

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

Introduction

States

Diagrammatic logics

Exceptions

Conclusion

Semantics of computational effects?

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

logic	programming	categories
<i>propositions</i>	<i>types</i>	<i>objects</i>
<i>proofs</i>	<i>terms</i>	<i>morphisms</i>
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
<i>effect</i>	<i>categorical structure ??</i>
(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 $T(B)$.

$p : A \rightarrow B$ is interpreted as $p : A \rightarrow T(B)$

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

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

effect	monad (T, η, μ)
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	$lookup : Val \rightarrow Loc$ $update : 1 \rightarrow Loc \times Val$ with 7 equations
exceptions	$raise_e : 0 \rightarrow 1$ for $e \in Exc$ with no equation

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

Effects as zooms (= spans of logics)

Following Moggi's remark:

$p : A \rightarrow B$ is interpreted as $p : A \rightarrow T(B)$

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, programming (exact, efficient, generic,...) in languages such as Axiom, C, C++,...

Notes

About the authors

Our background lies in **computer algebra**: abstract algebra, algorithmic, programming (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, programming (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

Introduction

States

Diagrammatic logics

Exceptions

Conclusion

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:

$$\begin{cases} \text{lookup}_X(\text{update}_X(n)) = n \\ \text{lookup}_Y(\text{update}_X(n)) = \text{lookup}_Y() \end{cases}$$

This is now formalized, by defining **three specifications**

The apparent specification

Notations: $Loc = \{X, Y, \dots\}$ = the set of **locations**
1 for “Unit”, with $()_A : A \rightarrow 1$ for all A

From the **syntax** we get the **apparent** equational specification Σ_1 :
for each location $i \in Loc$:

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

EFFECT: the intended semantics **is not** a model of Σ_1 .

The explicit specification

Let S be the “type of **states**”.

From the **semantics** we get the **explicit** equational specification Σ_2 :

For each location $i \in Loc$:

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

EFFECT: the intended semantics **is** a model of Σ_2 , *but*

Σ_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 Σ_0 :
for each location $i \in \text{Loc}$:

- ▶ V_i
- ▶ $\left\{ \begin{array}{l} l_i^{(1)} : 1 \rightarrow V_i \\ u_i^{(2)} : V_i \rightarrow 1 \end{array} \right.$
- ▶ $\left\{ \begin{array}{l} l_i^{(1)} \circ u_i^{(2)} \sim \text{id}_{V_i}^{(0)} \\ l_j^{(1)} \circ u_i^{(2)} \sim l_j^{(1)} \circ ()_{V_i}^{(0)} \text{ for all } j \neq i \end{array} \right.$

Claim. The *decorated* specification Σ_0 is “the most relevant”:

- ▶ both the apparent and the explicit specification may be recovered from Σ_0
- ▶ Σ_0 fits with the syntax (no type S)
- ▶ the intended semantics is a “decorated model” of Σ_0
- ▶ “decorated proofs” may be performed from Σ_0

Three specifications



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



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

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

- 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”:



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

Outline

Introduction

States

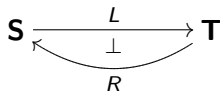
Diagrammatic logics

Exceptions

Conclusion

Diagrammatic logic

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



Diagrammatic logic

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

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

(Y = the contravariant Yoneda functor)

$$\begin{array}{ccc} \mathbf{S} & \xrightarrow{L} & \mathbf{T} \\ & \perp & \\ & \text{---} R \text{---} & \end{array}$$

$$\begin{array}{ccc} \mathbf{E}_S & \xrightarrow{e} & \mathbf{E}_T \\ Y_S \downarrow \parallel & & Y_T \downarrow \parallel \\ \mathbf{S} & \xrightarrow{L} & \mathbf{T} \\ & \perp & \\ & \text{---} R \text{---} & \end{array}$$

Diagrammatic logic

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

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

(Y = the contravariant Yoneda functor)

$$\begin{array}{ccc} \mathbf{S} & \xrightarrow{L} & \mathbf{T} \\ & \perp & \\ & \text{---} & \\ & R & \end{array}$$

$$\begin{array}{ccc} \mathbf{E}_S & \xrightarrow{e} & \mathbf{E}_T \\ Y_S \downarrow // & & Y_T \downarrow // \\ \mathbf{S} & \xrightarrow{L} & \mathbf{T} \\ & \perp & \\ & \text{---} & \\ & R & \end{array}$$

