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

Outline

Semantics of computational effects?

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

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

Semantics of computational effects?

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

logic	programming	categories
propositions	*types*	*objects*
proofs	*terms*	*morphisms*
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
effect	*categorical structure ??*
(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 $T(B)$.*

$$\boxed{p : A \to B} \text{ is interpreted as } \boxed{p : A \to T(B)}$$

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

Exceptions. $p : A \to B$ is interpreted as $p : A \to B + Exc$, 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	$lookup : Val \to Loc$ $update : 1 \to Loc \times Val$ with 7 equations
exceptions	$raise_e : 0 \to 1$ 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:

$$\boxed{p : A \to B} \text{ is interpreted as } \boxed{p : A \to T(B)}$$

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

Notes

About the authors

Our background lies in computer algebra: abstract algebra, algorithmic, programmation (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, programmation (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

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"

We define three specifications for dealing with states

$$\begin{tikzcd}
 & \text{DECORATED}: \Sigma_0 \arrow[dl, bend right, "F_1"'] \arrow[dr, bend left, "F_2"] & \\
\text{APPARENT}: \Sigma_1 & & \text{EXPLICIT}: \Sigma_2
\end{tikzcd}$$

The apparent specification

Notations

$Loc = \{X, Y, ...\}$ = the set of locations

1 = the unit type

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
- $\begin{cases} \text{lookup} & l_i : 1 \to V_i \\ \text{update} & u_i : V_i \to 1 \end{cases}$
- and 2 equations

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

The explicit specification

Notation

S = the "type of states"

From the semantics we get the explicit equational specification Σ_2
For each location $i \in Loc$:

- a type V_i for the values of i
- $\begin{cases} \text{lookup} & l_i : S \to V_i \\ \text{update} & u_i : V_i \times S \to S \end{cases}$
- and 2 equations

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

The decorated specification

Decorations for functions:

- (0) for pure functions
- (1) for accessors (= inspectors)
- (2) for modifiers

AND decorations for equations

With the decorations we form the decorated specification Σ_0
For each location $i \in Loc$:

- a type V_i for the values of i
- $\begin{cases} \text{lookup} & l_i^{(1)} : 1 \to V_i \\ \text{update} & u_i^{(2)} : V_i \to 1 \end{cases}$
- and 2 equations

Three specifications

F_1 F_2

DECORATED: Σ_0

$l_i^{(1)} : 1 \to V_i$

$u_i^{(2)} : V_i \to 1$

2 equations

APPARENT: Σ_1

$l_i : 1 \to V_i$

$u_i : V_i \to 1$

2 equations

EXPLICIT: Σ_2

$l_i : S \to V_i$

$u_i : V_i \times S \to S$

2 equations

- F_1: from decorated to apparent: wipe out all decorations
- F_2: from decorated to explicit: according to the decoration (next slide)

Expansion of decorations

The expansion F_2 provides the meaning of the decorations

	Σ_0	$\xmapsto{F_2}$	Σ_2
pure	$X \xrightarrow{f^{(0)}} Y$	$\xmapsto{F_2}$	$X \xrightarrow{f} Y$
accessor	$X \xrightarrow{f^{(1)}} Y$	$\xmapsto{F_2}$	$X \times S \xrightarrow{f} Y$
modifier	$X \xrightarrow{f^{(2)}} Y$	$\xmapsto{F_2}$	$X \times S \xrightarrow{f} Y \times S$

Relevance of decorations

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

A zoom for states

Claim. *The 3 specifications are defined in 3 "logics" related by a "span of logics":*

$$\begin{tikzcd} & \text{DECORATED}: L_0 \arrow[dl, bend right, "F_1"'] \arrow[dr, bend left, "F_2"] & \\ \text{APPARENT}: L_1 & & \text{EXPLICIT}: L_2 \end{tikzcd}$$

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

We have designed an "abstract" category of logics

Outline

A category of logics

A diagrammatic logic is a functor L
with a full and faithful right adjoint R [...]

