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

Dominique Duval

LJK, University of Grenoble

July 5., 2011 – PPS – Groupe de Travail Sémantique
Outline

Introduction

1. Duality, at the semantics level

2. Duality, at the logical level

3. About “decorated” proofs

Conclusion
The Curry Howard Lambek correspondence

<table>
<thead>
<tr>
<th>intuitionistic logic</th>
<th>typed lambda calculus</th>
<th>cartesian closed categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>propositions</td>
<td>types</td>
<td>objects</td>
</tr>
<tr>
<td>proofs</td>
<td>terms</td>
<td>morphisms</td>
</tr>
</tbody>
</table>

What about non-functional features in programming languages? i.e., what about computational effects?

Claim. Each computational effect has an associated logic

In this talk: 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

Reference:
J.-G. Dumas, D. Duval, L. Fousse, J.-C. Reynaud
States and exceptions considered as dual effects
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

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

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

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

  \[
  \begin{array}{c|c}
  X & Y \\
  \hline
  \text{Exc} & \text{Exc} \\
  \end{array}
  \]

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

  \[
  \begin{array}{c|c}
  X & Y \\
  \hline
  \text{Exc} & \text{Exc} \\
  \end{array}
  \]
Exceptions: the KEY THROW operations

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

For each \( i \in ExCstr \):

- \( Par_i \) = set of parameters
- \( t_i : Par_i \rightarrow Exc \) = the KEY THROW operations
  \[ \text{or } t_i : Par_i + Exc \rightarrow Exc \text{ such that } \forall e \in Exc, \ t_i(e) = e \]

\[ \begin{array}{c|c}
    Par_i & \\
    \hline
    Exc & \emptyset \\
    \hline
    Exc & Exc \\
\end{array} \]

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

E.g. \( Exc = \sum_i Par_i \) with the \( t_i \)'s as coprojections
Exceptions: the KEY CATCH operations

For each $i \in \text{ExCstr}$:

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

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

- $c_i$ 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: encapsulation

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

- The usual raising construction throws an exception viewed as an element of some type $X$
- The usual handling construction catches an exception inside a block carefully delimited.
Exceptions: the RAISE (or THROW) construction

The usual raising construction throws an exception viewed as an element of some type $X$.

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

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

\[
\begin{array}{c}
Par_i \xrightarrow{raise_{i,Y}} Y + Exc \\
\xrightarrow{t_i} Exc
\end{array}
\]
Exceptions: the HANDLE (or TRY...CATCH) construction

The usual handling construction catches an exception inside a block carefully delimited

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

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

\[ St = \text{set of states} \]
\[ Loc = \text{set of locations} \]

For each \( i \in Loc \):

- \( Val_i \) = set of values
- \( l_i : St \to Val_i \) = lookup function
  - or \( l_i : St \to Val_i \times St \) such that \( \forall s \in St, \ l_i(s) = (-, s) \)
- \( u_i : Val_i \times St \to St \) = update function
  - \( \forall v \in Val_i, \forall s \in St \)
    \[
    \begin{cases} 
    l_i(u_i(v, s)) = v \\
    l_j(u_i(v, s)) = l_j(s) \ (\forall j \neq i) 
    \end{cases} 
    \]

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

<table>
<thead>
<tr>
<th>States</th>
<th>Exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i \in \text{Loc}, \ Val_i )</td>
<td>( i \in \text{ExcCstr}, \ Par_i )</td>
</tr>
<tr>
<td>( St = \prod_{i \in \text{Loc}} \ Val_i )</td>
<td>( \text{Exc} = \sum_{i \in \text{ExcCstr}} \ Par_i )</td>
</tr>
<tr>
<td>( l_i : \ St \rightarrow \ Val_i )</td>
<td>( \text{Exc} \leftarrow \ Par_i : t_i )</td>
</tr>
<tr>
<td>( u_i : \ Val_i \times \ St \rightarrow \ St )</td>
<td>( \text{Par}_i + \text{Exc} \leftarrow \text{Exc} : c_i )</td>
</tr>
</tbody>
</table>

### States Diagram

\[
\begin{align*}
\text{Val}_i \times \text{St} & \xrightarrow{\text{pr}} \text{Val}_i \\
\text{St} & \xrightarrow{\text{id}} \text{Val}_i
\end{align*}
\]

\[
\begin{align*}
\text{Val}_i \times \text{St} & \xrightarrow{\text{pr}} \text{St} \xrightarrow{l_j} \text{Val}_j \\
\text{St} & \xrightarrow{\text{id}} \text{Val}_j
\end{align*}
\]

