Dijkstra monads and relational reasoning

Exequiel Rivas – Prosecco, Inria

Journée scientifique Nomadic Labs – Inria
21 de septembre 2020

F*: bridging the gap

- **F* is a functional programming language with effects**
  - like OCaml, F#, Haskell…
  - F* is extracted to OCaml
  - subset of F* compiled to efficient C code
  - home to HACL*, a formally verified library of modern cryptographic algorithms

- **Semi-automated verification system using SMT**
  - like Dafny, FramaC, Why3, etc.
- **With an expressive core language based on dependent type theory**
  - like Coq, Lean, Agda, Idris, etc.
- **A metaprogramming and tactic framework for interactive proof and user-defined automation:**
  - like Coq, Isabelle, Lean, Nuprl, etc.

Effects in F*

- F* is centered around verification of *effectful* code.
- Two classes of types:
  - *values* (int, list int, etc.).
  - *computations* (pure, divergent, stateful, etc.).

- Pure code cannot call effectful code, via sub-effecting pure code can be used in effectful code.
- Computations come with specs.
- Dependent function types:

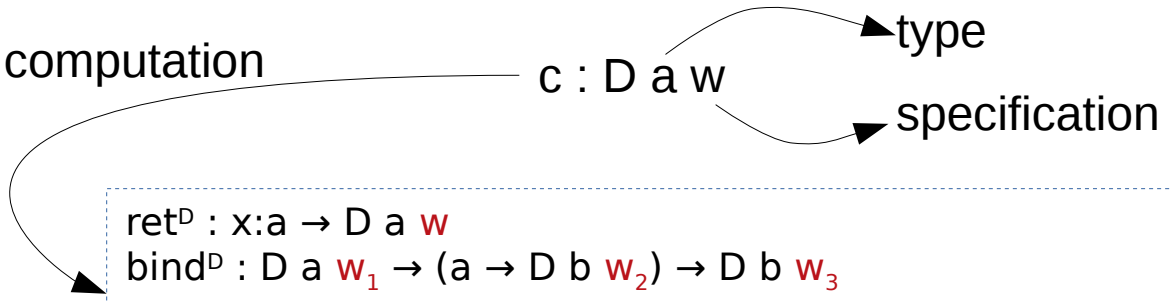
  x:t -> C.
- Call-by-value.

```
let test1 () : St int (fun s0 p -> p 42 (s0 + 5)) =
  let x = get () in
  set (x + 2);
  let y = get () in
  set (y + 3);
  42
```

Dijkstra monads

What's the nature of these effects?

Couple computation with its specification (describes behavior):



What's the mathematical structure that captures these effects?
Can we use it to obtain new effects in F*?

Dijkstra monads: slogan & theory

Dijkstra monad = computational monad + specification monad + monad morphism

- Effectful code is captured using a computational monad:

  $T : Type \to Type$

  $ret^T : x{:}a \to T\ a$

  $bind^T : T\ a \to (a \to T\ b) \to T\ b$

  $act^T : \ldots \to T\ a$

- Specifications are captured using specification monads:

  $W : Type \to Type$

  $ret^W : x{:}a \to W\ a$

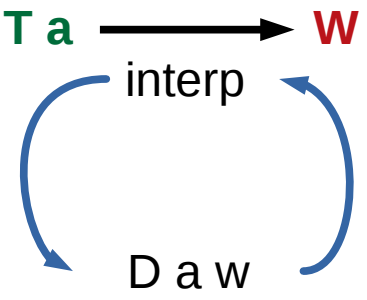
  $bind^W : W\ a \to (a \to W\ b) \to W\ b$

  $act^W : \ldots \to W\ a$

  $(\leq) : w_1{:}W\ a \to w_2{:}W\ a \to Type_0$

- Related through an effect observation:

  $interp : T\ a \to W\ a$



$ret^D : x{:}a \to D\ a\ (ret^W\ x)$
$bind^D : \#w{:}W\ a \to \#f{:}(a \to W\ b) \to \ldots \to D\ b\ (bind^W\ w\ f)$
$act^D : \ldots \to D\ a\ (act^W\ \ldots)$
$weaken^D : w_1{:}W\ a \to w_2{:}W\ a\{w_1 \leq w_2\} \to D\ a\ w_1 \to D\ a\ w_2$

$$\models c\ \{w\} = interp\ c \leq w$$

Dijkstra monads: examples

- The state Dijkstra monad is obtained as:

  $T\ a = S \to (S \times a)$

  $W\ a = S \to (S \times a \to Type_0) \to Type_0$

  interp (c : T a) = λs post. post (c s)

St a w

- The exception Dijkstra monad is obtained as:

  $T\ a = either\ e\ a$

  $W\ a = (either\ e\ a \to Type_0) \to Type_0$

  interp (c : T a) = λpost. post c

Exn a w

- The (demonic) non-determinism Dijkstra monad is obtained as:

  $T\ a = list\ a$

  $W\ a = (a \to Type_0) \to Type_0$

  interp (c : T a) = λpost. ∀ a ∈ c . post a

Nd a w

Dijkstra monads: implementation

```
new_effect {
  EXC : a:Type -> Effect
  with
       repr       = repr
T a  ; return     = return
     ; bind       = bind

     ; wp_type    = wp_type
W a  ; return_wp  = return_wp
     ; bind_wp    = bind_wp

     ; interp     = interp
}
```

- Implemented Dijkstra monads:
  F* and Coq.
- Mathematically inspired theory:
  built on top of monads and morphisms.
- Extend effects in F*:
  non-determinism, IO.
- Specification flexibility:
  context-free vs. history IO.
- Case study:
  small web server extracting to ML.

