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

This talk IS NOT about

extracting programs from proofs

This talk IS about

designing proof systems from programming features

The Curry Howard Lambek correspondence

intuitionistic logic	typed lambda calculus	cartesian closed categories
propositions	types	objects
proofs	terms	morphisms

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:

for states	for exceptions
$X \mapsto X \times S$ with fixed S	$X \mapsto X + E$ with fixed E

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

Exceptions: values

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

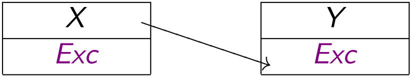
- non-exceptional values
- exceptions

$$X + \mathit{Exc} \;=\; \begin{array}{|c|}\hline X \\ \hline \mathit{Exc} \\ \hline \end{array}$$

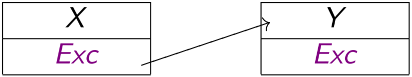
Exceptions: functions

$$f : X + \textcolor{purple}{Exc} \to Y + \textcolor{purple}{Exc}$$

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



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



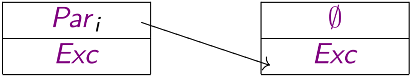
Exceptions: the KEY THROW operations

Exc $=$ set of exceptions
ExCstr $=$ set of exception constructors (or exception types)

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

- Par_i $=$ set of parameters
- $t_i : \mathit{Par}_i \to \mathit{Exc}$ $=$ the KEY THROW operations

  or $t_i : \mathit{Par}_i + \mathit{Exc} \to \mathit{Exc}$ such that $\forall\, e \in \mathit{Exc},\ t_i(e) = e$



– t_i throws exceptions of constructor i

– t_i propagates exceptions

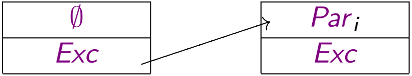
E.g. $\mathit{Exc} = \sum_i \mathit{Par}_i$ with the t_i's as coprojections

Exceptions: the KEY CATCH operations

For each $i \in ExCstr$:

- $c_i : Exc \to Par_i + Exc$ = the KEY CATCH operations

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



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

E.g. $Exc = \sum_i Par_i$ 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 ($raise_{i,Y}$ or $throw_{i,Y}$):

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

$$\begin{tikzcd}
Par_i \arrow[rr, "raise_{i,Y}"] \arrow[drr, "t_i"'] & \arrow[dr, phantom, "="] & Y + Exc \\
& & Exc \arrow[u, "\subseteq"']
\end{tikzcd}$$

Exceptions: the HANDLE (or TRY...CATCH) construction

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

$$\begin{tikzcd}
 & Par_i \arrow[d, "\subseteq"'] \arrow[drr, "g"] & & \\
Exc \arrow[r, "c_i"] & Par_i + Exc \arrow[rr] \arrow[rr, phantom, shift left=3, "="] \arrow[rr, phantom, shift right=3, "="'] & & Y + Exc \\
 & Exc \arrow[u, "\subseteq"] \arrow[urr, "\subseteq"'] & &
\end{tikzcd}$$

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

$$\begin{tikzcd}
 & Y \arrow[d, "\subseteq"'] \arrow[drr, "\subseteq"] & & \\
X \arrow[r, "f"] & Y + Exc \arrow[rr] \arrow[rr, phantom, shift left=3, "="] \arrow[rr, phantom, shift right=3, "="'] & & Y + Exc \\
 & Exc \arrow[u, "\subseteq"] \arrow[urr, "catch\, i\, \{g\}"'] & &
\end{tikzcd}$$

States

St = set of states
Loc = 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

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

E.g. $St = \prod_i Val_i$ with the l_i's as projections:
these equations define the u_i's

