# States and exceptions are dual effects

Jean-Guillaume Dumas, Dominique Duval,
Laurent Fousse, Jean-Claude Reynaud

LJK, University of Grenoble

August 29, 2010
Workshop on Categorical Logic in Brno
*(this is a long version of the talk presented at the workshop)*

# Outline

# Semantics of computational effects?

The categorical semantics of functional programming languages is based on the Curry-Howard-Lambek correspondence:

| logic | programming | categories |
|---|---|---|
| *propositions* | *types* | *objects* |
| *proofs* | *terms* | *morphisms* |
| intuitionistic logic | simply typed lambda calculus | cartesian closed categories |

What about categorical semantics of non-functional programming languages, i.e., languages with effects?

| programming | categories |
|---|---|
| *effect* | *categorical structure ??* |
| (global) states | ?? |
| exceptions | ?? |

# Effects as monads

Moggi [1989], cf. Haskell:

*Programs of type B with a parameter of type A are interpreted by morphisms from A to T(B).*

$$\boxed{p : A \to B} \text{ is interpreted as } \boxed{p : A \to T(B)}$$

States. $p : A \to B$ is interpreted as $p : A \times St \to B \times St$, or $p : A \to (B \times St)^{St}$, where $St$ is the set of states

Exceptions. $p : A \to B$ is interpreted as $p : A \to Exc$, where $Exc$ is the set of exceptions

| effect | monad $(T, \eta, \mu)$ |
| --- | --- |
| states | $T(X) = (X \times St)^{St}$ |
| exceptions | $T(X) = X + Exc$ |

Note. What about the handling (catching) of exceptions?

# Effects as Lawvere theories

Plotkin & Power [2001]:

*Use the connection between monads and Lawvere theories to give operations a primitive role, with the monad as derived*

States. $Loc$ is the set of locations, $Val$ is the set of values ($St = Val^{Loc}$ is the set of states)

Exceptions. $Exc$ is the set of exceptions

| effect | Lawvere theory generated by |
|---|---|
| states | $lookup : Val \to Loc$<br>$update : 1 \to Loc \times Val$<br>with 7 equations |
| exceptions | $raise_e : 0 \to 1$ for $e \in Exc$<br>with no equation |

Note. What about the handling (catching) of exceptions?

# Effects as zooms (= spans of logics)

Following Moggi's remark:

$$\boxed{p : A \to B} \text{ is interpreted as } \boxed{p : A \to T(B)}$$

More generally, we claim that an effect occurs when there is an apparent mismatch between syntax and semantics

- Without effects: a unique logic for syntax and semantics
- With effects: a logic for the (apparent) syntax,
  another logic for the semantics,
  and a span of logics (= a zoom) relating them

# Notes

### About the authors

Our background lies in computer algebra: abstract algebra, algorithmic, programmation (exact, efficient, generic,...) in languages such as Axiom, C, C++,...

# Notes

## About the authors

Our background lies in computer algebra: abstract algebra, algorithmic, programmation (exact, efficient, generic,...) in languages such as Axiom, C, C++,...

## About terminology SPECIFICATION vs. THEORY

In this talk, a logical theory is "saturated": every theorem that can be deduced from the theory belongs to the theory. We call specification a family of axioms and theorems that may be non-saturated. A specification presents (= generates) a theory, and several different specifications may present the same theory.

# Notes

### About the authors

Our background lies in computer algebra: abstract algebra, algorithmic, programmation (exact, efficient, generic,...) in languages such as Axiom, C, C++,...

### About terminology SPECIFICATION vs. THEORY

In this talk, a logical theory is "saturated": every theorem that can be deduced from the theory belongs to the theory. We call specification a family of axioms and theorems that may be non-saturated. A specification presents (= generates) a theory, and several different specifications may present the same theory.

### About terminology SYNTAX vs. SEMANTICS

In this talk, the syntax may include some axioms (logical semantics) and the semantics is denotational

# Outline

# Imperative programming

In imperative programming the state of the memory may be observed (lookup) and modified (update)
However, the state never appears explicitly in the syntax: there no "type of states"

After *updating* a location $X$ to a value $n$:

- a *lookup* to $X$ returns $n$
- while a *lookup* to $Y$ $(\neq X)$ returns the same value as the *lookup* to $Y$ before updating $X$.

