Designing proof systems from programming features: states and exceptions considered as dual effects

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

LJK, University of Grenoble

June 14., 2011 – Réalisabilité à Chambéry
Outline

Introduction

1. Duality, at the semantics level

2. Duality, at the logical level

3. About “decorated” proofs

Conclusion
This talk IS NOT about
  extracting programs from proofs

This talk IS about
  designing proof systems from programming features
What about **non-functional** features in programming languages? i.e., what about **computational effects**?

**Claim.** Each computational effect has an associated logic

**This talk IS about**

- the effects of states and exceptions, with their logics
A surprising result

There is a symmetry between the logics for states and exceptions, based on the well-known categorical duality:

<table>
<thead>
<tr>
<th>for states</th>
<th>for exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X \mapsto X \times S$</td>
<td>$X \mapsto X + E$</td>
</tr>
<tr>
<td>with fixed $S$</td>
<td>with fixed $E$</td>
</tr>
</tbody>
</table>
Outline

1. A symmetry between states and exceptions at the semantics level
2. A symmetry between states and exceptions at the logical level
3. About “decorated” proofs
Outline

Introduction

1. Duality, at the semantics level

2. Duality, at the logical level

3. About “decorated” proofs

Conclusion
Exceptions: values

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

- non-exceptional values
- exceptions

\[ X + \text{Exc} = \begin{array}{c} X \\ \text{Exc} \end{array} \]
Exceptions: functions

\[ f : X + \text{Exc} \rightarrow Y + \text{Exc} \]

- \textbf{f throws} an exception if it may map a non-exceptional value to an exception

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exc</td>
<td>Exc</td>
</tr>
</tbody>
</table>

- \textbf{f catches} an exception if it may map an exception to a non-exceptional value

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exc</td>
<td>Exc</td>
</tr>
</tbody>
</table>
Exceptions: the KEY THROW operations

\[ \text{Exc} = \text{set of exceptions} \]
\[ \text{ExCstr} = \text{set of exception constructors (or exception types)} \]

For each \( i \in \text{ExCstr} \):

- \( \text{Par}_i \) = set of parameters
- \( t_i : \text{Par}_i \to \text{Exc} \) = the KEY THROW operations
  or \( t_i : \text{Par}_i + \text{Exc} \to \text{Exc} \) such that \( \forall e \in \text{Exc}, \ t_i(e) = e \)

- \( t_i \) throws exceptions of constructor \( i \)
- \( t_i \) propagates exceptions

E.g. \( \text{Exc} = \sum_i \text{Par}_i \) with the \( t_i \)'s as coprojections
Exceptions: the KEY CATCH operations

For each \( i \in \text{ExCstr} \):

- \( c_i : \text{Exc} \rightarrow \text{Par}_i + \text{Exc} \) = the KEY CATCH operations

\[
\forall p \in \text{Par}_i \quad \begin{cases} 
    c_i(t_i(p)) = p \in \text{Par}_i \subseteq \text{Par}_i + \text{Exc} \\
    c_i(t_j(p)) = t_j(p) \in \text{Exc} \subseteq \text{Par}_i + \text{Exc} \quad (\forall j \neq i)
\end{cases}
\]

- \( c_i \) catches exceptions of constructor \( i \)
- \( c_i \) propagates exceptions of constructor \( j \neq i \)

E.g. \( \text{Exc} = \sum_i \text{Par}_i \) with the \( t_i \)'s as coprojections: these equations define the \( c_i \)'s
Exceptions: the RAISE (or THROW) construction

The key throwing and catching operations are encapsulated for building the usual raising and handling constructions.

