c_2) \leq^{\mathrm{W}_{\mathtt{rel}}} w$$

Three groups of inference rules for deriving relational judgements:

- ▷ logical rules (inherited from the metatheory)
- ▷ generic monadic rules
- ▷ effect specific rules

Generic monadic rules

$$\textsc{Ret}\ \frac{a_1 : A_1 \qquad a_2 : A_2}{\vdash \mathtt{ret}^{\mathcal{M}_1}\, a_1 \sim \mathtt{ret}^{\mathcal{M}_2}\, a_2 \ \left\{ \mathtt{ret}^{\mathrm{W}_{\mathtt{rel}}}\, (a_1, a_2) \right\}}$$

$$\textsc{Weaken}\ \frac{\vdash c_1 \sim c_2 \ \{ w \} \qquad w \leq w'}{\vdash c_1 \sim c_2 \ \{ w' \}}$$

$$\textsc{Bind}\ \frac{\begin{array}{c} \vdash m_1 \sim m_2 \ \{ w^m \} \\ \forall a_1, a_2 \quad \vdash f_1\, a_1 \sim f_2\, a_2 \ \left\{ w^f\, (a_1, a_2) \right\} \end{array}}{\vdash \mathtt{bind}^{\mathcal{M}_1}\, m_1\, f_1 \sim \mathtt{bind}^{\mathcal{M}_2}\, m_2\, f_2 \ \left\{ \mathtt{bind}^{\mathrm{W}_{\mathtt{rel}}}\, w^m\, w^f \right\}}$$

Relational Specification Monads

Provides a type of relational specifications for two result types A_1, A_2

$$\mathrm{W}^{\mathrm{St}}_{\texttt{rel}}(A_1, A_2) = \underbrace{((A_1 \times \mathcal{S}_1) \times (A_2 \times \mathcal{S}_2) \to \mathbb{P})}_{\text{post-relation}} \quad \longrightarrow \quad \underbrace{\mathcal{S}_1 \times \mathcal{S}_2 \to \mathbb{P}}_{\text{pre-relation}}$$

Relational Specification Monads

Provides a type of relational specifications for two result types A_1, A_2

$$\mathrm{W}^{\mathrm{St}}_{\mathtt{rel}}(A_1, A_2) = \underbrace{((A_1 \times \mathcal{S}_1) \times (A_2 \times \mathcal{S}_2) \to \mathbb{P})}_{\text{post-relation}} \quad \longrightarrow \quad \underbrace{\mathcal{S}_1 \times \mathcal{S}_2 \to \mathbb{P}}_{\text{pre-relation}}$$

$$\mathrm{W}^{\mathrm{St}}_{\mathtt{rel}} \quad : \quad \mathrm{Type} \times \mathrm{Type} \quad \longrightarrow \quad \mathrm{Type}$$

Relational Specification Monads

Provides a type of relational specifications for two result types A_1, A_2

$$\mathrm{W}^{\mathrm{St}}_{\mathtt{rel}}(A_1, A_2) = \underbrace{((A_1 \times \mathcal{S}_1) \times (A_2 \times \mathcal{S}_2) \to \mathbb{P})}_{\text{post-relation}} \quad \longrightarrow \quad \underbrace{\mathcal{S}_1 \times \mathcal{S}_2 \to \mathbb{P}}_{\text{pre-relation}}$$

$$\mathrm{W}^{\mathrm{St}}_{\mathtt{rel}} \quad : \quad \mathrm{Type} \times \mathrm{Type} \quad \longrightarrow \quad \mathrm{Type}$$

With operations induced by the continuation monad to $\mathbb{P}$

$$\mathtt{ret}^{\mathrm{W}^{\mathrm{St}}_{\mathtt{rel}}} : A_1 \times A_2 \longrightarrow \mathrm{W}^{\mathrm{St}}_{\mathtt{rel}}(A_1, A_2)$$

$$\mathtt{bind}^{\mathrm{W}^{\mathrm{St}}_{\mathtt{rel}}} : (A_1 \times A_2 \to \mathrm{W}^{\mathrm{St}}_{\mathtt{rel}}(B_1, B_2)) \longrightarrow \mathrm{W}^{\mathrm{St}}_{\mathtt{rel}}(A_1, A_2) \to \mathrm{W}^{\mathrm{St}}_{\mathtt{rel}}(B_1, B_2)$$