$$\begin{tikzcd} \mathbf{S} \arrow[r, "L"] \arrow[r, phantom, shift right=3ex, "\perp"] & \mathbf{T} \arrow[l, bend left, "R"] \end{tikzcd}$$

- $\mathbf{T}$: category of theories
- $\mathbf{S}$: 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

Models and proofs

With respect to a logic:

$$\begin{tikzcd} \mathbf{S} \arrow[r, bend left, "L"] \arrow[r, phantom, "\perp"] & \mathbf{T} \arrow[l, bend left, "R"] \end{tikzcd}$$

- A model M of a specification Σ with values in a theory Θ is a morphism $L\Sigma \to \Theta$ in $\mathbf{T}$, i.e., a morphism $\Sigma \to R\Theta$ in $\mathbf{S}$

  [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 $\mathbf{T}$

- A proof is a morphism in $\mathbf{T}$ [...]

Ex. Monadic equational logic

- $\mathbf{T}$: categories
- $\mathbf{S}$: "linear" sketches (= graphs with some composition)

Morphisms of logics

Based on arrow categories

- A morphism $F : L_1 \to L_2$ is a pair of left adjoint functors (F_S, F_T) such that $L_2 \circ F_S \cong F_T \circ L_1$ [...]

$$
\begin{tikzcd}
\mathbf{S}_1 \arrow[r, "L_1"] \arrow[d, "F_S"'] \arrow[d, phantom, shift left=6, "\dashv"] & \mathbf{T}_1 \arrow[l, bend left, "R_1"] \arrow[d, "F_T"'] \arrow[d, phantom, shift left=6, "\dashv"] \\
\mathbf{S}_2 \arrow[r, "L_2"] \arrow[u, bend left, "G_S"] & \mathbf{T}_2 \arrow[l, bend left, "R_2"] \arrow[u, bend left, "G_T"]
\end{tikzcd}
$$

This provides the category of diagrammatic logics

A zoom for states

$$\begin{tikzcd} & \text{DECORATED: } L_0 \arrow[dl, bend right, "F_1"'] \arrow[dr, bend left, "F_2"] & \\ \text{APPARENT: } L_1 & & \text{EXPLICIT: } L_2 \end{tikzcd}$$

- 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,...
- 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

Outline

Exceptions as dual of states?

Monads:

states	$T(X) = (X \times St)^{St}$
exceptions	$T(X) = X + Exc$

Lawvere theories:

states	$lookup : Val \to Loc$ $update : 1 \to Loc \times Val$ with 7 equations
exceptions	$raise_e : 0 \to 1$ 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

Claim. *The duality between $X \times S$ and $X + E$ extends as a duality between states and exceptions*

l_i	lookup	dual to	r_i	"raise"
u_i	update	dual to	h_i	"handle"

Dual of states: three specifications

$Etype$ = the set of exceptional types

P_i = the type of parameters of type i, for each $i \in Etype$

0 = the empty type

E = the "type of exceptions"

DECORATED: Σ_0
$r_i^{(1)} : P_i \to 0$
$h_i^{(2)} : 0 \to P_i$
2 equations

F_1

F_2

APPARENT: Σ_1
$r_i : P_i \to 0$
$h_i : 0 \to P_i$
2 equations

EXPLICIT: Σ_2
$r_i : P_i \to E$
$h_i : E \to P_i + E$
2 equations

Dual of states: decorations

Decorations for functions:

- (0) for pure functions
- (1) for propagators
- (2) for handlers

AND decorations for equations

The expansion functor F_2 provides the meaning of the decorations

pure	$X \xrightarrow{f^{(0)}} Y$	$\xmapsto{F_2}$	$X \xrightarrow{f} Y$
propagator	$X \xrightarrow{f^{(1)}} Y$	$\xmapsto{F_2}$	$X \xrightarrow{f} Y + E$
handler	$X \xrightarrow{f^{(2)}} Y$	$\xmapsto{F_2}$	$X + E \xrightarrow{f} Y + E$