### Exceptions Diagram

\[
\begin{align*}
\text{Par}_i + \text{Exc} & \xrightarrow{\text{in}} \text{Par}_i \\
\text{Exc} & \xrightarrow{\text{id}} \text{Par}_i
\end{align*}
\]

\[
\begin{align*}
\text{Par}_i + \text{Exc} & \xrightarrow{\text{in}} \text{Exc} \xrightarrow{t_j} \text{Par}_j \\
\text{Exc} & \xrightarrow{\text{id}} \text{Par}_j
\end{align*}
\]
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
[Moggi 1991] \textit{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 $TX \to TY$
Monads for effects: exceptions

The monad of exceptions is $TX = X + Exc$.

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

2. Second alternative.
   A program of type $Y$ with a parameter of type $X$ is interpreted by a morphism $X + Exc \rightarrow Y + Exc$.
   
   $\implies$ it may throw an exception
   $\implies$ 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
– a computation \( f^c : X \rightarrow Y \) stands for \( f : X \rightarrow TY \)
– a value \( f^v : X \rightarrow Y \) stands for \( f : X \rightarrow Y \xrightarrow{\eta_Y} TY \)
States, apparently

The intended semantics (one location):

\[
\begin{align*}
& l : St \rightarrow Val \\
& u : Val \times St \rightarrow St \\
& \forall v \in Val \ \forall s \in St \quad l(u(v, s)) = v
\end{align*}
\]

IS NOT a model of the apparent syntax

<table>
<thead>
<tr>
<th>Apparent</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l : \mathbb{1} \rightarrow V$</td>
</tr>
<tr>
<td>$u : V \rightarrow \mathbb{1}$</td>
</tr>
<tr>
<td>$l \circ u = id : V \rightarrow V$</td>
</tr>
</tbody>
</table>
States, explicitly

The intended semantics (one location)

\[
\begin{align*}
  l : St &\to Val \\
  u : Val \times St &\to St \\
  \forall v \in Val \forall s \in St \quad l(u(v,s)) = v
\end{align*}
\]

IS a model of the 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>
States, equationally

There are two equational logics “for states”:

<table>
<thead>
<tr>
<th>Apparent logic</th>
<th>Explicit logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT sound</td>
<td>sound</td>
</tr>
<tr>
<td>close to the syntax</td>
<td>FAR from the syntax</td>
</tr>
</tbody>
</table>

Claim. There is a third logic for states – NOT “truly” equational:

<table>
<thead>
<tr>
<th>Decorated logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>sound</td>
</tr>
<tr>
<td>close to the syntax</td>
</tr>
</tbody>
</table>
States as effect: decorations

The apparent syntax may be decorated:

- An operation \( f : X \rightarrow Y \) is decorated as
  \( f^{(0)} : X \rightarrow Y \) if \( f \) is pure
  \( f^{(1)} : X \rightarrow Y \) if \( f \) is an accessor (cf. const methods in C++)
  \( f^{(2)} : X \rightarrow Y \) if \( f \) is a modifier

- An equation \( f = g \) 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 : 1 \rightarrow V )</td>
<td>( l^{(1)} : 1 \rightarrow V )</td>
</tr>
<tr>
<td>( u : V \rightarrow 1 )</td>
<td>( u^{(2)} : V \rightarrow 1 )</td>
</tr>
<tr>
<td>( l \circ u = id_V : V \rightarrow V )</td>
<td>( l \circ u \stackrel{(wk)}{=} id_V : V \rightarrow V )</td>
</tr>
</tbody>
</table>
States as effect: expliciting the decorations

The decorated syntax may be explicited

- For operations:
  \[ f^{(0)} : X \to Y \text{ as } f : X \to Y \]
  \[ f^{(1)} : X \to Y \text{ as } f : X \times S \to Y \]
  \[ f^{(2)} : X \to Y \text{ as } f : X \times S \to Y \times S \]

- For equations:
  \[ f =^{(sg)} g \text{ as } f = g : X \times S \to Y \times S \]
  \[ f =^{(wk)} g \text{ as } pr_Y \circ f = pr_Y \circ g : X \times S \to Y \]

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

**Decorated**

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

**Apparent**

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

**Explicit**

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

The intended semantics

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

The intended semantics (one exception 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
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:

• An operation \( f : X \to Y \) is decorated as
  \( f^{(0)} : X \to Y \) if \( f \) is pure
  \( f^{(1)} : X \to Y \) if \( f \) is a propagator (it may throw exceptions)
  \( f^{(2)} : X \to Y \) if \( f \) is a catcher (it may throw and catch exc.)