This can be written in a loose functorial style:

$$\begin{cases} lookup_X(update_X(n)) = n \\ lookup_Y(update_X(n)) = lookup_Y(\,) \end{cases}$$

This is now formalized, by defining three specifications

# The apparent specification

Notations: $Loc = \{X, Y, ...\}$ = the set of locations
$1$ for "Unit", with $(\,)_A : A \to 1$ for all $A$

From the syntax we get the apparent equational specification $\Sigma_1$:
for each location $i \in Loc$:

- a type $V_i$ for the values of $i$
- two functions:
$$\begin{cases} \text{lookup} & l_i : 1 \to V_i \\ \text{update} & u_i : V_i \to 1 \end{cases}$$
- equations:
$$\begin{cases} l_i \circ u_i = \mathtt{id}_{V_i} \\ l_j \circ u_i = l_j \circ (\,)_{V_i} \text{ for all } j \neq i \end{cases}$$

EFFECT: the intended semantics is not a model of $\Sigma_1$.

# The explicit specification

Let $S$ be the "type of states".

From the semantics we get the explicit equational specification $\Sigma_2$:
For each location $i \in Loc$:

- $V_i$
- $\begin{cases} l_i : S \to V_i \\ u_i : V_i \times S \to S \end{cases}$
- $\begin{cases} l_i \circ u_i = \mathtt{pr}_{V_i} \\ l_j \circ u_i = l_j \circ \mathtt{pr}_S \text{ for all } j \neq i \end{cases}$

EFFECT: the intended semantics is a model of $\Sigma_2$, *but*
$\Sigma_2$ does not fit with the syntax, because of the "type of states" $S$

# Decorations

Let us introduce decorations for functions:

- $(0)$ for pure functions
- $(1)$ for accessors $(=$ inspectors$)$
- $(2)$ for modifiers

AND for equations:

- $\sim$ for weak equations (equality on values only)
- $=$ for strong equations (equality on values and state)

# The decorated specification

With the decorations we form the decorated specification $\Sigma_0$:
for each location $i \in Loc$:

- $V_i$
- $\begin{cases} l_i^{(1)} : 1 \to V_i \\ u_i^{(2)} : V_i \to 1 \end{cases}$
- $\begin{cases} l_i^{(1)} \circ u_i^{(2)} \sim \mathtt{id}_{V_i}^{(0)} \\ l_j^{(1)} \circ u_i^{(2)} \sim l_j^{(1)} \circ (\,)_{V_i}^{(0)} \text{ for all } j \neq i \end{cases}$

Claim. *The decorated specification $\Sigma_0$ is "the most relevant":*

- both the apparent and the explicit specification may be recovered from $\Sigma_0$
- $\Sigma_0$ fits with the syntax (no type $S$)
- the intended semantics is a "decorated model" of $\Sigma_0$
- "decorated proofs" may be performed from $\Sigma_0$

# Three specifications

$$
\begin{array}{l}
l_i^{(1)} : 1 \longrightarrow V_i \\
u_i^{(2)} : V_i \longrightarrow 1 \\
l_i^{(1)} \circ u_i^{(2)} \sim \mathtt{id}_{V_i}^{(0)} \\
l_j^{(1)} \circ u_i^{(2)} \sim l_j^{(1)} \circ (\,)_{V_i}^{(0)} \\
\hline
\text{DECORATED}: \Sigma_0
\end{array}
$$

$F_1$

$F_2$

$$
\begin{array}{l}
l_i : 1 \longrightarrow V_i \\
u_i : V_i \longrightarrow 1 \\
l_i \circ u_i = \mathtt{id}_{V_i} \\
l_j \circ u_i = l_j \circ (\,)_{V_i} \\
\hline
\text{APPARENT}: \Sigma_1
\end{array}
$$

$$
\begin{array}{l}
l_i : S \longrightarrow V_i \\
u_i : V_i \times S \longrightarrow S \\
l_i \circ u_i = \mathtt{pr}_{V_i} \\
l_j \circ u_i = l_j \circ \mathtt{pr}_S \\
\hline
\text{EXPLICIT}: \Sigma_2
\end{array}
$$

- $F_1$: from decorated to apparent: wipe out all decorations
- $F_2$: from decorated to explicit: according to the decoration

# Three logics

Claim: the 3 specifications are defined in 3 “logics” related by a “span of logics”:

$$\begin{tikzcd}[column sep=small] & \text{DECORATED}: L_0 \arrow[dl, bend right, "F_1"'] \arrow[dr, bend left, "F_2"] & \\ \text{APPARENT}: L_1 & & \text{EXPLICIT}: L_2 \end{tikzcd}$$

- What is a logic?
- What is a morphism of logics?