Relational Specification Monads

Provides a type of relational specifications for two result types A_1, A_2

$$\mathrm{W}^{\mathrm{St}}_{\mathtt{rel}}(A_1, A_2) = \underbrace{((A_1 \times \mathcal{S}_1) \times (A_2 \times \mathcal{S}_2) \to \mathbb{P})}_{\text{post-relation}} \quad \longrightarrow \quad \underbrace{\mathcal{S}_1 \times \mathcal{S}_2 \to \mathbb{P}}_{\text{pre-relation}}$$

$$\mathrm{W}^{\mathrm{St}}_{\mathtt{rel}} \quad : \quad \mathrm{Type} \times \mathrm{Type} \quad \longrightarrow \quad \mathcal{O}\mathrm{rd}$$

With operations induced by the continuation monad to $\mathbb{P}$

$$\mathtt{ret}^{\mathrm{W}^{\mathrm{St}}_{\mathtt{rel}}} : A_1 \times A_2 \longrightarrow \mathrm{W}^{\mathrm{St}}_{\mathtt{rel}}(A_1, A_2)$$

$$\mathtt{bind}^{\mathrm{W}^{\mathrm{St}}_{\mathtt{rel}}} : (A_1 \times A_2 \to \mathrm{W}^{\mathrm{St}}_{\mathtt{rel}}(B_1, B_2)) \longrightarrow \mathrm{W}^{\mathrm{St}}_{\mathtt{rel}}(A_1, A_2) \to \mathrm{W}^{\mathrm{St}}_{\mathtt{rel}}(B_1, B_2)$$

Relational Effect Observations

Observing stateful programs:

$$\theta^{\mathrm{St}} : \mathrm{St}_1\, A_1 \times \mathrm{St}_2\, A_2 \quad \longrightarrow \quad \mathrm{W}^{\mathrm{St}}_{\mathtt{rel}}(A_1, A_2)$$

$$\theta^{\mathrm{St}}(c_1, c_2) = \lambda post\ (s_1, s_2).\, post\,(c_1\, s_1, c_2\, s_2)$$

θ^{St} respects returns and binds

Relational Effect Observations

Observing stateful programs:

$$\theta^{\mathrm{St}} : \mathrm{St}_1\, A_1 \times \mathrm{St}_2\, A_2 \quad \longrightarrow \quad \mathrm{W}^{\mathrm{St}}_{\mathtt{rel}}(A_1, A_2)$$

$$\theta^{\mathrm{St}}(c_1, c_2) = \lambda post\ (s_1, s_2).\, post\,(c_1\, s_1, c_2\, s_2)$$

θ^{St} respects returns and binds

$$\frac{}{\vdash \mathtt{get}\,() \sim \mathtt{ret}\, a_2\ \{\, \theta^{\mathrm{St}}(\mathtt{get}\,(), \mathtt{ret}\, a_2)\,\}}$$

$$\theta^{\mathrm{St}}(\mathtt{get}\,(), \mathtt{ret}\, a_2) = \lambda post\ (s_1, s_2).\, post\ ((s_1, s_1), (a_2, s_2))$$

Relational Effect Observations

Observing stateful programs:

$$\theta^{\mathrm{St}} : \mathrm{St}_1\, A_1 \times \mathrm{St}_2\, A_2 \quad \longrightarrow \quad \mathrm{W}^{\mathrm{St}}_{\mathtt{rel}}(A_1, A_2)$$

$$\theta^{\mathrm{St}}(c_1, c_2) = \lambda post\ (s_1, s_2).\, post\,(c_1\, s_1, c_2\, s_2)$$

θ^{St} respects returns and binds

$$\frac{}{\vdash \mathtt{get}\,() \sim \mathtt{ret}\, a_2\ \left\{\, \theta^{\mathrm{St}}(\mathtt{get}\,(), \mathtt{ret}\, a_2)\, \right\}}$$

$$\theta^{\mathrm{St}}(\mathtt{get}\,(), \mathtt{ret}\, a_2) = \lambda post\ (s_1, s_2).\, post\ ((s_1, s_1), (a_2, s_2))$$

Also Relational effect observations for:

Non-determinism, Exception, IO, Probabilities...