Duality of semantics

States	Exceptions
$i \in Loc, \; Val_i$ $St \; (= \prod_{i \in Loc} Val_i)$	$i \in ExCstr, \; Par_i$ $Exc \; (= \sum_{i \in ExCstr} Par_i)$
$l_i : St \to Val_i$ $u_i : Val_i \times St \to St$	$Exc \leftarrow Par_i : t_i$ $Par_i + Exc \leftarrow Exc : c_i$
$\begin{tikzcd} Val_i \times St \arrow[r, "pr"] \arrow[d, "u_i"'] \arrow[dr, phantom, "="] & Val_i \arrow[d, "id"] \\ St \arrow[r, "l_i"] & Val_i \end{tikzcd}$ $\begin{tikzcd} Val_i \times St \arrow[r, "pr"] \arrow[d, "u_i"'] \arrow[drr, phantom, "="] & St \arrow[r, "l_j"] & Val_j \arrow[d, "id"] \\ St \arrow[rr, "l_j"] & & Val_j \end{tikzcd}$	$\begin{tikzcd} Par_i + Exc \arrow[dr, phantom, "="] & Par_i \arrow[l, "in"'] \\ Exc \arrow[u, "c_i"] & Par_i \arrow[l, "t_i"] \arrow[u, "id"'] \end{tikzcd}$ $\begin{tikzcd} Par_i + Exc \arrow[drr, phantom, "="] & Exc \arrow[l, "in"'] & Par_j \arrow[l, "t_j"'] \\ Exc \arrow[u, "c_i"] & & Par_j \arrow[ll, "t_j"] \arrow[u, "id"'] \end{tikzcd}$

- 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

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 $TX \to TY$

Monads for effects: exceptions

The monad of exceptions is $TX = X + \mathit{Exc}$.

1. First alternative.
   A program of type Y with a parameter of type X
   is interpreted by a morphism $X \to Y + \mathit{Exc}$.
   $\Longrightarrow$ it may throw an exception
   $\Longrightarrow$ it cannot catch an exception
2. Second alternative.
   A program of type Y with a parameter of type X
   is interpreted by a morphism $X + \mathit{Exc} \to Y + \mathit{Exc}$.
   $\Longrightarrow$ it may throw an exception
   $\Longrightarrow$ 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 $f : X \to Y \xrightarrow{\eta_Y} TY$

States, apparently

The intended semantics (one location):

$$\begin{cases} l : St \to Val \\ u : Val \times St \to St \\ \forall v \in Val \;\; \forall s \in St \;\; l(u(v,s)) = v \end{cases}$$

is not a model of the (equational) apparent syntax

Apparent
$l : \mathbb{1} \to V$
$u : V \to \mathbb{1}$
$l \circ u = id : V \to V$

States, explicitly

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

Explicit
$l : S \to V$
$u : V \times S \to S$
$l \circ u = pr : V \times S \to V$

$\Longrightarrow$ 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

$f : X \to Y$ is decorated as

- $f^{(0)} : X \to Y$ if f is pure
- $f^{(1)} : X \to Y$ if f is an accessor (cf. `const` methods in `C++`)
- $f^{(2)} : X \to Y$ 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)

Apparent
$l : \mathbb{1} \to V$
$u : V \to \mathbb{1}$
$l \circ u = id_V : V \to V$

Decorated
$l^{(1)} : \mathbb{1} \to V$
$u^{(2)} : V \to \mathbb{1}$
$l \circ u =^{(wk)} id_V : V \to V$

States as effect: meaning of the decorations

The decorated syntax may be explicited

$f^{(0)} : X \to Y$	as	$f : X \to Y$
$f^{(1)} : X \to Y$	as	$f : X \times S \to Y$
$f^{(2)} : X \to Y$	as	$f : X \times S \to Y \times S$
$f =^{(sg)} g : X \to Y$	as	$f = g : X \times S \to Y \times S$
$f =^{(wk)} g : X \to Y$	as	$pr_Y \circ f = pr_Y \circ g : X \times S \to Y$

Decorated
$l^{(1)} : \mathbb{1} \to V$
$u^{(2)} : V \to \mathbb{1}$
$l \circ u =^{(wk)} id_V : V \times S \to V$

Explicit
$l : \mathbb{1} \times S \to V$
$u : V \times S \to S$
$l \circ u = pr_V : V \times S \to V$

States as effect: three logics

Decorated

$$l^{(1)} : \mathbb{1} \to V$$
$$u^{(2)} : V \to \mathbb{1}$$
$$l \circ u =^{(wk)} id_V$$