- From key throwing \((t_i)\) to raising \((\text{raise}_{i,Y} \text{ or throw}_{i,Y})\):

\[
\text{raise}_{i,Y}(a) = t_i(a) \in Y + \text{Exc}
\]

\[\text{Par}_i \xrightarrow{\text{raise}_{i,Y}} Y + \text{Exc} \]

\[t_i \xrightarrow{=} \subseteq \text{Exc}\]
Exceptions: the HANDLE (or TRY...CATCH) construction

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

  \[ Exc \xrightarrow{c_i} Par_i + Exc \supseteq Y + Exc \]

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

  \[ X \xrightarrow{f} Y + Exc \supseteq Y + Exc \]
States

\( St = \text{set of states} \)
\( Loc = \text{set of locations} \)

For each \( i \in Loc \):

- \( Val_i = \text{set of values} \)
- \( l_i : St \to Val_i = \text{lookup function} \)
  or \( l_i : St \to Val_i \times St \) such that \( \forall s \in St, \ l_i(s) = (-, s) \)
- \( u_i : Val_i \times St \to St = \text{update function} \)

\[ \forall v_i \in Val_i \ \forall s \in St \left\{ \begin{array}{l}
  l_i(u_i(v_i, s)) = v_i \\
  l_j(u_i(v_i, s)) = l_j(s) \ (\forall j \neq i)
\end{array} \right. \]

E.g. \( St = \prod_i Val_i \) with the \( l_i \)'s as projections: these equations define the \( u_i \)'s
# Duality of semantics

## States

### $i \in \text{Loc}$, $\text{Val}_i$

$\text{St} \left( = \prod_{i \in \text{Loc}} \text{Val}_i \right)$

### $l_i : \text{St} \rightarrow \text{Val}_i$

### $u_i : \text{Val}_i \times \text{St} \rightarrow \text{St}$

### $\text{Val}_i \times \text{St} \xrightarrow{\text{pr}} \text{Val}_i$

$\downarrow \text{pr} = \downarrow \text{id}$

$\text{St} \xrightarrow{l_i} \text{Val}_i$

### $\text{Val}_i \times \text{St} \xrightarrow{\text{pr}} \text{St} \xrightarrow{l_j} \text{Val}_j$

$\downarrow \text{pr} = \downarrow \text{id}$

$\text{St} \xrightarrow{l_j} \text{Val}_j$

## Exceptions

### $i \in \text{ExCstr}$, $\text{Par}_i$

$\text{Exc} \left( = \sum_{i \in \text{ExCstr}} \text{Par}_i \right)$

### $\text{Exc} \leftarrow \text{Par}_i : t_i$

### $\text{Par}_i + \text{Exc} \leftarrow \text{Exc} : c_i$

### $\text{Val}_i \times \text{St} \xrightarrow{\text{pr}} \text{Val}_i$

$\downarrow \text{pr} = \downarrow \text{id}$

$\text{St} \xrightarrow{l_i} \text{Val}_i$

### $\text{Par}_i + \text{Exc} \xleftarrow{\text{in}} \text{Par}_i$

$\text{Exc} \xleftarrow{\text{in}} \text{Par}_i$

### $\text{Val}_i \times \text{St} \xrightarrow{\text{pr}} \text{St} \xrightarrow{l_j} \text{Val}_j$

$\downarrow \text{pr} = \downarrow \text{id}$

$\text{St} \xrightarrow{l_j} \text{Val}_j$

### $\text{Par}_i + \text{Exc} \xleftarrow{\text{in}} \text{Exc} \xleftarrow{t_j} \text{Par}_j$

$\text{Exc} \xleftarrow{t_j} \text{Par}_j$

### $\text{Val}_i \times \text{St} \xrightarrow{\text{pr}} \text{Val}_i$

$\downarrow \text{pr} = \downarrow \text{id}$

$\text{St} \xrightarrow{l_i} \text{Val}_i$

### $\text{Par}_i + \text{Exc} \xleftarrow{\text{in}} \text{Exc} \xleftarrow{t_j} \text{Par}_j$

$\text{Exc} \xleftarrow{t_j} \text{Par}_j$
So, there is a duality between states and exceptions, at the **semantics** level, involving a set of states $St$ and a set of exceptions $Exc$.

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.

In fact, the duality at the semantics level comes from a duality of states and exceptions seen as computational effects, at the **logical** level.
Outline

Introduction

1. Duality, at the semantics level

2. Duality, at the logical level

3. About “decorated” proofs

Conclusion
Monads for effects

[Moggi 1991] The basic idea behind the categorical semantics of effects is that we distinguish the object $X$ of values from the object $TX$ of computations (for some endofunctor $T$).