The problem with exceptions...

$$\mathrm{W}^{\mathrm{Exc}}_{\mathtt{rel}}(A_1, A_2) = ((A_1 + \mathcal{E}_1) \times (A_2 + \mathcal{E}_2) \to \mathbb{P}) \to \mathbb{P}$$

The problem with exceptions...

$$\mathrm{W}^{\mathrm{Exc}}_{\mathtt{rel}}(A_1, A_2) = ((A_1 + \mathcal{E}_1) \times (A_2 + \mathcal{E}_2) \to \mathbb{P}) \to \mathbb{P}$$

```
let bindWrelExc wm wf post =
  wm (λ x →
     match x with
     | Inl a₁ , Inl a₂ → wf (a₁ ,a₂) post
     | Inr e₁ , Inr e₂ → post (Inr e₁ , Inr e₂)
     | _ → ?? )
```

$\mathtt{wm} : \mathrm{W}^{\mathrm{Exc}}_{\mathtt{rel}}(A_1, A_2)$ $\mathtt{wf} : A_1 \times A_2 \to \mathrm{W}^{\mathrm{Exc}}_{\mathtt{rel}}(B_1, B_2)$

The problem with exceptions…

$$\mathrm{W}^{\mathrm{Exc}}_{\mathtt{rel}}(A_1, A_2) = ((A_1 + \mathcal{E}_1) \times (A_2 + \mathcal{E}_2) \to \mathbb{P}) \to \mathbb{P}$$

```
let bindWrelExc wm (wf₁, wf₂, wf) post =
  wm (λ x →
    match x with
    | Inl a₁, Inl a₂ → wf (a₁,a₂) post
    | Inr e₁, Inr e₂ → post (Inr e₁, Inr e₂)
    | Inl a₁, Inr e₂ → wf₁ a₁ (λ y₁ → post (y₁, Inr e₂))
    | Inr e₂, Inl a₂ → wf₂ a₂ (λ y₂ → post (Inr e₁, y₂)))
```

$\mathtt{wf}_1 : A_1 \to \mathrm{W}^{\mathrm{Exc}} B_1$ $\qquad$ $\mathtt{wf}_2 : A_2 \to \mathrm{W}^{\mathrm{Exc}} B_2$

$$\mathtt{wf} : A_1 \times A_2 \to \mathrm{W}^{\mathrm{Exc}}_{\mathtt{rel}}(B_1, B_2)$$

Reconstructing relational program logics (reloaded)

$$\mathrm{W}^{\mathrm{Exc}}_{\mathtt{rel}}(A_1, A_2) = ((A_1 + \mathcal{E}_1) \times (A_2 + \mathcal{E}_2) \to \mathbb{P}) \to \mathbb{P}$$

Refined Judgements $\quad \vdash c_1 \ \{w_1\} \sim c_2 \ \{w_2\} \mid w_{\mathtt{rel}}$

Reconstructing relational program logics (reloaded)

$$\mathrm{W}^{\mathrm{Exc}}_{\mathtt{rel}}(A_1, A_2) = ((A_1 + \mathcal{E}_1) \times (A_2 + \mathcal{E}_2) \to \mathbb{P}) \to \mathbb{P}$$

Refined Judgements $\quad \vdash c_1 \ \{w_1\} \sim c_2 \ \{w_2\} \mid w_{\mathtt{rel}}$

$$\frac{\begin{array}{ll} & \vdash m_1 \ \{w^m_1\} \sim m_2 \ \{w^m_2\} \mid w^m_{\mathtt{rel}} \\ \forall a_1, a_2 & \vdash f_1 \ a_1 \ \{w^f_1 \ a_1\} \sim f_2 \ a_2 \ \{w^m_2 \ a_2\} \mid w^f_{\mathtt{rel}} \ a_1 \ a_2 \end{array}}{\vdash \begin{array}{ll} \mathtt{bind}^{\mathrm{Exc}_1} \ m_1 \ f_1 & \{\mathtt{bind}^{\mathrm{W}^{\mathrm{Exc}}_1} \ w^m_1 \ w^f_1\} \\ \qquad \sim & \\ \mathtt{bind}^{\mathrm{Exc}_2} \ m_2 \ f_2 & \{\mathtt{bind}^{\mathrm{W}^{\mathrm{Exc}}_2} \ w^m_2 \ w^f_2\} \end{array} \Bigg| \ \mathtt{bind}^{\mathrm{W}^{\mathrm{Exc}}_{\mathtt{rel}}} \ \boldsymbol{w^m} \ \boldsymbol{w^f}}$$

Contributions

- An extension of specification monads and effect observation to relational verification
- A generic framework for deriving relational program logics for arbitrary monadic effects
- A new insight on relational programs logics with exceptions