• An equation \( f = g \) is decorated as
  \( f =^{(sg)} g \) (strong) if \( f \) and \( g \) coincide on exc. 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 \to \emptyset )</td>
<td>( t^{(1)} : P \to \emptyset )</td>
</tr>
<tr>
<td>( c : \emptyset \to P )</td>
<td>( c^{(2)} : \emptyset \to P )</td>
</tr>
<tr>
<td>( c \circ t = id : P \to P )</td>
<td>( c^{(2)} \circ t^{(1)} =^{(wk)} id^{(0)} : P \to P )</td>
</tr>
</tbody>
</table>
Exceptions as effect: expliciting the decorations

The decorated syntax may be explicited

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

- For equations:
  \[ f =^{(sg)} g \text{ as } f = g : X \times S \to Y \times S \]
  \[ f =^{(wk)} g \text{ as } f \circ in_X = g \circ in_X : X \to Y + E \]

<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 = in : P \to P + E ]</td>
</tr>
</tbody>
</table>
Exceptions as effect: three logics

Decorated

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

Apparent

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

Explicit

\[
\begin{align*}
t : & \quad P \rightarrow E \\
c : & \quad E \rightarrow P + E \\
c \circ t &= in_P
\end{align*}
\]

The intended semantics

- IS NOT a model of the apparent syntax (effect)
- IS a model of the explicit syntax (obviously)
- IS 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>$0$</td>
</tr>
<tr>
<td>$l_i^{(1)} : 1 \to V_i$</td>
<td>$0 \leftarrow P_i : t_i^{(1)}$</td>
</tr>
<tr>
<td>$u_i^{(2)} : V_i \to 1$</td>
<td>$P_i \leftarrow 0 : c_i^{(2)}$</td>
</tr>
<tr>
<td>$V_i \xrightarrow{id} V_i$</td>
<td>$P_i \xleftarrow{id} P_i$</td>
</tr>
<tr>
<td>$u_i \downarrow \xleftarrow{(wk)} l_i \xrightarrow{id} V_i$</td>
<td>$c_i \xleftarrow{(wk)} \uparrow P_i$</td>
</tr>
<tr>
<td>$1 \xrightarrow{l_i} V_i$</td>
<td>$0 \xleftarrow{t_i} P_i$</td>
</tr>
<tr>
<td>$V_i \to 1 \xrightarrow{l_j} V_j$</td>
<td>$P_i \xleftarrow{0 \xleftarrow{t_j} P_j}$</td>
</tr>
<tr>
<td>$u_i \downarrow \xleftarrow{(wk)} l_j \xrightarrow{id} V_j$</td>
<td>$c_i \xleftarrow{(wk)} \uparrow P_j$</td>
</tr>
<tr>
<td>$1 \xrightarrow{l_j} V_j$</td>
<td>$0 \xleftarrow{t_j} P_j$</td>
</tr>
</tbody>
</table>
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 with respect to one logic $L$

A language with effects is defined with respect to a span of logics $L_{\text{deco}}$, $L_{\text{app}}$, $L_{\text{expl}}$

Morphisms of logics are defined in the category of diagrammatic logics [Duval&Lair 2002]. This is based on:

- Adjunctions [Kan 1958]
- Categories of fractions [Gabriel&Zisman 1967]
- Limit sketches [Ehresmann 1968]
A diagrammatic logic is a left adjoint functor $L$ with a full and faithful right adjoint $R$ 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 $M : \Sigma \rightarrow \Theta$ is an "oblique" morphism: $M : L\Sigma \rightarrow \Theta$ in $T$ or $M : \Sigma \rightarrow R\Theta$ 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 $F: L_1 \rightarrow L_2$

is a pair of left adjoint functors $(F_S, F_T)$

in a commutative square

\[
\begin{array}{ccc}
S_1 & \xrightarrow{L_1} & T_1 \\
\downarrow F_S & \simeq & \downarrow F_T \\
S_2 & \xrightarrow{L_2} & T_2
\end{array}
\]

induced by a commutative square of limit sketches

This yields the category of diagrammatic logics
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)