Programs of type $Y$ with a parameter of type $X$ ought to be interpreted by morphisms with codomain $TY$, but for their domain there are two alternatives, either $X$ or $TX$.

1. Moggi chooses the first alternative:
   a program $X \to Y$ is interpreted by a morphism $X \to TY$
   Then $T$ must be a monad – for substitution with a strength – for the context

2. The second alternative would be:
   a program $X \to Y$ is interpreted by a morphism $TX \to TY$
The monad of exceptions is $TX = X + \text{Exc}$.

1. First alternative.
   A program of type $Y$ with a parameter of type $X$ is interpreted by a morphism $X \to Y + \text{Exc}$.
   
   $\implies$ it may throw an exception
   
   $\implies\implies$ it cannot catch an exception

2. Second alternative.
   A program of type $Y$ with a parameter of type $X$ is interpreted by a morphism $X + \text{Exc} \to Y + \text{Exc}$.
   
   $\implies$ it may throw an exception
   
   $\implies\implies$ it may catch an exception
Effects, more generally

Claim. A computational effect is

an apparent lack of soundness

There is a computational effect when:

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

The monads approach from this point of view:

– operations are decorated as values or computations
  and every value can be seen as a computation
  (the base category is in the Kleisli category)
– a computation \( f^c : X \to Y \) stands for \( f : X \to TY \)
– a value \( f^v : X \to Y \) stands for \( f : X \to Y \stackrel{\eta_Y}{\to} TY \)
States, apparently

The intended semantics (one location):

\[
\begin{align*}
    l &: St \to Val \\
    u &: Val \times St \to St
\end{align*}
\]

\[
\forall v \in Val \quad \forall s \in St \quad l(u(v,s)) = v
\]

is not a model of the (equational) apparent syntax

\[
\begin{array}{|c|}
\hline
\text{Apparent} \\
\hline
l &: 1 \to V \\
u &: V \to 1 \\
l \circ u &= id : V \to V \\
\hline
\end{array}
\]
States, explicitly

The intended semantics (one location) is a model of the (equational) explicit syntax

<table>
<thead>
<tr>
<th>Explicit</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l : S \to V )</td>
</tr>
<tr>
<td>( u : V \times S \to S )</td>
</tr>
<tr>
<td>( l \circ u = pr : V \times S \to V )</td>
</tr>
</tbody>
</table>

\[\implies\] Two equational logics for states:
- The apparent logic is not sound, but close to the syntax
- The explicit logic is sound, but far from the syntax

Claim. There is a logic sound and close to the syntax, but it is not truly equational: it is a decorated logic
States as effect: decorations

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 (cf. \texttt{const} methods in C++)

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

<table>
<thead>
<tr>
<th>Apparent</th>
<th>Decorated</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l : \mathbb{1} \rightarrow V )</td>
<td>( l^{(1)} : \mathbb{1} \rightarrow V )</td>
</tr>
<tr>
<td>( u : V \rightarrow \mathbb{1} )</td>
<td>( u^{(2)} : V \rightarrow \mathbb{1} )</td>
</tr>
<tr>
<td>( l \circ u = id_V : V \rightarrow V )</td>
<td>( l \circ u =^{(wk)} id_V : V \rightarrow V )</td>
</tr>
</tbody>
</table>
States as effect: meaning of the decorations

The decorated syntax may be explicitated

\[ f^{(0)} : X \rightarrow Y \quad \text{as} \quad f : X \rightarrow Y \]
\[ f^{(1)} : X \rightarrow Y \quad \text{as} \quad f : X \times S \rightarrow Y \]
\[ f^{(2)} : X \rightarrow Y \quad \text{as} \quad f : X \times S \rightarrow Y \times S \]

\[ f =^{(sg)} g : X \rightarrow Y \quad \text{as} \quad f = g : X \times S \rightarrow Y \times S \]
\[ f =^{(wk)} g : X \rightarrow Y \quad \text{as} \quad pr_Y \circ f = pr_Y \circ g : X \times S \rightarrow Y \]