Accepted at POPL'20 as *The Next 700 Relational Program Logics*

A Unifying Categorical Framework

Relative monads & morphisms

A **relative monad** [Altenkirch et al.'15] on a functor $\mathcal{J} : \mathcal{I} \to \mathcal{C}$ is

- a functor $\mathcal{T} : \mathcal{I} \to \mathcal{C}$,
- equipped with operations $(a, b \in \mathcal{I})$ and equations

$$\mathtt{ret}_a^{\mathcal{T}} \in \mathcal{C}(\mathcal{J}a, \mathcal{T}a) \qquad \mathtt{bind}_{a,b}^{\mathcal{T}} : \mathcal{C}(\mathcal{J}a, \mathcal{T}b) \to \mathcal{C}(\mathcal{T}a, \mathcal{T}b)$$

Relative monads & morphisms

A **relative monad** [Altenkirch et al.'15] on a functor $\mathcal{J} : \mathcal{I} \to \mathcal{C}$ is

- a functor $\mathcal{T} : \mathcal{I} \to \mathcal{C}$,
- equipped with operations $(a, b \in \mathcal{I})$ and equations

$$\mathtt{ret}^{\mathcal{T}}_a \in \mathcal{C}(\mathcal{J}a, \mathcal{T}a) \qquad \mathtt{bind}^{\mathcal{T}}_{a,b} : \mathcal{C}(\mathcal{J}a, \mathcal{T}b) \to \mathcal{C}(\mathcal{T}a, \mathcal{T}b)$$

Relative monad morphism $\theta^{\mathtt{rel}}$ from $\mathcal{T}_1$ to $\mathcal{T}_2$ over $\mathcal{F} : \mathcal{J}_1 \to \mathcal{J}_2$

$$\begin{tikzcd} & \mathcal{C}_1 \arrow[dd, "\mathcal{F}"] \\ \mathcal{I} \arrow[ur, "\mathcal{J}_1"] \arrow[dr, "\mathcal{J}_2"'] \arrow[r, phantom, "\circlearrowright"] & \\ & \mathcal{C}_2 \end{tikzcd} \qquad \begin{tikzcd} & \mathcal{C}_1 \arrow[dd, "\mathcal{F}"] \\ \mathcal{I} \arrow[ur, "\mathcal{T}_1"] \arrow[dr, "\mathcal{T}_2"'] \arrow[r, phantom, "\Downarrow \theta^{\mathtt{rel}}"] & \\ & \mathcal{C}_2 \end{tikzcd}$$

respecting `ret` and `bind`.

Concrete instances

A *specification monad* is a relative monad over

$$\mathcal{D}\mathrm{iscr} : \mathrm{Type} \longrightarrow \mathcal{O}\mathrm{rd}$$

A *relational specification monad* is a relative monad over

$$\mathcal{J}_{\times} : \begin{array}{ccc} \mathrm{Type} \times \mathrm{Type} & \longrightarrow & \mathcal{O}\mathrm{rd} \\ (A_1, A_2) & \longmapsto & \mathcal{D}\mathrm{iscr}(A_1 \times A_2) \end{array}$$

$\mathrm{W}^{\mathrm{Exc}}_{\mathtt{rel}}$ is a $\mathcal{J}_{\mathtt{rel}}$-relative monad where

$$\mathcal{J}_{\mathtt{rel}} : \begin{array}{ccc} \mathrm{Type} \times \mathrm{Type} & \longrightarrow & \mathcal{S}\mathrm{pan}(\mathcal{O}\mathrm{rd}) \\ (A_1, A_2) & \longmapsto & \begin{tikzcd} & \mathcal{D}\mathrm{iscr}(A_1 \times A_2) \arrow[dl] \arrow[dr] & \\ \mathcal{D}\mathrm{iscr}\, A_1 & & \mathcal{D}\mathrm{iscr}\, A_2 \end{tikzcd} \end{array}$$

Enrichment

For a specification monad W

$$\mathtt{bind}^{\mathrm{W}}_{A,B} \quad : \quad \mathcal{O}\mathrm{rd}(\mathcal{D}\mathrm{iscr}\, A, \mathrm{W}\, B) \quad \longrightarrow \quad \mathcal{O}\mathrm{rd}(\mathrm{W}\, A, \mathrm{W}\, B)$$

$\mathtt{bind}$ monotonic $\iff$ $\mathtt{bind}$ lifts to $\mathcal{O}\mathrm{rd}$-enriched categories

Enrichment

For a specification monad W

$$\mathtt{bind}^{\mathrm{W}}_{A,B} \quad : \quad \mathcal{O}\mathrm{rd}(\mathcal{D}\mathrm{iscr}\, A, \mathrm{W}\, B) \quad \longrightarrow \quad \mathcal{O}\mathrm{rd}(\mathrm{W}\, A, \mathrm{W}\, B)$$

`bind` monotonic $\iff$ `bind` lifts to $\mathcal{O}\mathrm{rd}$-enriched categories

What if we want to preserve another structure, e.g. measurability?

