

# States and exceptions considered as dual effects

Dominique Duval  
with J.-G. Dumas, L. Fousse, J.-C. Reynaud

LJK, University of Grenoble

*March 27, 2011 – ACCAT 2011*  
*dedicated to the memory of Prof. Jochen Pfalzgraf*

# Outline

## Introduction

1. The duality, explicitly
2. About effects
3. The duality, “effect”-ively

## Conclusion

# Outline

1. Analyzing the semantics of exceptions yields a symmetry between states and exceptions  
at the semantics level.
2. States and exceptions are computational effects, but  
what is an effect?
3. Analyzing the syntax of exceptions as effects yields a symmetry between states and exceptions  
as computational effects.

# Outline

## Introduction

1. The duality, explicitly
2. About effects
3. The duality, “effect”-ively

## Conclusion

# Exceptions

When dealing with exceptions, there are two kinds of values:

- ▶ non-exceptional values
- ▶ exceptions

A function:

- ▶ **throws** an exception if it may  
map a non-exceptional value to an exception
- ▶ **catches** an exception if it may  
map an exception to a non-exceptional value

# Exceptions: key operations

$Exc$  = set of **exceptions**

$ExcStr$  = set of **exception constructors**

For each  $i \in ExcStr$ :

- ▶  $Par_i$  = set of **parameters**
- ▶  $t_i : Par_i \rightarrow Exc$  = **KEY throw** function
- ▶  $c_i : Exc \rightarrow Par_i + Exc$  = **KEY catch** function

$$\forall p \in Par_i \quad \begin{cases} c_i(t_i(p)) = p \in Par_i \subseteq Par_i + Exc \\ c_i(t_j(p)) = t_j(p) \in Exc \subseteq Par_i + Exc \quad (\forall j \neq i) \end{cases}$$

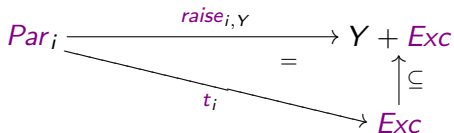
- $c_i$  **catches** exceptions of constructor  $i$
- $c_i$  **propagates** exceptions of constructor  $j \neq i$

When  $Exc = \sum_i Par_i$  with the key-throws as projections this is an inductive definition of the key-catches

# Exceptions: raise

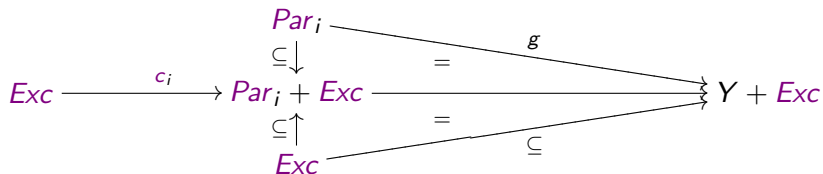
- From **key throwing** ( $t_i$ ) to **raising** ( $raise_{i,Y}$ ):

$$raise_{i,Y}(a) = t_i(a) \in Y + Exc$$

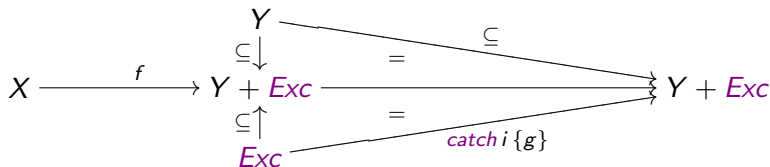


# Exceptions: handle

- From **key catching** ( $c_i$ ) to **catching** ( $catch\ i\ \{g\}$ ):



- From **catching** ( $catch\ i\ \{g\}$ ) to **handling** ( $f\ handle\ i\ \Rightarrow\ g$ ):





# States

$St$  = set of **states**

$Loc$  = set of **locations**

For each  $i \in Loc$ :

- ▶  $Val_i$  = set of **values**
- ▶  $l_i : St \rightarrow Val_i$  = **lookup** function
- ▶  $u_i : Val_i \times St \rightarrow St$  = **update** function

$$\forall v_i \in Val_i \quad \forall s \in St \quad \begin{cases} l_i(u_i(v_i, s)) = v_i \\ l_j(u_i(v_i, s)) = l_j(s) \quad (\forall j \neq i) \end{cases}$$