<table>
<thead>
<tr>
<th>Decorated</th>
<th>Explicit</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l^{(1)} : \mathbb{1} \rightarrow V )</td>
<td>( l : \mathbb{1} \times S \rightarrow V )</td>
</tr>
<tr>
<td>( u^{(2)} : V \rightarrow \mathbb{1} )</td>
<td>( u : V \times S \rightarrow S )</td>
</tr>
<tr>
<td>( l \circ u =^{(wk)} id_V : V \times S \rightarrow V )</td>
<td>( l \circ u = pr_V : V \times S \rightarrow V )</td>
</tr>
</tbody>
</table>
States as effect: three logics

\[
\begin{align*}
\text{Decorated} \\
(l^{(1)} : \mathbb{1} \to V \\
u^{(2)} : V \to \mathbb{1} \\
l \circ u = (\text{wk}) \ id_V
\end{align*}
\]

\[
\begin{align*}
\text{Apparent} \\
l : \mathbb{1} \to V \\
u : V \to \mathbb{1} \\
l \circ u = id_V
\end{align*}
\]

\[
\begin{align*}
\text{Explicit} \\
l : S \to V \\
u : V \times S \to S \\
l \circ u = pr_V
\end{align*}
\]

The intended semantics

- is NOT a model of the apparent syntax (effect)
- is a model of the explicit syntax (obviously)
- is also a model of the decorated syntax (by adjunction)
Exceptions as effect

The intended **semantics** (one exc. constructor):

\[
\begin{align*}
  t & : \text{Par} \rightarrow \text{Exc} \\
  c & : \text{Exc} \rightarrow \text{Par} + \text{Exc} \\
  \forall p \in \text{Par} \quad c(t(p)) &= p
\end{align*}
\]

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

<table>
<thead>
<tr>
<th>Apparent</th>
<th>Explicit</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t : P \rightarrow \emptyset )</td>
<td>( t : P \rightarrow E )</td>
</tr>
<tr>
<td>( c : \emptyset \rightarrow P )</td>
<td>( c : E \rightarrow P + E )</td>
</tr>
<tr>
<td>( c \circ t = \text{id} : P \rightarrow P )</td>
<td>( c \circ t = \text{in} : P \rightarrow P + E )</td>
</tr>
</tbody>
</table>
Exceptions as effect: decorations

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 a propagator (it may throw exceptions)
- \( f^{(2)} : X \rightarrow Y \) if \( f \) is a catcher (it may throw and catch exceptions)

\( f = g \) is decorated as

- \( f =^{(sg)} g \) (strong) if \( f \) and \( g \) coincide on exceptions and on values
- \( f =^{(wk)} g \) (weak) if \( f \) and \( g \) coincide on values (only)

<table>
<thead>
<tr>
<th>Apparent</th>
<th>Decorated</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t : P \rightarrow \emptyset )</td>
<td>( t^{(1)} : P \rightarrow \emptyset )</td>
</tr>
<tr>
<td>( c : \emptyset \rightarrow P )</td>
<td>( c^{(2)} : \emptyset \rightarrow P )</td>
</tr>
<tr>
<td>( c \circ t = id : P \rightarrow P )</td>
<td>( c^{(2)} \circ t^{(1)} =^{(wk)} id^{(0)} : P \rightarrow P )</td>
</tr>
</tbody>
</table>
Exceptions as effect: meaning of the decorations

The decorated syntax may be explicated

\[ f^{(0)} : X \to Y \quad \text{as} \quad f : X \to Y \]
\[ f^{(1)} : X \to Y \quad \text{as} \quad f : X \to Y + E \]
\[ f^{(2)} : X \to Y \quad \text{as} \quad f : X + E \to Y + E \]

\[ f =^{(sg)} g : X \to Y \quad \text{as} \quad f = g : X \times S \to Y \times S \]
\[ f =^{(wk)} g : X \to Y \quad \text{as} \quad f \circ \text{in}_X = g \circ \text{in}_X : X \to Y + E \]