Enrichment

For a specification monad W

$$\mathtt{bind}^{\mathrm{W}}_{A,B} \quad : \quad \mathcal{O}\mathrm{rd}(\mathcal{D}\mathrm{iscr}\,A, \mathrm{W}\,B) \quad \longrightarrow \quad \mathcal{O}\mathrm{rd}(\mathrm{W}\,A, \mathrm{W}\,B)$$

`bind` monotonic $\iff$ `bind` lifts to $\mathcal{O}\mathrm{rd}$-enriched categories

What if we want to preserve another structure, e.g. measurability?

$\rightsquigarrow$ Enriched relative monads

Enrichment

For a specification monad W

$$\texttt{bind}^{\mathrm{W}}_{A,B} \quad : \quad \mathcal{O}\mathrm{rd}(\mathcal{D}\mathrm{iscr}\, A, \mathrm{W}\, B) \quad \longrightarrow \quad \mathcal{O}\mathrm{rd}(\mathrm{W}\, A, \mathrm{W}\, B)$$

$\texttt{bind}$ monotonic $\iff$ $\texttt{bind}$ lifts to $\mathcal{O}\mathrm{rd}$-enriched categories

What if we want to preserve another structure, e.g. measurability?

$\rightsquigarrow$ Enriched relative monads

A **relative** variant of *The Formal Theory of Monads* [Street'72]

Relative monads synthetically

Use **framed bicategories** to abstract over hom-distributors

$\bullet$

Relative monads synthetically

Use **framed bicategories** to abstract over hom-distributors

-
-

Relative monads synthetically

Use **framed bicategories** to abstract over hom-distributors

$$\begin{tikzcd}
\bullet & \\
\bullet & \bullet
\end{tikzcd}$$

Relative monads synthetically

Use **framed bicategories** to abstract over hom-distributors

$$\begin{tikzcd} \bullet & \bullet \\ \bullet & \bullet \end{tikzcd}$$

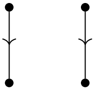
Relative monads synthetically

Use **framed bicategories** to abstract over hom-distributors

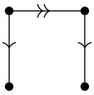
$$\begin{tikzcd}
\bullet \arrow[d, "\shortmid" marking] & \bullet \\
\bullet & \bullet
\end{tikzcd}$$

Relative monads synthetically

Use **framed bicategories** to abstract over hom-distributors

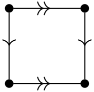


Use **framed bicategories** to abstract over hom-distributors



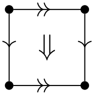
Relative monads synthetically

Use **framed bicategories** to abstract over hom-distributors



Relative monads synthetically

Use **framed bicategories** to abstract over hom-distributors



Relative monads synthetically

Use **framed bicategories** to abstract over hom-distributors

$$
\begin{tikzcd}
\bullet \arrow[r] \arrow[d] \arrow[dr, phantom, "\Downarrow"] & \bullet \arrow[d] \\
\bullet \arrow[r] & \bullet
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
\bullet \arrow[d, "\mathcal{F}"'] & \bullet \arrow[d, "\mathcal{G}"] \\
\bullet \arrow[r, "\mathcal{R}"'] & \bullet
\end{tikzcd}
$$

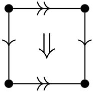
Relative monads synthetically

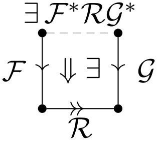
Use **framed bicategories** to abstract over hom-distributors

$$\begin{tikzcd} \bullet \arrow[r, two heads] \arrow[d] \arrow[dr, phantom, "\Downarrow"] & \bullet \arrow[d] \\ \bullet \arrow[r, two heads] & \bullet \end{tikzcd} \qquad \begin{tikzcd} \bullet \arrow[r, dashed, "\exists\, \mathcal{F}^* \mathcal{R} \mathcal{G}^*"] \arrow[d, "\mathcal{F}"'] \arrow[dr, phantom, "\Downarrow \exists"] & \bullet \arrow[d, "\mathcal{G}"] \\ \bullet \arrow[r, two heads, "\mathcal{R}"'] & \bullet \end{tikzcd}$$