Relational reasoning

Discussed unary reasoning, all about a *single* program run:

$$c : D\ a\ w \quad \equiv \quad \vDash c\ \{w\}$$

But, a number of interesting properties are relational:
they talk about two executions of (possible different) programs.

E.g., non-interference:

$$\forall\ M_1\ M_2\ .\ M_1 =_L M_2 \wedge (P, M_1) \rightsquigarrow^* M_1' \wedge (P, M_2) \rightsquigarrow^* M_2' \Rightarrow M_1' =_L M_2'$$

Is it possible to extend the Dijkstra monad framework to account for relational properties?

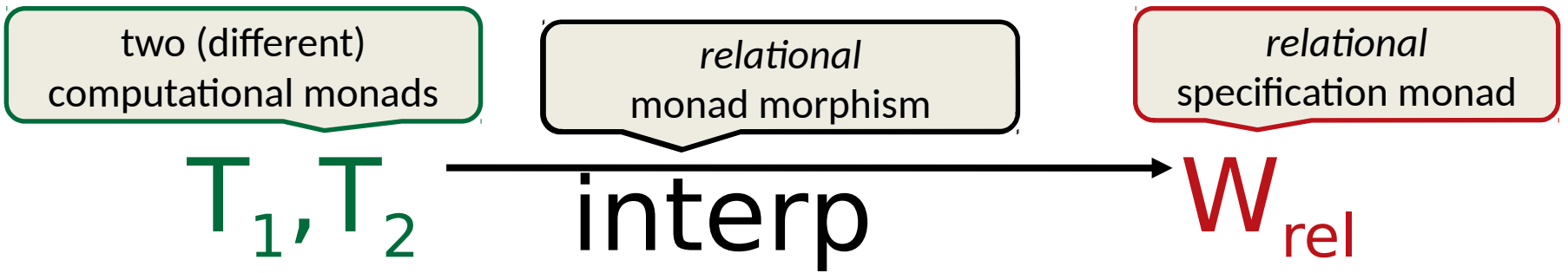
As a source of inspiration, we took the relational Hoare logic of Benton, and the subsequent developments by Barthe et al.:

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

Relational reasoning and Dijkstra monads

As before, effectful code is captured using computational monads.

Specification monads now have to account for relational properties: technically, a relative monad on a product category.



Define relational program logics,

using interp for the semantics: $\vDash c_1 \sim c_2 \{w\} = \text{interp}\ (c_1, c_2) \leq w$

Relational reasoning: quick example

$T_1\,A = S_1 \to A \times S_1 \qquad T_2\,A = S_2 \to A \times S_2$

$W_{rel}\,A_1\,A_2 = ((A_1 \times S_1) \times (A_2 \times S_2) \to \mathrm{Type}_0) \to S_1 \times S_2 \to \mathrm{Type}_0$

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

$\vDash c_1 \sim c_2\ \{w\} = \mathrm{interp}\ (c_1, c_2) \le w$

$$\textsc{Ret}\ \frac{a_1 : A_1 \qquad a_2 : A_2}{\vdash \mathsf{ret}^{M_1}\ a_1 \sim \mathsf{ret}^{M_2}\ a_2\ \left\{\ \mathsf{ret}^{W}\ (a_1, a_2)\ \right\}}$$

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

$$\frac{}{\vdash \mathsf{get}\ () \sim \mathsf{ret}\ a_2\ \{\ \lambda \varphi\ (s_1, s_2).\ \varphi\ ((s_1, s_1), (a_2, s_2))\ \}}$$

$$\textsc{Bind}\ \frac{\vdash m_1 \sim m_2\ \{\ w^m\ \} \qquad \forall a_1, a_2 \vdash f_1\ a_1 \sim f_2\ a_2\ \left\{\ w^f\ (a_1, a_2)\ \right\}}{\vdash \mathsf{bind}^{M_1}\ m_1\ f_1 \sim \mathsf{bind}^{M_2}\ m_2\ f_2\ \left\{\ \mathsf{bind}^{W_{rel}}\ w^m\ w^f\ \right\}}$$

$$\frac{}{\vdash \mathsf{put}\ s \sim \mathsf{ret}\ a_2\ \{\ \lambda \varphi\ (s_1, s_2).\ \varphi\ (((), s), (a_2, s_2))\ \}}$$

Prototype implemented in Coq

Ongoing and future work

- Dijkstra monads: we are currently working on a proper account for *partiality*.
- Layered effects: implementation of Dijkstra monads on top of layered effects. Is there a mathematical structure capturing layered effects?
- Following the web server line, we are doing some experiments on ML & F* inter-operability and secure compilation properties.
- Relational reasoning: exploring the relational framework with probabilities.
- Relational reasoning: find a good F* presentation.
- Relational reasoning: LIO non-interference in F* using parametricity.
- Théo Laurent just starting his PhD: sub-typing & effects in F*.

Thanks