# Outline

# Diagrammatic logic

A diagrammatic logic is a functor $L$ with a full and faithful right adjoint $R$

$$\begin{tikzcd} \mathbf{S} \arrow[rr, "L"] \arrow[rr, phantom, shift right=3, "\perp"] & & \mathbf{T} \arrow[ll, bend left, "R"] \end{tikzcd}$$

# Diagrammatic logic

A diagrammatic logic is a functor $L$
with a full and faithful right adjoint $R$

$$\begin{tikzcd} \mathbf{S} \arrow[r, "L"] \arrow[r, phantom, shift right=3, "\perp"] & \mathbf{T} \arrow[l, bend left, "R"] \end{tikzcd}$$

induced by a morphism of limit sketches
($\mathbf{S}$ and $\mathbf{T}$ are locally presentable categories)

$$\begin{tikzcd} \mathbf{E}_S \arrow[r, "e"] \arrow[d, "Y_S"'] & \mathbf{E}_T \arrow[d, "Y_T"'] \\ \mathbf{S} \arrow[r, "L"] \arrow[r, phantom, shift right=3, "\perp"] & \mathbf{T} \arrow[l, bend left, "R"] \end{tikzcd}$$

($Y =$ the contravariant Yoneda functor)

# Diagrammatic logic

A diagrammatic logic is a functor $L$
with a full and faithful right adjoint $R$

$$\begin{tikzcd} \mathbf{S} \arrow[r, "L"] \arrow[r, phantom, shift right=3, "\perp"] & \mathbf{T} \arrow[l, bend left, "R"] \end{tikzcd}$$

induced by a morphism of limit sketches
($\mathbf{S}$ and $\mathbf{T}$ are locally presentable categories)

$$\begin{tikzcd} \mathbf{E}_S \arrow[r, "e"] \arrow[d, "Y_S"'] & \mathbf{E}_T \arrow[d, "Y_T"'] \\ \mathbf{S} \arrow[r, "L"] \arrow[r, phantom, shift right=3, "\perp"] & \mathbf{T} \arrow[l, bend left, "R"] \end{tikzcd}$$

($Y =$ the contravariant Yoneda functor)

Ex. Monadic equational logic

- **S**: "linear" sketches (= graphs with some composition)
- **T**: categories

Ex. Equational logic

- **S**: finite product sketches
- **T**: categories with finite products

# Models

- $\mathbf{T}$: category of theories
- $\mathbf{S}$: category of specifications
- $\Sigma$ is a presentation of $L(\Sigma)$ for every specification $\Sigma$

  *$R$ full and faithful* $\iff$
  *$R(\Theta)$ is a presentation of $\Theta$ for every theory $\Theta$*
- a model $M$ of a specification $\Sigma$ with values in a theory $\Theta$ is a morphism $L\Sigma \to \Theta$ in $\mathbf{T}$, i.e., a morphism $\Sigma \to R\Theta$ in $\mathbf{S}$

$$\begin{tikzcd}
\mathbf{S} \arrow[rr, "L"] \arrow[rr, phantom, shift right=3, "\perp"] & & \mathbf{T} \arrow[ll, bend left, "R"] \\
\Sigma \arrow[u, phantom, sloped, "\in"] \arrow[rr, "M"'] & & \Theta \arrow[u, phantom, sloped, "\in"]
\end{tikzcd}$$

Ex. Monadic equational logic with $\Theta_{set}$ the category of sets

- $\Sigma_{nat}$: $N,\ z : 1 \to N,\ s : N \to N$
- $M_{nat} : \Sigma_{nat} \to \Theta_{set}$: $M(N) = \mathbb{N},\ M(z) = 0,\ M(s)(x) = x + 1$

# Inference rules

[Gabriel-Zisman 1967] *R is full and faithful* $\iff$ *(up to equiv.)*
*L is a localization: L makes some morphisms in* **S** *invertible in* **T**

- an entailment is a morphism $\tau$ in **S** with $L\tau$ invertible in **T**:
$$\Sigma \mathrel{\underset{\dashleftarrow}{\xrightarrow{\tau}}} \Sigma'$$
- an instance of $\Sigma_0$ in $\Sigma$ is a cospan in **S** with $\tau$ an entailment:
$$\Sigma_0 \xrightarrow{\sigma} \Sigma' \mathrel{\underset{\dashrightarrow}{\xleftarrow{\tau}}} \Sigma$$
- an inference rule with hypothesis $H$ and conclusion $C$ is an instance of $C$ in $H$:
$$H \mathrel{\underset{\dashleftarrow}{\longrightarrow}} H' \longleftarrow C$$