Apparent

$$l : \mathbb{1} \to V$$
$$u : V \to \mathbb{1}$$
$$l \circ u = id_V$$

Explicit

$$l : S \to V$$
$$u : V \times S \to S$$
$$l \circ u = pr_V$$

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):

$$\begin{cases} t : Par \to Exc \\ c : Exc \to Par + Exc \\ \forall\, p \in Par \;\; c(t(p)) = p \end{cases}$$

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

Apparent
$t : P \to \mathbb{0}$
$c : \mathbb{0} \to P$
$c \circ t = id : P \to P$

Explicit
$t : P \to E$
$c : E \to P + E$
$c \circ t = in : P \to P + E$

Exceptions as effect: decorations

The apparent syntax may be decorated

$f : X \to Y$ 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 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)

Apparent
$t : P \to \mathbb{0}$
$c : \mathbb{0} \to P$
$c \circ t = id : P \to P$

Decorated
$t^{(1)} : P \to \mathbb{0}$
$c^{(2)} : \mathbb{0} \to P$
$c^{(2)} \circ t^{(1)} =^{(wk)} id^{(0)} : P \to P$

Exceptions as effect: meaning of the decorations

The decorated syntax may be explicited

$f^{(0)} : X \to Y$	as	$f : X \to Y$
$f^{(1)} : X \to Y$	as	$f : X \to Y + E$
$f^{(2)} : X \to Y$	as	$f : X + E \to Y + E$

$f =^{(sg)} g : X \to Y$	as	$f = g : X \times S \to Y \times S$
$f =^{(wk)} g : X \to Y$	as	$f \circ in_X = g \circ in_X : X \to Y + E$

Decorated
$t^{(1)} : P \to \mathbb{0}$
$c^{(2)} : \mathbb{0} \to P$
$c^{(2)} \circ t^{(1)} =^{(wk)} id^{(0)} : P \to P$

Explicit
$t : P \to E$
$c : E \to P + E$
$c \circ t = in : P \to P + E$

Exceptions as effect: three logics

Decorated

$t^{(1)} : P \to \mathbb{0}$

$c^{(2)} : \mathbb{0} \to P$

$c \circ t =^{(wk)} id_P$

Apparent

$t : P \to \mathbb{0}$

$c : \mathbb{0} \to P$

$c \circ t = id_P$

Explicit

$t : P \to E$

$c : E \to P + E$

$c \circ t = in_P$

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

States	Exceptions
$i \in Loc,\ V_i$ $\mathbb{1}$	$i \in ExCstr,\ P_i$ $\mathbb{0}$
$l_i^{(1)} : \mathbb{1} \to V_i$ $u_i^{(2)} : V_i \to \mathbb{1}$	$\mathbb{0} \leftarrow P_i : t_i^{(1)}$ $P_i \leftarrow \mathbb{0} : c_i^{(2)}$
$\begin{tikzcd} V_i \arrow[r, "id"] \arrow[d, "u_i"'] \arrow[dr, phantom, "=^{(wk)}"] & V_i \arrow[d, "id"] \\ \mathbb{1} \arrow[r, "l_i"] & V_i \end{tikzcd}$ $\begin{tikzcd} V_i \arrow[r] \arrow[d, "u_i"'] & \mathbb{1} \arrow[r, "l_j"] \arrow[d, phantom, "=^{(wk)}"] & V_j \arrow[d, "id"] \\ \mathbb{1} \arrow[rr, "l_j"] & & V_j \end{tikzcd}$	$\begin{tikzcd} P_i & P_i \arrow[l, "id"'] \\ \mathbb{0} \arrow[u, "c_i"] \arrow[ur, phantom, "=^{(wk)}"] & P_i \arrow[l, "t_i"'] \arrow[u, "id"'] \end{tikzcd}$ $\begin{tikzcd} P_i & \mathbb{0} \arrow[l] & P_j \arrow[l, "t_j"'] \\ \mathbb{0} \arrow[u, "c_i"] & =^{(wk)} & P_j \arrow[ll, "t_j"'] \arrow[u, "id"'] \end{tikzcd}$