<table>
<thead>
<tr>
<th>Decorated</th>
<th>Explicit</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t^{(1)} : P \to \emptyset )</td>
<td>( t : P \to E )</td>
</tr>
<tr>
<td>( c^{(2)} : \emptyset \to P )</td>
<td>( c : E \to P + E )</td>
</tr>
<tr>
<td>( c^{(2)} \circ t^{(1)} =^{(wk)} id^{(0)} : P \to P )</td>
<td>( c \circ t = \text{in} : P \to P + E )</td>
</tr>
</tbody>
</table>
Exceptions as effect: three logics

\[
\begin{align*}
\text{Decorated} \\
(t^{(1)} : P \rightarrow \emptyset) \\
(c^{(2)} : \emptyset \rightarrow P) \\
c \circ t =^{(wk)} id_P
\end{align*}
\]

\[
\begin{align*}
\text{Apparent} \\
t : P \rightarrow \emptyset \\
c : \emptyset \rightarrow P \\
c \circ t = id_P
\end{align*}
\]

\[
\begin{align*}
\text{Explicit} \\
t : P \rightarrow E \\
c : E \rightarrow P + E \\
c \circ t = id_P
\end{align*}
\]

The intended semantics

- is NOT a model of the apparent syntax (effect)
- is a model of the explicit syntax (obviously)
- is also a model of the decorated syntax (by adjunction)
### Duality of effects

<table>
<thead>
<tr>
<th>States</th>
<th>Exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i \in \text{Loc}, ; V_i )</td>
<td>( i \in \text{ExCstr}, ; P_i )</td>
</tr>
<tr>
<td>( 1 )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( l_i^{(1)} : 1 \to V_i )</td>
<td>( \emptyset \leftarrow P_i : t_i^{(1)} )</td>
</tr>
<tr>
<td>( u_i^{(2)} : V_i \to 1 )</td>
<td>( P_i \leftarrow \emptyset : c_i^{(2)} )</td>
</tr>
</tbody>
</table>

\[
V_i \xrightarrow{id} V_i \\
\downarrow l_i = (wk) \downarrow id \\
1 \xrightarrow{u_i} V_i \\
\]

\[
V_i \xrightarrow{1} l_j \xrightarrow{V_i} V_j \\
\downarrow u_i = (wk) \downarrow id \\
1 \xrightarrow{l_j} V_j \\
\]

\[
P_i \xleftarrow{id} P_i \\
\uparrow c_i = (wk) \uparrow id \\
\emptyset \xleftarrow{t_i} P_i \\
\]

\[
P_j \xleftarrow{0} t_j \xleftarrow{P_j} P_i \\
\uparrow c_i = (wk) \uparrow id \\
\emptyset \xleftarrow{t_j} P_j \\
\]
Outline

Introduction

1. Duality, at the semantics level

2. Duality, at the logical level

3. About “decorated” proofs

Conclusion
Operations and equations

- The monads approach leads to Lawvere theories for getting operations and equations [Plotkin&Power 2001]. This can be extended
  - with exception monads [Schroeder&Mossakowski 2004]
  - with coalgebras [Levy 2006]
  - with handlers [Plotkin&Pretnar 2009]

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

- Our approach generalizes algebraic specifications. It involves (decorated) operations and equations

Then
- catching exceptions is symmetric to updating states
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

$L_{deco}$

$L_{app}$

$L_{expl}$

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

- Adjunctions [Kan 1958]
- Categories of fractions [Gabriel&Zisman 1967]
- Limit sketches [Ehresmann 1968]
One logic: models

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

\[
\begin{array}{ccc}
S & \overset{L}{\to} & T \\
\downarrow & & \downarrow \\
\perp & \leftarrow & \perp
\end{array}
\]

induced by a morphism of limit sketches

- $S$ is the category of specifications
- $T$ is the category of theories
- Each specification $\Sigma$ presents the theory $L\Sigma$
- A model $M : \Sigma \to \Theta$ is an "oblique" morphism: $M : L\Sigma \to \Theta$ in $T$ or $M : \Sigma \to R\Theta$ in $S$
One logic: proofs

\( T \) is a category of fractions on \( S \):

a fraction is a cospan in \( S \) with numerator \( \sigma \) and denominator \( \tau \) such that \( L \tau \) is invertible in \( T \)

\[
\Sigma_1 \xrightarrow{\sigma} \Sigma'_2 \xleftarrow{\tau} \Sigma_2
\]

This fraction can be seen as

- an instance of the specification \( \Sigma_1 \) in \( \Sigma_2 \)
- or an inference rule with hypothesis \( \Sigma_2 \) and conclusion \( \Sigma_1 \).