Ex. Monadic equational logic

- ▶ \mathbf{S} : “linear” sketches (= graphs with some composition)
- ▶ \mathbf{T} : categories

Ex. Equational logic

- ▶ \mathbf{S} : finite product sketches
- ▶ \mathbf{T} : categories with finite products

Models

- ▶ **T**: category of **theories**
- ▶ **S**: category of **specifications**
- ▶ Σ is a **presentation** of $L(\Sigma)$ for every specification Σ
*R full and faithful \iff
 $R(\Theta)$ is a presentation of Θ for every theory Θ*
- ▶ a **model** M of a specification Σ with values in a theory Θ is a morphism $L\Sigma \rightarrow \Theta$ in **T**, i.e., a morphism $\Sigma \rightarrow R\Theta$ in **S**

$$\begin{array}{ccc} \mathbf{S} & \xrightleftharpoons[L]{L} & \mathbf{T} \\ \Psi \swarrow & \perp & \searrow \Psi \\ & R & \\ \Sigma & \xrightarrow{M} & \Theta \end{array}$$

Ex. Monadic equational logic with Θ_{set} the category of sets

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

Inference rules

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

- ▶ an **entailment** is a morphism τ in \mathbf{S} with $L\tau$ invertible in \mathbf{T} :

$$\Sigma \xleftarrow{\tau} \Sigma'$$

- ▶ an **instance** of Σ_0 in Σ is a cospan in \mathbf{S} with τ an entailment:

$$\Sigma_0 \xrightarrow{\sigma} \Sigma' \xleftarrow{\tau} \Sigma$$

- ▶ an **inference rule** with **hypothesis** H and **conclusion** C is an instance of C in H :

$$H \xleftarrow{\quad} H' \longleftarrow C$$

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

- ▶ $H: f, g, h, f = g$
- ▶ $H': f, g, h, f = g, f \circ h = g \circ h$ with $H \xleftarrow{\quad} H'$ the inclusion
- ▶ $C: a, b, a = b$ with $C \rightarrow H'$ such that $a \mapsto f \circ h, b \mapsto g \circ h$

Proofs

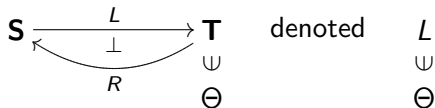
- ▶ the **inference step** with respect to an inference rule $\rho : C \rightarrow H$ maps every instance $\iota : H \rightarrow \Sigma$ to the instance $\iota \circ \rho : C \rightarrow \Sigma$
- Composition holds in the bicategory of spans over **S**, which involves a pushout in **S**:

$$\begin{array}{ccccc} H & \xrightarrow{\quad} & H' & \xleftarrow{\quad} & C \\ \downarrow & \swarrow PO & \downarrow & \searrow = & \\ \Sigma & \xrightarrow{\quad} & \Sigma' & \xrightarrow{\quad} & \Sigma'' \end{array}$$

- ▶ 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

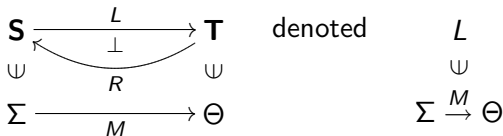
Languages

Given a logic and a theory:



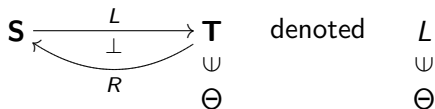
in this talk, a **language** is:

- ▶ a **syntax** Σ : a specification Σ in \mathbf{S}
- ▶ a **semantics** $M : \Sigma \rightarrow \Theta$: a model M of Σ with values in Θ



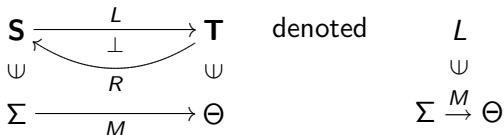
Languages

Given a logic and a theory:



in this talk, a **language** is:

- ▶ a **syntax** Σ : a specification Σ in \mathbf{S}
- ▶ a **semantics** $M : \Sigma \rightarrow \Theta$: a model M of Σ with values in Θ

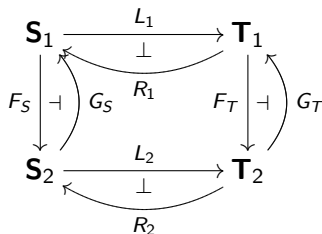


Note.

