Verificación de Programas con F*

Clase 9 – 21/05/2024

WPs en F*

¡Continuado!

- Estuvimos usando WPs todo el tiempo.
- El efecto `Pure a pre post` está definido sobre `PURE a wp`. Toda especificación de computaciones es con WPs
  - Estas WPs son funcionales (las funciones devuelven un valor) a diferencia de las puramente imperativas que estuvimos formalizando. No es una diferencia mayor.
- `Pure a pre post = PURE a (fun p -> pre /\ (forall x. post x ==> p x))`
- Cada efecto tiene su propio *cálculo* de WPs, que indica cómo computar WPs para programas en ese efecto. Cada cálculo forma una mónada.
- Opcional: ver paper Dijkstra Monads for Free
  - Desde una mónada à-la-Haskell, deriva un cálculo de WPs correcto automáticamente
  - F* se extiende con un nuevo efecto (simulado)

Mónada de Dijkstra – PURE (y DIV)

Para una computación pura, una WP transforma la postcondición (sobre el resultado) a una proposición pura.

"Mónada de Dijkstra"

```
PURE a wp ← Tipo de computación (efecto + tipo + especificación), wp tiene tipo pure_wp a
pure_wp a = (a -> prop) -> prop
return_wp #a (x:a) : wp a = fun p -> p x
bind_wp #a #b (w : wp a) (f : a -> wp b) : wp b = fun p -> w (fun x -> f x p)
```

Forma una mónada (en el sentido usual)

Podemos codificar pre/postcondiciones sobre WPs:

```
Pure a (requires pre) (ensures post) =
PURE a (fun p -> pre /\ (forall x. post x ==> p x))
```

Mónada de Dijkstra – Estado mutable

Para una computación con estados, temenos que poder hablar del estado inicial y final.

```
val st : Type
STATE a wp
st_wp a = (a -> state -> prop) -> (state -> prop)
return_wp #a (x:a) : wp a = fun p s0 -> p x s0
bind_wp #a #b (w : wp a) (f : a -> wp b) : wp b = fun p s0 -> w s0 (fun x s1 -> f x p s1)
```

```
State a (requires pre) (ensures post) =
STATE a (fun p s0 -> pre s0 /\ (forall x s1. post x s1 ==> p x s1))
```

```
val get () : STATE st (fun p s0 -> p s0 s0)
val set (s:st) : STATE unit (fun p _ -> p () s)
```

DM4F

$$
\begin{array}{lcl}
\mathrm{ST}_{wp}\ t & = & S \to (t \times S \to \mathsf{Type}_0) \to \mathsf{Type}_0 \\
\mathrm{returnwp}_{st}\ v & = & \lambda s_0\ p.\ p\ (v, s_0) \\
\mathrm{bindwp}_{st}\ wp\ f & = & \lambda s_0\ p.\ wp\ s_0\ (\lambda vs.\ f\ (\mathbf{fst}\ vs)\ (\mathbf{snd}\ vs)\ p) \\
\mathrm{getwp}_{st} & = & \lambda s_0\ p.\ p\ (s_0, s_0) \\
\mathrm{setwp}_{st}\ s_1 & = & \lambda \_\ p.\ p\ ((), s_1)
\end{array}
$$

$$
\begin{array}{lcl}
\mathrm{ST}\ t & = & S \to t \times S \\
\mathrm{return}_{st}\ v & = & \lambda s_0.\ (v, s_0) \\
\mathrm{bind}_{st}\ m\ f & = & \lambda s_0.\ \mathbf{let}\ vs = m\ s_0\ \mathbf{in}\ f\ (\mathbf{fst}\ vs)\ (\mathbf{snd}\ vs) \\
\mathrm{get} & = & \lambda s_0.\ (s_0, s_0) \\
\mathrm{set}\ s_1 & = & \lambda \_.\ ((), s_1)
\end{array}
$$