The inference step is the composition of fractions:

applying a rule with hypothesis \( H \) and conclusion \( C \) to an instance of \( H \) in \( \Sigma \) yields an instance of \( C \) in \( \Sigma \).
A category of logics

A morphism of logics $F : L_1 \to L_2$
is a pair of left adjoint functors $(F_S, F_T)$ with a commutative
square induced by a commutative square of limit sketches

$$
\begin{array}{ccc}
S_1 & \xrightarrow{L_1} & T_1 \\
\downarrow F_S & \Downarrow \approx & \downarrow F_T \\
S_2 & \xrightarrow{L_2} & T_2
\end{array}
$$

This yields the category of diagrammatic logics
Which provides a framework for spans of logics

$$
\begin{array}{ccc}
& L_{\text{deco}} & \\
L_{\text{app}} & \xleftarrow{} & L_{\text{expl}}
\end{array}
$$
In this talk, for states and exceptions, \( L_{\text{app}} \) and \( L_{\text{expl}} \) are (variants of) equational logic. Each decorated proof is mapped to an equational proof:

- either by dropping the decorations (by \( F_{\text{app}} \)) → an “uninteresting” proof
- or by expliciting the decorations (by \( F_{\text{expl}} \)) → a “complicated” proof
Some decorated rules for states (1)

\[
\begin{align*}
(0\text{-to-1}) & \quad \frac{f(0)}{f(1)} \\
(1\text{-to-2}) & \quad \frac{f(1)}{f(2)}
\end{align*}
\]

\[
\begin{align*}
(sg\text{-trans}) & \quad \frac{f(2) = (sg) g(2)}{f(2) = (sg) h(2)} \\
(sg\text{-subs}) & \quad \frac{g_1(2) = (sg) g_2(2)}{(g_1 \circ f)(2) = (sg) (g_2 \circ f)(2)} \\
(sg\text{-repl}) & \quad \frac{f_1(2) = (sg) f_2(2)}{(g \circ f_1)(2) = (sg) (g \circ f_2)(2)}
\end{align*}
\]

\[
\begin{align*}
(wk\text{-trans}) & \quad \frac{f(2) = (wk) g(2)}{f(2) = (wk) h(2)} \\
(wk\text{-subs}) & \quad \frac{g_1(2) = (wk) g_2(2)}{(g_1 \circ f)(2) = (wk) (g_2 \circ f)(2)} \\
(wk\text{-repl}) & \quad \frac{f_1(2) = (wk) f_2(2) g(0)}{(g \circ f_1)(2) = (wk) (g \circ f_2)(2)}
\end{align*}
\]
Some decorated rules for states (2)

\[
\begin{array}{c|c}
\text{(sg-to-wk)} & f^{(2)} =_{(sg)} g^{(2)} \\
& f^{(2)} =_{(wk)} g^{(2)} \\
\text{(wk-to-sg)} & f^{(1)} =_{(wk)} g^{(1)} \\
& f^{(1)} =_{(sg)} g^{(1)}
\end{array}
\]

And there is a “decorated product”

\[
(l_j^{(1)} : 1 \to V_j)_{j \in \text{Loc}}
\]

such that

\[
f^{(2)} =_{(sg)} g^{(2)} : X \to 1 \iff \\
\forall j \in \text{Loc}, \ (l_j \circ f)^{(2)} =_{(wk)} (l_j \circ g)^{(2)} : X \to V_j
\]
A decorated proof (for states)

**Proposition.** For every \( i \in \text{Loc} \):

- Semantically: \( \forall s \in \text{St}, \ u_i(l_i(s), s) = s \)
- Explicitly: \( u_i \circ l_i = id_S \)
- Decorated: \( u_i^{(2)} \circ l_i^{(1)} = (sg) \ id_1^{(0)} \)