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

Morphisms of logics

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

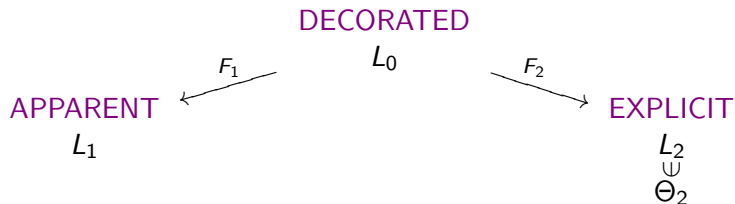


induced by a commutative square of limit sketches.

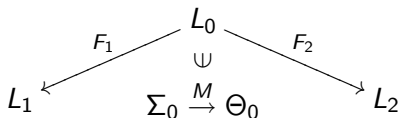
This provides the **category of diagrammatic logics**

Languages with effects

Given a span of logics (a zoom) and a theory Θ_2 in L_2

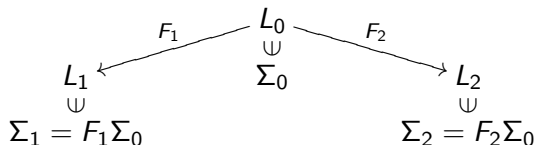


a **language with effects** is a **language** (without effects!)
with respect to the **decorated** logic L_0 and the theory $\Theta_0 = G_2\Theta_2$

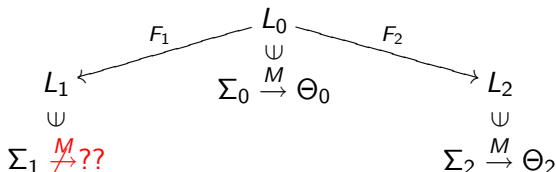


Effect as mismatch between apparent syntax and semantics

From the decorated syntax Σ_0 we get
an explicit syntax Σ_2 in L_2
and an apparent syntax Σ_1 in L_1



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



A zoom for states



- ▶ 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 $1, \dots$
- ▶ 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

$$L_0 \xrightarrow{F_2} L_2$$

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

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

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

$$\boxed{\begin{array}{c} X \xrightleftharpoons[k^{(2)}]{h^{(2)}} Y \\ h^{(2)} \sim k^{(2)} \end{array}} \xrightarrow{F_2} \boxed{\begin{array}{c} X \times S \xrightleftharpoons[k]{h} Y \times S \\ \text{pr}_Y \circ h = \text{pr}_Y \circ k \end{array}}$$

Outline

Introduction

States

Diagrammatic logics

Exceptions

Conclusion

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	$lookup : Val \rightarrow Loc$ $update : 1 \rightarrow Loc \times Val$ with 7 equations
exceptions	$raise_e : 0 \rightarrow 1$ for $e \in Exc$ 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	$l_i^{(1)} : 1 \rightarrow V_i$ $u_i^{(2)} : V_i \rightarrow 1$ with 2 equations
exceptions	$r_i^{(1)} : P_i \rightarrow 0$ $h_i^{(2)} : 0 \rightarrow P_i$ 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

$$L_0 \xrightarrow{F_2} L_2$$

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

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

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

$$\boxed{\begin{array}{c} X \xrightleftharpoons[k^{(2)}]{h^{(2)}} Y \\ h^{(2)} \sim k^{(2)} \end{array}} \xrightarrow{F_2} \boxed{\begin{array}{c} X + E \xrightleftharpoons[k]{h} Y + E \\ h \circ \text{inj}_X = k \circ \text{inj}_X \end{array}}$$

Exceptions as dual of states (informally)

- **States** (in a pointer-free language)

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

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**

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

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 $i \in Etype$ a type P_i for the **parameters** of type i

0 for “Empty”, with $[]_A : 0 \rightarrow A$ for all A

E is the “type of **exceptions**”