Ex. Substitution rule $\dfrac{f = g}{f \circ h = g \circ h}$

- $H$: $f, g, h,\ f = g$
- $H'$: $f, g, h,\ f = g,\ f \circ h = g \circ h$ with $H \mathrel{\underset{\dashleftarrow}{\rightarrow}} H'$ the inclusion
- $C$: $a, b,\ a = b$ with $C \longrightarrow H'$ such that $a \mapsto f \circ h,\ b \mapsto g \circ h$

# Proofs

- the inference step with respect to an inference rule $\rho : C \to H$ maps every instance $\iota : H \to \Sigma$ to the instance $\iota \circ \rho : C \to \Sigma$

  Composition holds in the bicategory of spans over **S**, which involves a pushout in **S**:

$$
\begin{tikzcd}
 & H \arrow[r, shift left] \arrow[d] \arrow[dr, phantom, "PO"] & H' \arrow[l, dashed, shift left] \arrow[d] & C \arrow[l] \arrow[dl] \arrow[l, phantom, shift right=3, "="] \\
\Sigma \arrow[r, shift left] & \Sigma' \arrow[l, dashed, shift left] \arrow[r, shift left] & \Sigma'' \arrow[l, dashed, shift left] &
\end{tikzcd}
$$

- an inference system is a morphism of limit sketches $e$ such that $L$ is induced by $e$
- a proof is a morphism in **T**, described in terms of a given inference system

# Languages

Given a logic and a theory:

$$\begin{tikzcd} \mathbf{S} \arrow[r, "L"] \arrow[r, phantom, shift right=3, "\perp"] & \mathbf{T} \arrow[l, bend left, "R"] \\ & \Theta \arrow[u, phantom, "\Cup"] \end{tikzcd} \qquad \text{denoted} \qquad \begin{matrix} L \\ \Cup \\ \Theta \end{matrix}$$

in this talk, a language is:

- a syntax $\Sigma$: a specification $\Sigma$ in $\mathbf{S}$
- a semantics $M : \Sigma \to \Theta$: a model $M$ of $\Sigma$ with values in $\Theta$

$$\begin{tikzcd} \mathbf{S} \arrow[r, "L"] \arrow[r, phantom, shift right=3, "\perp"] & \mathbf{T} \arrow[l, bend left, "R"] \\ \Sigma \arrow[u, phantom, "\Cup"] \arrow[r, "M"'] & \Theta \arrow[u, phantom, "\Cup"] \end{tikzcd} \qquad \text{denoted} \qquad \begin{matrix} L \\ \Cup \\ \Sigma \xrightarrow{M} \Theta \end{matrix}$$

# Languages

Given a logic and a theory:

$$\begin{tikzcd} \mathbf{S} \arrow[r, "L"] \arrow[r, phantom, shift right=3, "\perp"] & \mathbf{T} \arrow[l, bend left, "R"] \\ & \Theta \arrow[u, phantom, sloped, "\Subset"] \end{tikzcd} \quad \text{denoted} \quad \begin{array}{c} L \\ \Subset \\ \Theta \end{array}$$

in this talk, a language is:

- a syntax $\Sigma$: a specification $\Sigma$ in $\mathbf{S}$
- a semantics $M : \Sigma \to \Theta$: a model $M$ of $\Sigma$ with values in $\Theta$

$$\begin{tikzcd} \mathbf{S} \arrow[r, "L"] \arrow[r, phantom, shift right=3, "\perp"] & \mathbf{T} \arrow[l, bend left, "R"] \\ \Sigma \arrow[u, phantom, sloped, "\Subset"] \arrow[r, "M"'] & \Theta \arrow[u, phantom, sloped, "\Subset"] \end{tikzcd} \quad \text{denoted} \quad \begin{array}{c} L \\ \Subset \\ \Sigma \xrightarrow{M} \Theta \end{array}$$

**Note.**

- this syntax may include some axioms (logical semantics)
- this semantics is denotational

# Morphisms of logics

(Based on arrow categories.) A morphism $F : L_1 \to L_2$ is a pair of left adjoint functors $(F_S, F_T)$ such that $L_2 \circ F_S \cong F_T \circ L_1$

$$
\begin{tikzcd}[column sep=huge, row sep=huge]
\mathbf{S}_1 \arrow[r, "L_1"] \arrow[r, phantom, shift right=2, "\perp"] \arrow[d, "F_S"'] \arrow[d, phantom, shift left=2, "\dashv"] & \mathbf{T}_1 \arrow[l, bend left, "R_1"] \arrow[d, "F_T"'] \arrow[d, phantom, shift left=2, "\dashv"] \\
\mathbf{S}_2 \arrow[u, bend right, "G_S"'] \arrow[r, "L_2"] \arrow[r, phantom, shift right=2, "\perp"] & \mathbf{T}_2 \arrow[u, bend right, "G_T"'] \arrow[l, bend left, "R_2"]
\end{tikzcd}
$$

induced by a commutative square of limit sketches.