<table>
<thead>
<tr>
<th>(0-to-1)</th>
<th>$f^{(0)}$</th>
<th>$f^{(1)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1-to-2)</td>
<td>$f^{(1)}$</td>
<td>$f^{(2)}$</td>
</tr>
</tbody>
</table>

**sg-substitution (sg-sub)**

| $g_1^{(2)} = (sg) g_2^{(2)}$ | $(g_1 \circ f)^{(2)} = (sg) (g_2 \circ f)^{(2)}$ |

**sg-replacement (sg-repl)**

| $f_1^{(2)} = (sg) f_2^{(2)}$ | $(g \circ f_1)^{(2)} = (sg) (g \circ f_2)^{(2)}$ |

**wk-substitution (wk-sub)**

| $g_1^{(2)} = (wk) g_2^{(2)}$ | $(g_1 \circ f)^{(2)} = (wk) (g_2 \circ f)^{(2)}$ |

**wk-replacement (wk-repl)**

| $f_1^{(2)} = (wk) f_2^{(2)} g^{(0)}$ | $(g \circ f_1)^{(2)} = (wk) (g \circ f_2)^{(2)}$ |
Some decorated rules for states (2)

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

and the lookup’s form a “decorated product” \((l_j^{(1)})_{j \in \text{Loc}}\) such that

\[
f(2) = (sg) g(2) \iff \forall j \in \text{Loc}, \quad (l_j \circ f)(2) = (wk) (l_j \circ g)(2)
\]

\[\xymatrix{X \ar[r]^f \ar[d]_g & 1 \ar@{..>}[d] \ar[dl]_{l_i} \ar[dr]^{V_i} \\
& \vdots & \vdots \\
& l_j \ar[ul] & V_j}
\]
A decorated proof (for states)

Proposition. For every $i \in \mathit{Loc}$:

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

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

When $j = i$:

\[
\begin{align*}
(l_i \circ u_i &= (wk) \ id_{V_i} \\
\Rightarrow \quad l_i \circ u_i \circ l_i &= (wk) \ l_i 
\end{align*}
\]

When $j \neq i$:

\[
\begin{align*}
(l_j \circ u_i &= (wk) \ l_j \circ \langle \rangle_{V_i} \\
\Rightarrow \quad l_j \circ u_i \circ l_i &= (wk) \ l_j \circ \langle \rangle_{V_i} \circ l_i \\
\Rightarrow \quad l_j \circ u_i \circ l_i &= (wk) \ l_j \\
\Rightarrow \quad l_j \circ \langle \rangle_{V_i} \circ l_i &= (sg) \ id_1 \\
\Rightarrow \quad l_j \circ \langle \rangle_{V_i} \circ l_i &= (sg) \ l_j \\
\Rightarrow \quad l_j \circ \langle \rangle_{V_i} \circ l_i &= (wk) \ l_j
\end{align*}
\]

$\Rightarrow \quad l_j \circ u_i \circ l_i = (wk) \ l_j$
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: $t_i^{(1)} \circ c_i^{(2)} = (^{(sg)} \; id_{1}^{(0)})$

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

Equations from [Plotkin&Power 2002] as stated in [Melliès 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 \( i \).

\[
\forall i \in \text{Loc},
\]

\[
u_i^{(2)} \circ (u_i \times id_{V_i})^{(2)} =^{(sg)} u_i^{(2)} \circ \pi_2^{(0)}
\]

▶ Commutation update-update:

the order of storing in two different locations \( i \) and \( j \)

does not matter.

\[
\forall i \neq j \in \text{Loc},
\]

\[
u_j^{(2)} \circ (u_i \times id_{V_j})^{(2)} =^{(sg)} u_i^{(2)} \circ (id_{V_i} \times u_j)^{(2)}
\]

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

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

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

- **Commutation catch-catch:**
  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
A question

[Melliès 2010] About the notion of monad and the notion of sheaf on a Grothendieck topology:

*It is fascinating to observe that the most promising links between mathematics and programming languages emerged at these somewhat *himalayan heights.*

Mount Everest, 8 848 m.

**Question.** What is the “height” of our (naive?) approach?

Grand pic de Belledonne, 2 977 m.
Thanks for your attention