When  $St = \prod_i Val_i$  with the lookups as projections

this is a coinductive definition of the updates

# Duality of semantics

| States  | Exceptions  |
|---|---|
| $i \in Loc, Val_i$<br>$St (= \prod_{i \in Loc} Val_i)$  | $i \in ExCstr, Par_i$<br>$Exc (= \sum_{i \in ExCstr} Par_i)$  |
| $l_i : St \rightarrow Val_i$<br>$u_i : Val_i \times St \rightarrow St$  | $Exc \leftarrow Par_i : t_i$<br>$Par_i + Exc \leftarrow Exc : c_i$  |
| $  \begin{array}{ccc}  Val_i \times St & \xrightarrow{pr} & Val_i \\  u_i \downarrow & \quad \quad \quad & \downarrow id \\  St & \xrightarrow{l_i} & Val_i  \end{array}  $<br>$  \begin{array}{ccccc}  Val_i \times St & \xrightarrow{pr} & St & \xrightarrow{l_j} & Val_j \\  u_i \downarrow & & \downarrow id & & \\  St & \xrightarrow{l_j} & & & Val_j  \end{array}  $ | $  \begin{array}{ccc}  Par_i + Exc & \xleftarrow{in} & Par_i \\  c_i \uparrow & \quad \quad \quad & \uparrow id \\  Exc & \xleftarrow{t_i} & Par_i  \end{array}  $<br>$  \begin{array}{ccccc}  Par_i + Exc & \xleftarrow{in} & Exc & \xleftarrow{t_j} & Par_j \\  c_i \uparrow & & \downarrow id & & \uparrow id \\  Exc & \xleftarrow{t_j} & & & Par_j  \end{array}  $ |

So, there IS a duality between states and exceptions.

But states and exceptions are **computational effects**:  
the “type of states” and the “type of exceptions” are hidden,  
they do not appear explicitly in the syntax

We will see that the duality of their semantics  
comes from a duality of states and exceptions  
seen as **computational effects**.

But...  
what is a **computational effect**?

# Outline

## Introduction

1. The duality, explicitly
2. About effects
3. The duality, “effect”-ively

## Conclusion

# Monads for effects

[Moggi 1991]

*The basic idea behind the categorical semantics of effects is that we distinguish the object  $A$  of values from the object  $TA$  of computations.*

*Programs of type  $B$  with a parameter of type  $A$  ought to be interpreted by morphisms with codomain  $TB$ , but for their domain there are two alternatives, either  $A$  or  $TA$ .*

*We choose the first alternative, because it entails the second. Indeed computations of type  $A$  are the same as values of type  $TA$ .*

a program:  $A \rightarrow B$

is interpreted by a morphism:  $A \rightarrow TB$

# Monads for effects: exceptions

The monad of **exceptions** is  $TA = A + Exc$ .

A program of type  $B$  with a parameter of type  $A$ :

- ▶ **throws** an exception if it may map  $x \in A$  to  $e \in Exc$
- ▶ **catches** an exception if it may map  $e \in Exc$  to  $y \in B$

**Monads for effects.** A program of type  $B$  with a parameter of type  $A$  is interpreted by a morphism  $A \rightarrow B + Exc$ .

$\Rightarrow$  it may throw an exception

$\Rightarrow$  it **cannot catch** an exception

**Second alternative.** A program of type  $B$  with a parameter of type  $A$  is interpreted by a morphism  $A + Exc \rightarrow B + Exc$ .

$\Rightarrow$  it may throw an exception

$\Rightarrow$  it may catch an exception

# What is an effect?

**Claim.** A computational effect is  
an apparent lack of soundness.

There is a computational effect when:

- ▶ at first sight, the intended denotational semantics is not a model of the syntax,
- ▶ but the syntax may be “decorated” so as to recover soundness.

# States as effect

The intended denotational **semantics** (one location):

$$\left\{ \begin{array}{l} l : St \rightarrow Val \\ u : Val \times St \rightarrow St \\ \forall v \in Val \ \forall s \in St \ l(u(v, s)) = v \end{array} \right.$$

is not a model of the **apparent syntax**  
but it is a model of the **explicit syntax**

| <i>Apparent</i>                    |
|------------------------------------|
| $l : 1 \rightarrow V$              |
| $u : V \rightarrow 1$              |
| $l \circ u = id : V \rightarrow V$ |

| <i>Explicit</i>                             |
|---|
| $l : S \rightarrow V$                       |
| $u : V \times S \rightarrow S$              |
| $l \circ u = pr : V \times S \rightarrow V$ |