Mónada de Dijkstra – Excepciones

```
type result (a:Type) : Type =
  | V of a
  | E of exn
  | Err of string (* no recuperable *)

EXN a wp
exn_wp a = (result a -> prop) -> prop
return_wp #a (x:a) : wp a = fun p -> p (V x)
bind_wp #a #b (w : wp a) (f : a -> wp b) : wp b =
  fun p -> w (function V x -> f x p
                       | E e -> p (E e)
                       | Err msg -> p (Err msg))

val raise (#a:Type) (e:exn) : EXN a (fun p -> p (E e))
```

Mónada de Dijkstra – Excepciones

```
if x = 0 then
  raise Zero;
let y = 100 / x in
…
```

```
let _ =
  (if x = 0 then
   then raise Zero
   else ())
in
let y = 100 / x in
…
```

- La WP es algo como:

```
wp = bind_wp wp1 (fun () -> wp2)
wp1    = fun p -> (x = 0 ==> p (E Zero)) /\ (x <> 0 ==> p (V ())) (* recuerden WP del if en Imp *)
wp2 () = fun p -> ϕ
```

```
wp = fun p ->
       (x = 0 ==> p (E Zero)) /\
       (x <> 0 ==> ϕ) (* codifica control de flujo *)
```

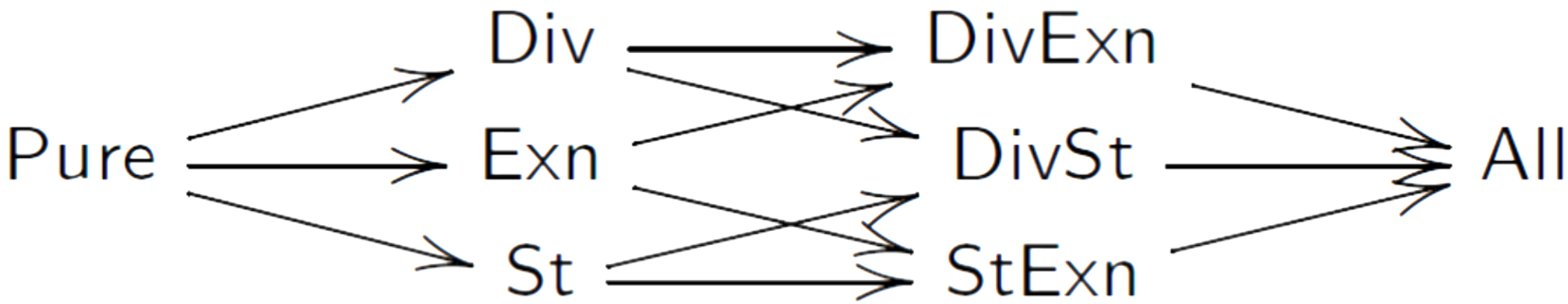
Mónada de Dijkstra – Todo junto

```
type result (a:Type) : Type =
  | V of a
  | E of exn
  | Err of string (* no recuperable *)

EXN a wp
all_wp a = (result a -> state -> prop) -> state -> prop
return_wp #a (x:a) : wp a = fun p s0 -> p (V x) s0
bind_wp #a #b (w : wp a) (f : a -> wp b) : wp b =
  fun p s0 -> w (function V x, s1 -> f x p s1
                        | E e, s1 -> p (E e) s1
                        | Err msg, s1 -> p (Err msg) s1)

val raise (#a:Type) (e:exn) : ALL a (fun p s0 -> p (E e) s0)
```

Retículo de efectos



Tareas

- Empezar a pensar proyectos
  - Una estructura de datos
  - Algún algoritmo relativamente simple
  - Algún programa real (con IO, etc) y algunas pruebas ligeras sobre el mismo
  - Formalizar alguna estructura matemática (grupo, anillo, etc)
  - Formalizar algún lenguaje de programación
- En breve subo un archivo sobre lo de hoy
- Ver bibliografía
- Próxima clase: intro a tácticas + calc