This provides the category of diagrammatic logics

# Languages with effects

Given a span of logics (a zoom) and a theory $\Theta_2$ in $L_2$

$$
\begin{tikzcd}
& \text{DECORATED} \\
& L_0 \arrow[dl, "F_1"'] \arrow[dr, "F_2"] & \\
\text{APPARENT} \\ L_1 & & \text{EXPLICIT} \\ L_2 \\ \Uplus \\ \Theta_2
\end{tikzcd}
$$

a language with effects is a language (without effects!)
with respect to the decorated logic $L_0$ and the theory $\Theta_0 = G_2\Theta_2$

$$
\begin{tikzcd}
& \begin{array}{c} L_0 \\ \Uplus \\ \Sigma_0 \xrightarrow{M} \Theta_0 \end{array} \arrow[dl, "F_1"'] \arrow[dr, "F_2"] & \\
L_1 & & L_2
\end{tikzcd}
$$

# Effect as mismatch between apparent syntax and semantics

From the decorated syntax $\Sigma_0$ we get
an explicit syntax $\Sigma_2$ in $L_2$
and an apparent syntax $\Sigma_1$ in $L_1$

$$\begin{tikzcd}
 & L_0 \arrow[dl, "F_1"'] \arrow[dr, "F_2"] & \\
L_1 & \Cup & L_2 \\
\Cup & \Sigma_0 & \Cup \\
\Sigma_1 = F_1\Sigma_0 & & \Sigma_2 = F_2\Sigma_0
\end{tikzcd}$$

The decorated semantics $M : \Sigma_0 \to \Theta_0$ is “equivalent” to
the explicit semantics $M : \Sigma_2 \to \Theta_2$ provided by the adjunction
but there is NO “equivalent” apparent semantics

$$\begin{tikzcd}
 & L_0 \arrow[dl, "F_1"'] \arrow[dr, "F_2"] & \\
L_1 & \Cup & L_2 \\
\Cup & \Sigma_0 \xrightarrow{M} \Theta_0 & \Cup \\
\Sigma_1 \overset{M}{\not\to} ?? & & \Sigma_2 \xrightarrow{M} \Theta_2
\end{tikzcd}$$

# A zoom for states

$$
\begin{array}{ccc}
 & \text{DECORATED: } L_0 & \\
\swarrow^{F_1} & & \searrow^{F_2} \\
\text{APPARENT: } L_1 & & \text{EXPLICIT: } L_2
\end{array}
$$

- $L_1$ is the monadic equational logic: a theory of $L_1$ is a category
- a theory of $L_2$ is a category with a distinguished object $S$ and with a functor $- \times S$
- a theory of $L_0$ is made of three embedded categories with the same objects $\mathbf{C}^{(0)} \subseteq \mathbf{C}^{(1)} \subseteq \mathbf{C}^{(2)}$, with 1,...
- $F_1$ omits the decorations: it maps $\mathbf{C}^{(0)} \subseteq \mathbf{C}^{(1)} \subseteq \mathbf{C}^{(2)}$ to $\mathbf{C}^{(2)}$
- $F_2$ provides the meaning of the decorations, it can be described "pointwise" since it preserves colimits (next slide)

# Expansion of decoration, for states

the expansion functor $F_2$ provides the meaning of the decorations

$$L_0 \xrightarrow{F_2} L_2$$

$$\boxed{X \xrightarrow{f^{(0)}} Y} \overset{F_2}{\longmapsto} \boxed{X \xrightarrow{f} Y}$$

$$\boxed{X \xrightarrow{g^{(1)}} Y} \overset{F_2}{\longmapsto} \boxed{X \times S \xrightarrow{g} Y}$$

$$\boxed{X \xrightarrow{h^{(2)}} Y} \overset{F_2}{\longmapsto} \boxed{X \times S \xrightarrow{h} Y \times S}$$

$$\boxed{\begin{array}{c} X \overset{h^{(2)}}{\underset{k^{(2)}}{\rightrightarrows}} Y \\ h^{(2)} \sim k^{(2)} \end{array}} \overset{F_2}{\longmapsto} \boxed{\begin{array}{c} X \times S \overset{h}{\underset{k}{\rightrightarrows}} Y \times S \\ \mathtt{pr}_Y \circ h = \mathtt{pr}_Y \circ k \end{array}}$$