# Decorations for states

The **apparent syntax** may be **decorated**

$f : X \rightarrow Y$  is decorated as

$f^{(0)} : X \rightarrow Y$  if  $f$  is pure

$f^{(1)} : X \rightarrow Y$  if  $f$  is an accessor

$f^{(2)} : X \rightarrow Y$  if  $f$  is a modifier

$f = g$  is decorated as

$f =^{(sg)} g$  (strong) if  $f$  and  $g$  coincide on results and on states

$f =^{(wk)} g$  (weak) if  $f$  and  $g$  coincide on results (only)

| <i>Apparent</i>                      |
|--------------------------------------|
| $l : 1 \rightarrow V$                |
| $u : V \rightarrow 1$                |
| $l \circ u = id_V : V \rightarrow V$ |

| <i>Decorated</i>                            |
|---|
| $l^{(1)} : 1 \rightarrow V$                 |
| $u^{(2)} : V \rightarrow 1$                 |
| $l \circ u =^{(wk)} id_V : V \rightarrow V$ |

# Meaning of the decorations for states

The decorated syntax may be explicited

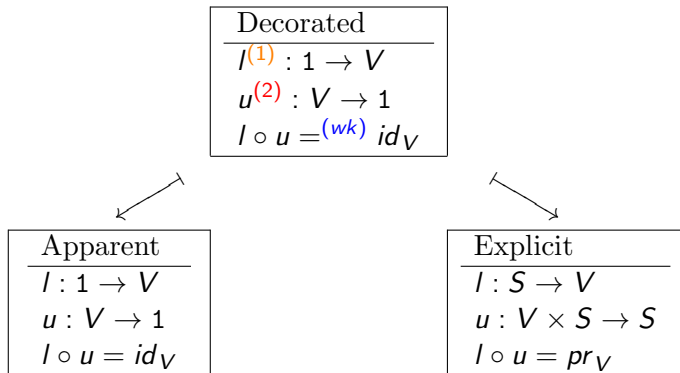
$$\begin{array}{ll} f^{(0)} : X \rightarrow Y & \text{as } f : X \rightarrow Y \\ f^{(1)} : X \rightarrow Y & \text{as } f : X \times S \rightarrow Y \\ f^{(2)} : X \rightarrow Y & \text{as } f : X \times S \rightarrow Y \times S \end{array}$$

$$\begin{array}{ll} f =^{(sg)} g : X \rightarrow Y & \text{as } f = g : X \times S \rightarrow Y \times S \\ f =^{(wk)} g : X \rightarrow Y & \text{as } pr_Y \circ f = pr_Y \circ g : X \times S \rightarrow Y \end{array}$$

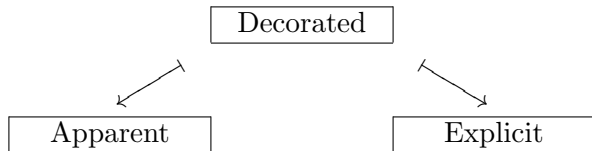
| <i>Decorated</i>                                     |
|--|
| $l^{(1)} : 1 \rightarrow V$                          |
| $u^{(2)} : V \rightarrow 1$                          |
| $l \circ u =^{(wk)} id_V : V \times S \rightarrow V$ |

| <i>Explicit</i>                               |
|---|
| $l : 1 \times S \rightarrow V$                |
| $u : V \times S \rightarrow S$                |
| $l \circ u = pr_V : V \times S \rightarrow V$ |

## States as effect: decorations



# Three syntaxes for one effect



The intended semantics

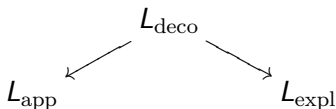
- ▶ is NOT a model of the apparent syntax (effect)
- ▶ is a model of the explicit syntax (obviously)
- ▶ it is also a model of the decorated syntax (by **adjunction**)

# A framework for effects

A language without effects is defined wrt **one** logic

$$L$$

A language with effects is defined wrt **a span** of logics



Defined in the category of **diagrammatic logics** [Duval&Lair 2002]  
which is based on **categorical notions**:

- ▶ adjunctions
- ▶ categories of fractions
- ▶ limit sketches

# Operations and equations

Our approach generalizes **algebraic specifications**

⇒ it involves (decorated) operations and equations  
handling exceptions is “symmetric” to updating states

The monads approach leads to **Lawvere theories**  
for getting operations and equations [Plotkin&Power 2001] but

- ▶ lookup, update, raise are **algebraic operations**
- ▶ handle is **not** an algebraic operation

The approach of monads and Lawvere theories can be extended  
for handling exceptions

- ▶ with **exception monads** [Schroeder&Mossakowski 2004]
- ▶ with **coalgebras** [Levy 2006]
- ▶ with **handlers** [Plotkin&Pretnar 2009]