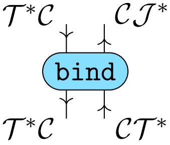
Relative monads synthetically

Use **framed bicategories** to abstract over hom-distributors



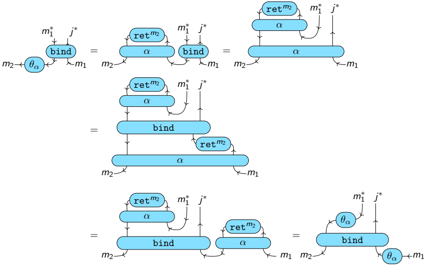


and string diagrams to works with these abstract objects.



Contributions

- Extending Relative monad morphisms over distinct base functors
- A formal theory of relative monads in framed bicategories
- Conservativity over the classic formal theory of monads



String diagrams at work

Summary

- Specifications for arbitrary monadic effects through monads
- Effect observations decouple program syntax from multiple semantics
- Enables a generic reconstruction of Dijkstra monads and relational program logics

Contributions:

- $\rightsquigarrow$ Introduction of specification monads and their transformers
- $\rightsquigarrow$ Connecting Dijkstra monads to effect observations
- $\rightsquigarrow$ And extending to the relational setting
- Implemented in Coq
- $\rightsquigarrow$ A synthetic theory of relative monads in framed bicategories

Discrete Probabilities

Computational monad:

- **Giry monad**: formal barycentric sums (finite distributions)
- **Distributions**

Specification monads

$$\text{Unary}\quad \mathrm{W}^{\mathrm{Prob}}(X) = (X \to \mathcal{I}) \xrightarrow{mon,cont} \mathcal{I}$$

$$\text{Relational}\quad \mathrm{W}^{\mathrm{Prob}}(A_1, A_2) = (A_1 \times A_2 \to \mathcal{I}) \xrightarrow{mon,cont} \mathcal{I}$$

Relational Effect Observation

$$\theta^{\mathrm{Prob}}(c_1, c_2) = \lambda post.\ \inf_{d \sim c_1, c_2} \sum_{a_1 : A_1, a_2 : A_2} d(a_1, a_2) \cdot post(a_1, a_2)$$

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

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

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

Observation 1: if C and $\mathcal{M}$ are monads, $\mathcal{T}^{C}(\mathcal{M}) = C[\mathcal{M}/\mathbb{M}]$ is a monad

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: if C and $\mathcal{M}$ are monads, $\mathcal{T}^C(\mathcal{M}) = C[\mathcal{M}/\mathbb{M}]$ is a monad

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

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: if C and $\mathcal{M}$ are monads, $\mathcal{T}^C(\mathcal{M}) = C[\mathcal{M}/\mathbb{M}]$ is a monad

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

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

Logical rules

$$\mathbb{B}\text{-}\textsc{Elim}\ \frac{\texttt{if}\ b\ \texttt{then}\ \vdash c_1 \sim c_2\ \left\{\, w^{\top} \,\right\}\ \texttt{else}\ \vdash c_1 \sim c_2\ \left\{\, w^{\perp} \,\right\}}{\vdash c_1 \sim c_2\ \left\{\, \texttt{if}\ b\ \texttt{then}\ w^{\top}\ \texttt{else}\ w^{\perp} \,\right\}}$$

$\mathbb{N}$-Elim

$$\frac{\begin{array}{c} n : \mathbb{N} \qquad w = \texttt{elim}^{\mathbb{N}}\ w_0\ w_{suc} \qquad \vdash c_1[0/n] \sim c_2[0/n]\ \{\, w_0 \,\} \\ \forall n : \mathbb{N},\ \vdash c_1 \sim c_2\ \{\, w\ n \,\} \quad \Rightarrow \quad \vdash c_1[\mathtt{S}\ n/n] \sim c_2[\mathtt{S}\ n/n]\ \{\, w_{suc}\ (w\ n) \,\} \end{array}}{\vdash c_1 \sim c_2\ \{\, w\ n \,\}}$$

- ▷ Independent from the effects and the observation $\theta^{\texttt{rel}}$
- ▶ Use the ambient metatheory (e.g. Coq)
- ▷ Validated by dependent pattern matching