# Outline

# Exceptions as dual of states?

Monads:

| | |
|---|---|
| states | $T(X) = (X \times St)^{St}$ |
| exceptions | $T(X) = X + Exc$ |

# Exceptions as dual of states?

Monads:

| | |
|---|---|
| states | $T(X) = (X \times St)^{St}$ |
| exceptions | $T(X) = X + Exc$ |

Lawvere theories:

| | |
|---|---|
| states | $lookup : Val \to Loc$<br>$update : 1 \to Loc \times Val$<br>with 7 equations |
| exceptions | $raise_e : 0 \to 1$ for $e \in Exc$<br>with no equation |

# Exceptions as dual of states!

When effects are described by zooms there is a duality which provides a new point of view on exceptions

- States involve the functor $X \times S$
  for some distinguished "type of states" $S$
- Exceptions involve the functor $X + E$
  for some distinguished "type of exceptions" $E$

The well-known duality between $X \times S$ and $X + E$ extends as a duality between states and exceptions

| | |
|---|---|
| states | $l_i^{(1)} : 1 \to V_i$<br>$u_i^{(2)} : V_i \to 1$<br>with 2 equations |
| exceptions | $r_i^{(1)} : P_i \to 0$<br>$h_i^{(2)} : 0 \to P_i$<br>with 2 equations |

# Decorations for exceptions

The same decorations for exceptions as for states,
with different meaning

The meaning of decorations for functions:

- $(0)$ for pure functions
- $(1)$ for propagators
- $(2)$ for handlers

and for equations:

- $\sim$ for weak equations (equality on non-exceptional arguments)
- $=$ for strong equations (equality on all arguments)

# Expansion of decoration, for exceptions

The expansion functor $F_2$ provides the meaning of the decorations
it is dual to the expansion functor $F_2$ for states

$$L_0 \xrightarrow{F_2} L_2$$

$$\boxed{X \xrightarrow{f^{(0)}} Y} \xmapsto{F_2} \boxed{X \xrightarrow{f} Y}$$

$$\boxed{X \xrightarrow{g^{(1)}} Y} \xmapsto{F_2} \boxed{X \xrightarrow{g} Y + E}$$

$$\boxed{X \xrightarrow{h^{(2)}} Y} \xmapsto{F_2} \boxed{X + E \xrightarrow{h} Y + E}$$

$$\boxed{\begin{tikzcd} X \arrow[r, bend left, "h^{(2)}"] \arrow[r, bend right, "k^{(2)}"'] & Y \end{tikzcd} \quad h^{(2)} \sim k^{(2)}} \xmapsto{F_2} \boxed{\begin{tikzcd} X + E \arrow[r, bend left, "h"] \arrow[r, bend right, "k"'] & Y + E \end{tikzcd} \quad h \circ \mathtt{inj}_X = k \circ \mathtt{inj}_X}$$

# Exceptions as dual of states (informally)

- States (in a pointer-free language)

$$\begin{cases} l_i \circ u_i = \mathtt{id}_{V_i} \\ l_j \circ u_i = l_j \circ (\,)_{V_i} \end{cases}$$

In order to lookup the value of a location, only the **previous** updating of *this* location is required, everything that has been executed since this previous updating is irrelevant

- Exceptions

$$\begin{cases} h_i \circ r_i = \mathtt{id}_{P_i} \\ h_i \circ r_j = [\,]_{P_i} \circ r_j \end{cases}$$

When some exception is raised, the **following** handler for *this* type of exceptions is immediately executed, everything that is written until this following handler is irrelevant

# Exceptions as dual of states (formally)

Notations: *Etype* is the set of exceptional types

for each $i \in \mathit{Etype}$ a type $P_i$ for the parameters of type $i$

0 for "Empty", with $[\,]_A : 0 \to A$ for all $A$

$E$ is the "type of exceptions"