Outline

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

  $\Longrightarrow$ 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

$$
\begin{tikzcd}
 & L_{\mathrm{deco}} \arrow[dl] \arrow[dr] & \\
L_{\mathrm{app}} & & L_{\mathrm{expl}}
\end{tikzcd}
$$

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

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

induced by a morphism of limit sketches

- $\mathbf{S}$ is the category of specifications
- $\mathbf{T}$ is the category of theories
- Each specification Σ presents the theory $L\Sigma$
- A model $M : \Sigma \to \Theta$ is an "oblique" morphism:
  $M\colon L\Sigma \to \Theta$ in $\mathbf{T}$ or $M\colon \Sigma \to R\Theta$ in $\mathbf{S}$

One logic: proofs

$\mathbf{T}$ is a category of fractions on $\mathbf{S}$:
a fraction is a cospan in $\mathbf{S}$ with numerator σ
and denominator τ such that $L\tau$ is invertible in $\mathbf{T}$

$$\Sigma_1 \xrightarrow{\ \sigma\ } \Sigma_2' \underset{}{\overset{\tau}{\leftrightarrows}} \Sigma_2$$

This fraction can be seen as

- an instance of the specification Σ_1 in Σ_2
- or an inference rule with hypothesis Σ_2 and conclusion Σ_1.

The inference step is the composition of fractions:
applying a rule with hypothesis H and conclusion C
to an instance of H in Σ
yields an instance of C in Σ.

A category of logics

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

$$
\begin{tikzcd}
\mathbf{S}_1 \arrow[r, "L_1"] \arrow[d, "F_S"'] \arrow[dr, phantom, "\cong"] & \mathbf{T}_1 \arrow[d, "F_T"] \\
\mathbf{S}_2 \arrow[r, "L_2"'] & \mathbf{T}_2
\end{tikzcd}
$$

This yields the category of diagrammatic logics

Which provides a framework for spans of logics

$$
\begin{tikzcd}
 & L_{\mathrm{deco}} \arrow[dl] \arrow[dr] & \\
L_{\mathrm{app}} & & L_{\mathrm{expl}}
\end{tikzcd}
$$

Decorated proofs

$$\begin{tikzcd}[column sep=small] & L_{\text{deco}} \arrow[dl, "F_{\text{app}}"'] \arrow[dr, "F_{\text{expl}}"] & \\ L_{\text{app}} & & L_{\text{expl}} \end{tikzcd}$$

In this talk, for states and exceptions,
L_{app} and L_{expl} are (variants of) equational logic.
Each decorated proof is mapped to an equational proof

- either by dropping the decorations (by F_{app})
  $\rightarrow$ an "uninteresting" proof
- or by expliciting the decorations (by F_{expl})
  $\rightarrow$ a "complicated" proof

Some decorated rules for states (1)

$$(\text{0-to-1}) \quad \frac{f^{(0)}}{f^{(1)}}$$

$$(\text{1-to-2}) \quad \frac{f^{(1)}}{f^{(2)}}$$

$$(sg\text{-trans}) \quad \frac{f^{(2)} =^{(sg)} g^{(2)} \quad g^{(2)} =^{(sg)} h^{(2)}}{f^{(2)} =^{(sg)} h^{(2)}}$$

$$(sg\text{-subs}) \quad \frac{g_1^{(2)} =^{(sg)} g_2^{(2)}}{(g_1 \circ f)^{(2)} =^{(sg)} (g_2 \circ f)^{(2)}}$$

$$(sg\text{-repl}) \quad \frac{f_1^{(2)} =^{(sg)} f_2^{(2)}}{(g \circ f_1)^{(2)} =^{(sg)} (g \circ f_2)^{(2)}}$$

$$(wk\text{-trans}) \quad \frac{f^{(2)} =^{(wk)} g^{(2)} \quad g^{(2)} =^{(wk)} h^{(2)}}{f^{(2)} =^{(wk)} h^{(2)}}$$

$$(wk\text{-subs}) \quad \frac{g_1^{(2)} =^{(wk)} g_2^{(2)}}{(g_1 \circ f)^{(2)} =^{(wk)} (g_2 \circ f)^{(2)}}$$

$$(wk\text{-repl}) \quad \frac{f_1^{(2)} =^{(wk)} f_2^{(2)} \quad g^{(0)}}{(g \circ f_1)^{(2)} =^{(wk)} (g \circ f_2)^{(2)}}$$