Dual of states: a zoom for exceptions

$$\begin{tikzcd} & \text{DECORATED: } L_0 \arrow[dl, bend right, "F_1"'] \arrow[dr, bend left, "F_2"] & \\ \text{APPARENT: } L_1 & & \text{EXPLICIT: } L_2 \end{tikzcd}$$

- 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 E and with a functor $- + E$
- 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 0,...
- 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

Exceptions: interpretation of $r_i^{(1)}$ and $h_i^{(2)}$

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*

Exceptions: interpretation of $r_i^{(1)}$ and $h_i^{(2)}$

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*

The expansion and interpretation of $r_i^{(1)}$ and $h_i^{(2)}$:

$r_i : P_i \to E$	$p \mapsto e = r_i(p)$
$h_i : E \to P_i + E$	$\begin{cases} e = r_i(p) \mapsto p \\ e = r_j(p) \mapsto e \quad (j \neq i) \end{cases}$

Exceptions: encapsulation of $r_i^{(1)}$

In raising an exception, the empty type is hidden

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

$$\begin{tikzcd} P_i \arrow[r, "r_i^{(1)}"'] \arrow[rr, bend left, "raise_{i,X}^{(1)}"] & 0 \arrow[r, "[\,]^{(0)}"'] & X \end{tikzcd}$$

- first $r_i^{(1)}$ raises an exception of exceptional type i
- then $[\,]_X^{(0)}$ converts this exception to type X

Exceptions: encapsulation of $h_i^{(2)}$

For handling an exception of type i raised by $f^{(1)} : X \to Y$, using $g^{(1)} : P_i \to Y$:

- $f^{(1)}(x)$ is called, if it returns $y \in Y$ THEN *return* y
- otherwise some exception e is raised, then apply $h_i^{(2)}$ to test whether $e = r_i(p)$,
  if so THEN *return* $g^{(1)}(p)$, ELSE *return* e

$$
\begin{tikzcd}
 & Y \arrow[d, "\mathtt{id}^{(0)}"'] \arrow[rrr, bend left, "\mathtt{id}^{(0)}"] & & & \\
X \arrow[r, "f^{(1)}"] & Y \arrow[rrr, "{[\mathtt{id}|g\circ h_i]^{(2)}}"] & & & Y \\
 & 0 \arrow[u, "{[\,]^{(0)}}"] \arrow[rr, "h_i^{(2)}"'] & & P_i \arrow[ur, bend right, "g^{(1)}"'] &
\end{tikzcd}
$$

Exceptions: encapsulation of $h_i^{(2)}$

For handling an exception of type i raised by $f^{(1)} : X \to Y$, using $g^{(1)} : P_i \to Y$:

- $f^{(1)}(x)$ is called, if it returns $y \in Y$ THEN *return* y
- otherwise some exception e is raised, then apply $h_i^{(2)}$ to test whether $e = r_i(p)$,
  if so THEN *return* $g^{(1)}(p)$, ELSE *return* e

$$\begin{tikzcd}
 & Y \arrow[d, "\mathtt{id}^{(0)}"'] \arrow[rrr, bend left, "\mathtt{id}^{(0)}"] & & & \\
X \arrow[r, "f^{(1)}"] & Y \arrow[rrr, "{[\mathtt{id}|g\circ h_i]^{(2)}}"] & & & Y \\
 & 0 \arrow[u, "{[\,]^{(0)}}"] \arrow[rr, "h_i^{(2)}"'] & & P_i \arrow[ur, bend right, "g^{(1)}"'] &
\end{tikzcd}$$

- finally, this handler $[\mathtt{id}|g \circ h_i]^{(2)} \circ f^{(1)}$ is encapsulated in a propagator $handle_{i,f,g}^{(1)}$

$$\begin{tikzcd}
X \arrow[r, "f^{(1)}"] \arrow[rrr, bend left, "handle_{i,f,g}^{(1)}"] & Y \arrow[rr, "{[\mathtt{id}|g\circ h_i]^{(2)}}"] & & Y
\end{tikzcd}$$

Outline

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).