Three specifications

$$\boxed{\begin{array}{l} r_i^{(1)} : P_i \to 0 \\ h_i^{(2)} : 0 \to P_i \\ h_i^{(2)} \circ r_i^{(1)} = \mathtt{id}_{P_i}^{(0)} \\ h_i^{(2)} \circ r_j^{(1)} = [\,]_{P_i}^{(0)} \circ r_j^{(1)} \\ \hline \text{DECORATED: } \Sigma_0 \end{array}}$$

$$\Sigma_1 \xleftarrow{F_1} \Sigma_0 \xrightarrow{F_2} \Sigma_2$$

$$\boxed{\begin{array}{l} r_i : P_i \to 0 \\ h_i : 0 \to P_i \\ h_i \circ r_i = \mathtt{id}_{P_i} \\ h_i \circ r_j = [\,]_{P_i} \circ r_j \\ \hline \text{APPARENT: } \Sigma_1 \end{array}} \qquad \boxed{\begin{array}{l} r_i : P_i \to E \\ h_i : E \to P_i + E \\ h_i \circ r_i = \mathtt{inj}_{P_i} \\ h_i \circ r_j = \mathtt{inj}_E \circ r_j \\ \hline \text{EXPLICIT: } \Sigma_2 \end{array}}$$

# Exceptions: encapsulation of $r_i$

Claim.

- $r_i^{(1)}$ and $h_i^{(2)}$ are the core operations for raising and handling exceptions of type $i$
- they are encapsulated inside operations $raise_{i,X}^{(1)}$ and $handle_{i,f,g}^{(1)}$ which are expanded as the usual operations *raise* and *handle*

In raising an exception, the empty type is hidden

$$raise_{i,X}^{(1)} = [\,]_X^{(0)} \circ r_i^{(1)} : P_i \to X$$

$$\begin{tikzcd}
P_i \arrow[r, "r_i^{(1)}"'] \arrow[rr, bend left, "raise_{i,X}^{(1)}"] & 0 \arrow[r, "{[\,]^{(0)}}"'] & X
\end{tikzcd}$$

first $r_i$ raises an exception of exceptional type $i$
then $[\,]_X$ converts this exception to type $X$

# Exceptions: encapsulation of $h_i$

The handling of exceptions is a powerful programming technique, carefully encapsulated thanks to the property:

$$\forall\ f^{(2)} : X \to Y \ \ \exists!\ \lfloor f \rfloor^{(1)} : X \to Y \ \ f^{(2)} \sim \lfloor f \rfloor^{(1)}$$

explicitly:

$$\forall\ f : X + E \to Y + E \ \ \exists!\ \lfloor f \rfloor : X + E \to Y + E$$

such that $\lfloor f \rfloor = f$ on $X$ and $\lfloor f \rfloor$ propagates exceptions.

For all $f^{(1)} : X \to Y$ and $g^{(1)} : P_i \to Y$
the handling by $g^{(1)}$ of an exception of type $i$ raised in $f^{(1)}$ is

$$handle^{(1)}_{i,f,g} = \lfloor [\mathtt{id}_Y | g \circ h_i]^{(2)}_Y \circ f^{(1)} \rfloor^{(1)} : X \to Y$$

$$\begin{tikzcd}
X \arrow[r, "f^{(1)}"] \arrow[rrr, bend left, "handle^{(1)}_{i,f,g}"] & Y \arrow[rr, "\mathtt{id}^{(0)}"] & & Y \\
& 0 \arrow[u, "{[\,]^{(0)}}"] \arrow[r, "h_i^{(2)}"] & P_i \arrow[ur, bend right, "g^{(1)}"'] &
\end{tikzcd}$$

# Some details

The handling by $g^{(1)}$ of an exception of type $i$ raised in $f^{(1)}$ is

$$handle^{(1)}_{i,f,g} = \lfloor [\mathtt{id}_Y | g \circ h_i]^{(2)}_Y \circ f^{(1)} \rfloor^{(1)} : X \to Y$$

which means: first build $[\mathtt{id}_Y | g \circ h_i]^{(2)} \circ f^{(1)}$

$$\begin{tikzcd}
 & Y \arrow[d, "\mathtt{id}^{(0)}"'] \arrow[rr, bend left, "\mathtt{id}^{(0)}"] \arrow[drr, phantom, "\sim"] & & \\
X \arrow[r, "f^{(1)}"] & Y \arrow[rr, "{[\mathtt{id}|g \circ h_i]^{(2)}}"] \arrow[drr, phantom, "="] & & Y \\
 & 0 \arrow[u, "{[\,]^{(0)}}"] \arrow[r, "h_i^{(2)}"] & P_i \arrow[ur, bend right, "g^{(1)}"'] &
\end{tikzcd}$$

then encapsulate $[\mathtt{id}_Y | g \circ h_i]^{(2)}_Y \circ f^{(1)}$

$$\begin{tikzcd}
X \arrow[r, "f^{(1)}"] \arrow[rr, bend left, "handle^{(1)}_{i,f,g}"] & Y \arrow[r, "{[\mathtt{id}|g \circ h_i]^{(2)}}"] \arrow[phantom, u, "\sim"] & Y
\end{tikzcd}$$