**Proof.** \( \forall j \in \text{Loc} \), \( l_j^{(1)} \circ u_i^{(2)} \circ l_i^{(1)} = (wk) \ l_j^{(1)} \)

When \( j = i \):

\[
\begin{align*}
&(wk\text{-subs}) \quad l_i \circ u_i = (wk) \ id_V, \\
&\frac{\quad l_i \circ u_i = (wk) \ id_V}{\quad l_i \circ u_i \circ l_i = (wk) \ l_i}
\end{align*}
\]

When \( j \neq i \):

\[
\begin{align*}
&(wk\text{-subs}) \quad l_j \circ u_i = (wk) \ l_j \circ \langle \rangle V, \\
&(wk\text{-trans}) \quad l_j \circ u_i \circ l_i = (wk) \ l_j \circ \langle \rangle V \circ l_i, \\
&(sg\text{-repl}) \quad \langle \rangle V_i \circ l_i = (sg) \ id_1, \\
&(sg\text{-to-wk}) \quad l_j \circ \langle \rangle V_i \circ l_i = (sg) \ l_j, \\
&\frac{\quad l_j \circ \langle \rangle V_i \circ l_i = (sg) \ l_j}{\quad l_j \circ \langle \rangle V_i \circ l_i = (wk) \ l_j}
\end{align*}
\]

\( l_j \circ u_i \circ l_i = (wk) \ l_j \)
Decorated rules and proofs (for exceptions)

Decorated rules and proofs for exceptions are dual to decorated rules and proofs for states.

**Proposition.** For every \( i \in \text{ExCstr} \):

- Semantically: \( \forall e \in \text{Exc}, \; t_i(c_i(e)) = e \)
- Explicitly: \( t_i \circ c_i = id_E \)
- Decorated: \( t_i^{(1)} \circ c_i^{(2)} \equiv^{(sg)} id^{(0)}_1 \)

**Proof.** Dual to the proof for states.
More decorated proofs (for states)

Equations from [Plotkin&Power 2002] as stated in [Mellies 2010]

- Interaction update-update:
  storing a value $v$ and then a value $v'$ at the same location $i$ is just like storing the value $v'$ in the location $i$. $\forall i \in \text{Loc}$,
  \[
u_i^{(2)} \circ \pi^{(0)} \circ (u_i \times id_{V_i})^{(2)} \equiv^{(sg)} u_i^{(2)} \circ \pi^{(0)}\]

- Commutation update-update:
  the order of storing in two different locations $i$ and $j$ does not matter. $\forall i \neq j \in \text{Loc}$,
  \[
u_j^{(2)} \circ \pi^{(0)} \circ (u_i \times id_{V_j})^{(2)} \equiv^{(sg)} u_i^{(2)} \circ \pi^{(0)} \circ (id_{V_i} \times u_j)^{(2)} \]

Decorated proofs in [Dumas&Duval&Fousse&Reynaud 2011]
More decorated proofs (for exceptions)

- when catching an exception constructor $i$ twice, the second catcher is never used. \( \forall i \in \text{ExCstr}, \)

\[
\text{try } \{ f \} \text{catch } i \{g\} \text{catch } i \{h\} =^{(sg)} \text{try } \{ f \} \text{catch } i \{g\}
\]

- when catching two different exception constructors $i$ and $j$, the order of catching does not matter. \( \forall i \neq j \in \text{ExCstr}, \)

\[
\text{try } \{ f \} \text{catch } i \{g\} \text{catch } j \{h\} =^{(sg)} \text{try } \{ f \} \text{catch } j \{h\} \text{catch } i \{g\}
\]

Proof:
1. Start from the previous equations for states
2. Dualize
3. Encapsulate
Outline

Introduction

1. Duality, at the semantics level

2. Duality, at the logical level

3. About “decorated” proofs

Conclusion
Conclusion

- An effect is an apparent lack of soundness
- Designing proof systems from programming features: each computational effect has an associated logic
- States and exceptions may be considered as dual effects

Future work

- Using a proof assistant (Coq) for decorated proofs
- Combining effects by composing the spans of logics
Thanks for your attention