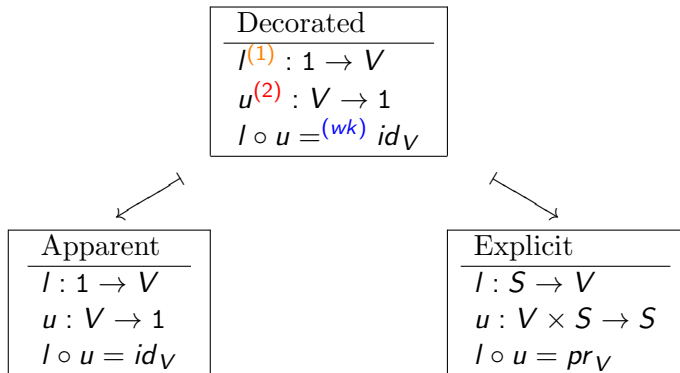
# Outline

## Introduction

1. The duality, explicitly
2. About effects
3. The duality, “effect”-ively

## Conclusion

# States





# Exceptions: decorations

$$f^{(0)} : X \rightarrow Y$$

$$f^{(1)} : X \rightarrow Y$$

$$f^{(2)} : X \rightarrow Y$$

pure

thrower

catcher

$$f : X \rightarrow Y$$

$$f : X \rightarrow Y + E$$

$$f : X + E \rightarrow Y + E$$

$$f =^{(sg)} g : X \rightarrow Y$$

$$f =^{(wk)} g : X \rightarrow Y$$

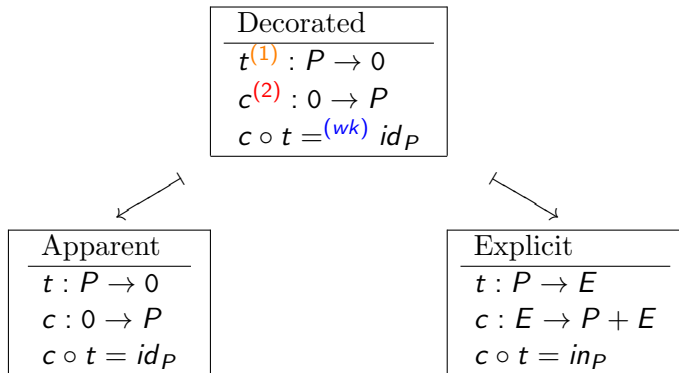
strong

weak

$$f = g : X + E \rightarrow Y + E$$

$$f \circ in_X = g \circ in_X : X \rightarrow Y + E$$

# Exceptions: key operations



# Duality of effects

| States  | Exceptions  |
|---|---|
| $i \in \text{Loc}, \text{Val}_i$<br>$1$   | $i \in \text{ExCstr}, \text{Par}_i$<br>$0$  |
| $l_i^{(1)} : 1 \rightarrow V_i$<br>$u_i^{(2)} : V_i \rightarrow 1$  | $0 \leftarrow P_i : t_i^{(1)}$<br>$P_i \leftarrow 0 : c_i^{(2)}$  |
| $  \begin{array}{ccc}  V_i & \xrightarrow{id} & V_i \\  u_i \downarrow & \stackrel{= (wk)}{\underset{l_i}{\longrightarrow}} & \downarrow id \\  1 & \xrightarrow{\quad} & V_i  \end{array}  $<br>$  \begin{array}{ccc}  V_i & \xrightarrow{\langle \rangle} 1 \xrightarrow{l_j} & V_j \\  u_i \downarrow & \stackrel{= (wk)}{\underset{l_j}{\longrightarrow}} & \downarrow id \\  1 & \xrightarrow{\quad} & V_j  \end{array}  $ | $  \begin{array}{ccc}  P_i & \xleftarrow{id} & P_i \\  c_i \uparrow & \stackrel{= (wk)}{\underset{t_i}{\longleftarrow}} & \uparrow id \\  0 & \xleftarrow{\quad} & P_i  \end{array}  $<br>$  \begin{array}{ccc}  P_i & \xleftarrow{[]} 0 \xleftarrow{t_j} & P_j \\  c_i \uparrow & \stackrel{= (wk)}{\underset{t_j}{\longleftarrow}} & \uparrow id \\  0 & \xleftarrow{\quad} & P_j  \end{array}  $ |

# Outline

## Introduction

1. The duality, explicitly
2. About effects
3. The duality, “effect”-ively

## Conclusion

# Conclusion

- ▶ An effect is an apparent lack of soundness
- ▶ a span of diagrammatic logics for each effect
- ▶ a new point of view on states
- ▶ a completely new point of view on exceptions
- ▶ a duality between states and exceptions

## Future work

- ▶ combining effects
- ▶ operational semantics