Some decorated rules for states (2)

$$\boxed{\begin{array}{cc} (sg\text{-to-}wk) & \dfrac{f^{(2)} =^{(sg)} g^{(2)}}{f^{(2)} =^{(wk)} g^{(2)}} \\ (wk\text{-to-}sg) & \dfrac{f^{(1)} =^{(wk)} g^{(1)}}{f^{(1)} =^{(sg)} g^{(1)}} \end{array}}$$

And there is a "decorated product"

$$(l_j^{(1)} : \mathbb{1} \to V_j)_{j \in Loc}$$

such that

$$f^{(2)} =^{(sg)} g^{(2)} : X \to \mathbb{1} \iff$$

$$\forall j \in Loc,\ (l_j \circ f)^{(2)} =^{(wk)} (l_j \circ g)^{(2)} : X \to V_j$$

A decorated proof (for states)

Proposition. For every $i \in Loc$:

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

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

When $j = i$:

$$(wk\text{-subs})\ \frac{l_i \circ u_i =^{(wk)} id_{V_i}}{l_i \circ u_i \circ l_i =^{(wk)} l_i}$$

When $j \neq i$:

$$(wk\text{-trans})\ \dfrac{(wk\text{-subs})\ \dfrac{l_j \circ u_i =^{(wk)} l_j \circ \langle\,\rangle_{V_i}}{l_j \circ u_i \circ l_i =^{(wk)} l_j \circ \langle\,\rangle_{V_i} \circ l_i} \qquad (sg\text{-to-}wk)\ \dfrac{(sg\text{-repl})\ \dfrac{\dfrac{\vdots}{\langle\,\rangle_{V_i} \circ l_i =^{(sg)} id_{\mathbb{1}}}}{l_j \circ \langle\,\rangle_{V_i} \circ l_i =^{(sg)} l_j}}{l_j \circ \langle\,\rangle_{V_i} \circ l_i =^{(wk)} l_j}}{l_j \circ u_i \circ l_i =^{(wk)} l_j}$$

Decorated rules and proofs (for exceptions)

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

Proposition. For every $i \in ExCstr$:

- Semantically: $\forall e \in Exc,\ t_i(c_i(e)) = e$
- Explicitly: $t_i \circ c_i = id_E$
- Decorated: $t_i^{(1)} \circ c_i^{(2)} =^{(sg)} id_{\mathbb{1}}^{(0)}$

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. $\forall i \in Loc$,

$$u_i^{(2)} \circ \pi^{(0)} \circ (u_i \times id_{V_i})^{(2)} =^{(sg)} u_i^{(2)} \circ \pi^{(0)}$$

- *Commutation update-update:*
  the order of storing in two different locations i and j does not matter. $\forall i \neq j \in Loc$,

$$u_j^{(2)} \circ \pi^{(0)} \circ (u_i \times id_{V_j})^{(2)} =^{(sg)} u_i^{(2)} \circ \pi^{(0)} \circ (id_{V_i} \times u_j)^{(2)}$$

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. $\forall i \in \mathit{ExCstr}$,

$$\mathit{try}\ \{f\} \mathit{catch}\, i\, \{g\} \mathit{catch}\, i\, \{h\} =^{(sg)} \mathit{try}\ \{f\} \mathit{catch}\, i\, \{g\}$$

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

$$\mathit{try}\ \{f\} \mathit{catch}\, i\, \{g\} \mathit{catch}\, j\, \{h\} =^{(sg)} \mathit{try}\ \{f\} \mathit{catch}\, j\, \{h\} \mathit{catch}\, i\, \{g\}$$

Proof:

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

Outline

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