Three specifications



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



$$\begin{array}{l} r_i : P_i \rightarrow 0 \\ h_i : 0 \rightarrow P_i \\ h_i \circ r_i = \text{id}_{P_i} \\ h_i \circ r_j = []_{P_i} \circ r_j \end{array} \quad \hline \text{APPARENT: } \Sigma_1$$

$$\begin{array}{l} r_i : P_i \rightarrow E \\ h_i : E \rightarrow P_i + E \\ h_i \circ r_i = \text{inj}_{P_i} \\ h_i \circ r_j = \text{inj}_E \circ r_j \end{array} \quad \hline \text{EXPLICIT: } \Sigma_2$$

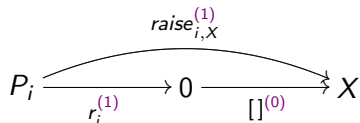
Exceptions: encapsulation of r_i

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

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



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:

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

explicitly:

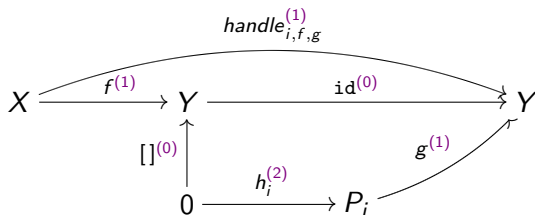
$$\forall f : X + E \rightarrow Y + E \quad \exists! \lfloor f \rfloor : X + E \rightarrow Y + E$$

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

For all $f^{(1)} : X \rightarrow Y$ and $g^{(1)} : P_i \rightarrow Y$

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

$$\text{handle}_{i,f,g}^{(1)} = \llbracket \text{id}_Y \mid g \circ h_i \rrbracket_Y^{(2)} \circ f^{(1)} \rfloor^{(1)} : X \rightarrow Y$$

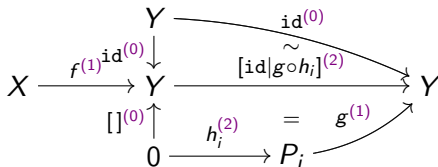


Some details

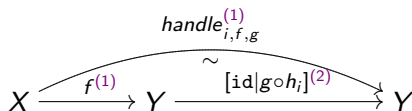
The handling by $g^{(1)}$ of an exception of type i raised in $f^{(1)}$ is

$$\text{handle}_{i,f,g}^{(1)} = \llbracket [\text{id}_Y | g \circ h_i]_Y^{(2)} \circ f^{(1)} \rrbracket^{(1)} : X \rightarrow Y$$

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

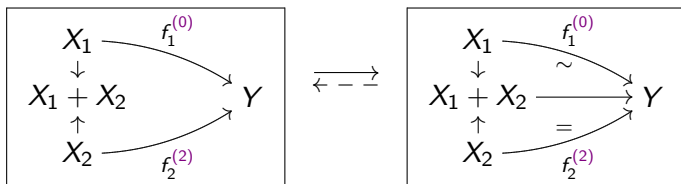


then encapsulate $[\text{id}_Y | g \circ h_i]^{(2)} \circ f^{(1)}$



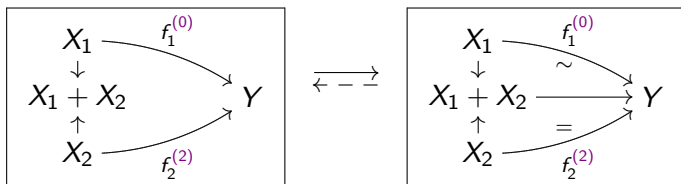
Duality, backwards

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

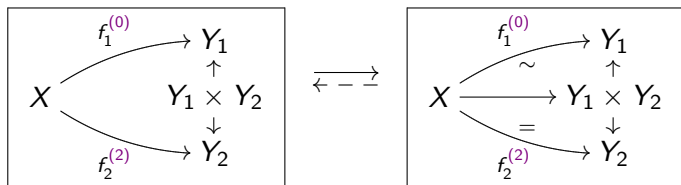


Duality, backwards

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



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



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

Outline

Introduction

States

Diagrammatic logics

Exceptions

Conclusion

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).