# Duality, backwards

The rule for decorated sums builds $[f_1^{(0)}|f_2^{(2)}]^{(2)}$

$$\boxed{\begin{tikzcd}
X_1 \arrow[d] \arrow[rr, bend left, "f_1^{(0)}"] & & \\
X_1 + X_2 & & Y \\
X_2 \arrow[u] \arrow[rr, bend right, "f_2^{(2)}"'] & &
\end{tikzcd}}
\quad
\begin{tikzcd}
{} \arrow[r, shift left] & {} \arrow[l, dashed, shift left]
\end{tikzcd}
\quad
\boxed{\begin{tikzcd}
X_1 \arrow[d] \arrow[rr, bend left, "f_1^{(0)}"] & \sim & \\
X_1 + X_2 \arrow[rr] & & Y \\
X_2 \arrow[u] \arrow[rr, bend right, "f_2^{(2)}"'] & = &
\end{tikzcd}}$$

# Duality, backwards

The rule for decorated sums builds $[f_1^{(0)} | f_2^{(2)}]^{(2)}$

$$\begin{tikzcd}
X_1 \arrow[d] \arrow[drr, bend left, "f_1^{(0)}"] & & \\
X_1 + X_2 & & Y \\
X_2 \arrow[u] \arrow[urr, bend right, "f_2^{(2)}"'] & &
\end{tikzcd}
\quad
\begin{tikzcd}
\arrow[r, shift left] & \arrow[l, dashed, shift left]
\end{tikzcd}
\quad
\begin{tikzcd}
X_1 \arrow[d] \arrow[drr, bend left, "f_1^{(0)}"] \arrow[drr, phantom, "\sim"] & & \\
X_1 + X_2 \arrow[rr] & & Y \\
X_2 \arrow[u] \arrow[urr, bend right, "f_2^{(2)}"'] \arrow[urr, phantom, "="] & &
\end{tikzcd}$$

Dually, the rule for decorated products builds $(f_1^{(0)}, f_2^{(2)})^{(2)}$

$$\begin{tikzcd}
 & & Y_1 \\
X \arrow[urr, bend left, "f_1^{(0)}"] \arrow[drr, bend right, "f_2^{(2)}"'] & & Y_1 \times Y_2 \arrow[u] \arrow[d] \\
 & & Y_2
\end{tikzcd}
\quad
\begin{tikzcd}
\arrow[r, shift left] & \arrow[l, dashed, shift left]
\end{tikzcd}
\quad
\begin{tikzcd}
 & & Y_1 \\
X \arrow[urr, bend left, "f_1^{(0)}"] \arrow[urr, phantom, "\sim"] \arrow[rr] \arrow[drr, bend right, "f_2^{(2)}"'] \arrow[drr, phantom, "="] & & Y_1 \times Y_2 \arrow[u] \arrow[d] \\
 & & Y_2
\end{tikzcd}$$

This rule is the key for dealing with multivariate functions when there are effects (an alternative to the strength of the monad)

# Outline

This talk.

- effect as an apparent mismatch between syntax and semantics
- the category of diagrammatic logics
- zooms (= spans of logics) for effects
- a new point of view on states
- a completely new point of view on exceptions with handling
- a duality between states and exceptions

This talk.

- effect as an apparent mismatch between syntax and semantics
- the category of diagrammatic logics
- zooms (= spans of logics) for effects
- a new point of view on states
- a completely new point of view on exceptions with handling
- a duality between states and exceptions

Future work.

- other effects
- combining effects
- operational semantics

# Some papers


- J.-G. Dumas, D. Duval, L. Fousse, J.-C. Reynaud.
  *States and exceptions are dual effects.*
  arXiv:1001.1662 (2010).
- J.-G. Dumas, D. Duval, J.-C. Reynaud.
  *Cartesian effect categories are Freyd-categories.*
  JSC (2010).
- C. Dominguez, D. Duval.
  *Diagrammatic logic applied to a parameterization process.*
  MSCS 20(04) p. 639-654 (2010).
- D. Duval, J.-C. Reynaud.
  *Dynamic logic and exceptions: an introduction.*
  Mathematics, Algorithms, Proofs. Dagstuhl Seminar 05021 (2005).
- D. Duval.
  *Diagrammatic Specifications.*
  MSCS (13) 857-890 (2003).