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

Introduction

States

Diagrammatic logics

Exceptions

Conclusion
Semantics of computational effects?

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

<table>
<thead>
<tr>
<th>logic</th>
<th>programming</th>
<th>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>
<tr>
<td>intuitionistic</td>
<td>simply typed lambda</td>
<td>cartesian closed</td>
</tr>
<tr>
<td>logic</td>
<td>calculus</td>
<td>categories</td>
</tr>
</tbody>
</table>
Semantics of computational effects?

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

<table>
<thead>
<tr>
<th>logic</th>
<th>programming</th>
<th>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>
<tr>
<td>intuitionistic logic</td>
<td>simply typed lambda calculus</td>
<td>cartesian closed categories</td>
</tr>
</tbody>
</table>

What about categorical semantics of non-functional programming languages, i.e., languages with effects?

<table>
<thead>
<tr>
<th>programming</th>
<th>categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>effect</td>
<td>categorical structure ??</td>
</tr>
<tr>
<td>(global) states</td>
<td>??</td>
</tr>
<tr>
<td>exceptions</td>
<td>??</td>
</tr>
</tbody>
</table>
Effects as monads

Moggi [1989], cf. Haskell:

*Programs of type $B$ with a parameter of type $A$ are interpreted by morphisms from $A$ to $T(B)$.***

$\begin{align*}
\text{p : A} & \rightarrow \text{B} \text{ is interpreted as } \text{p : A} & \rightarrow & \text{T(B)}
\end{align*}$

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

**Exceptions.** $\text{p : A} \rightarrow \text{B}$ is interpreted as $\text{p : A} \rightarrow \text{B} + \text{Exc}$, where $\text{Exc}$ is the set of exceptions

<table>
<thead>
<tr>
<th>effect</th>
<th>monad $(T, \eta, \mu)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>states</td>
<td>$T(X) = (X \times \text{St})^{\text{St}}$</td>
</tr>
<tr>
<td>exceptions</td>
<td>$T(X) = X + \text{Exc}$</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>effect</th>
<th>Lawvere theory generated by</th>
</tr>
</thead>
<tbody>
<tr>
<td>states</td>
<td><em>lookup : Val → Loc</em></td>
</tr>
<tr>
<td></td>
<td><em>update : 1 → Loc × Val</em></td>
</tr>
<tr>
<td></td>
<td>with 7 equations</td>
</tr>
<tr>
<td>exceptions</td>
<td><em>raise_e : 0 → 1 for e ∈ Exc</em></td>
</tr>
<tr>
<td></td>
<td>with no equation</td>
</tr>
</tbody>
</table>

Note. What about the handling (catching) of exceptions?
Effects as zooms (≡ spans of logics)

Following Moggi’s remark:

\[ p : A \rightarrow B \]

is interpreted as

\[ p : A \rightarrow T(B) \]

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++,...
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 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

Introduction
States
Diagrammatic logics
Exceptions
Conclusion
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

- DECORATED: $\Sigma_0$
- APPARENT: $\Sigma_1$
- EXPLICIT: $\Sigma_2$
The apparent specification

Notations

\[ \text{Loc} = \{ X, Y, \ldots \} = \text{the set of locations} \]
\[ 1 = \text{the unit type} \]

From the syntax we get the apparent equational specification \( \Sigma_1 \)
For each location \( i \in \text{Loc} \):

- a type \( V_i \) for the values of \( i \)
- \( \text{lookup} \quad l_i : 1 \rightarrow V_i \)
- \( \text{update} \quad u_i : V_i \rightarrow 1 \)
- and 2 equations

**EFFECT:** the intended semantics is not a model of \( \Sigma_1 \).
The explicit specification

Notation

\[ S = \text{the "type of states"} \]

From the semantics we get the explicit equational specification \( \Sigma_2 \)
For each location \( i \in \text{Loc} \):

- a type \( V_i \) for the values of \( i \)
- \( \{ \)
  - lookup \( l_i : S \rightarrow V_i \)
  - update \( u_i : V_i \times S \rightarrow S \)
- \( \} \)
- and 2 equations

**EFFECT**: the intended semantics is a model of \( \Sigma_2 \), *but* \( \Sigma_2 \) 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 $\Sigma_0$

For each location $i \in \text{Loc}$:
- a type $V_i$ for the values of $i$
- $\left\{\begin{array}{l}
\text{lookup } l_i^{(1)} : 1 \rightarrow V_i \\
\text{update } u_i^{(2)} : V_i \rightarrow 1
\end{array}\right.$
- and 2 equations
Three specifications

\[
\begin{align*}
\text{DECORATED: } \Sigma_0 \\
\quad l_i^{(1)} & : 1 \rightarrow V_i \\
\quad u_i^{(2)} & : V_i \rightarrow 1 \\
\text{2 equations}
\end{align*}
\]

\[
\begin{align*}
\text{APPARENT: } \Sigma_1 \\
\quad l_i & : 1 \rightarrow V_i \\
\quad u_i & : V_i \rightarrow 1 \\
\text{2 equations}
\end{align*}
\]

\[
\begin{align*}
\text{EXPLICIT: } \Sigma_2 \\
\quad l_i & : S \rightarrow V_i \\
\quad u_i & : V_i \times S \rightarrow S \\
\text{2 equations}
\end{align*}
\]

- \( F_1 \): 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

- **pure**
  
  \[
  X \xrightarrow{f^{(0)}} Y \]
  
  $F_2$:
  \[
  \Sigma_0 \xrightarrow{F_2} \Sigma_2
  \]

- **accessor**
  
  \[
  X \xrightarrow{f^{(1)}} Y \]
  
  $F_2$:
  \[
  X \xrightarrow{f^{(1)}} Y \xrightarrow{F_2} X \times S \xrightarrow{f} Y
  \]

- **modifier**
  
  \[
  X \xrightarrow{f^{(2)}} Y \]
  
  $F_2$:
  \[
  X \xrightarrow{f^{(2)}} Y \xrightarrow{F_2} X \times S \xrightarrow{f} Y \times S
  \]
Relevance of decorations

Claim. The decorated specification $\Sigma_0$ is “the most relevant”:

- both the apparent and the explicit specification may be recovered from $\Sigma_0$
- $\Sigma_0$ fits with the syntax (no type $S$)
- the intended semantics is a “decorated model” of $\Sigma_0$
- “decorated proofs” may be performed from $\Sigma_0$
A zoom for states

Claim. The 3 specifications are defined in 3 “logics” related by a “span of logics”:

\[
\begin{align*}
&\text{DECORATED: } L_0 \\
&\text{APPARENT: } L_1 \\
&\text{EXPLICIT: } L_2
\end{align*}
\]

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

We have designed an “abstract” category of logics
A diagrammatic logic is a functor \( L \)
with a full and faithful right adjoint \( R \) [...]

\[
\begin{array}{c}
S \\
\downarrow \quad \perp \\
R \\
\end{array}
\quad \xrightarrow{L} \quad
\begin{array}{c}
T \\
\end{array}
\]

- \( T \): category of theories
- \( S \): category of specifications
- \( \Sigma \) is a presentation of \( L(\Sigma) \) for every specification \( \Sigma \)

\( R \) full and faithful \( \iff \)
\( R(\Theta) \) is a presentation of \( \Theta \) for every theory \( \Theta \)
Models and proofs

With respect to a logic:

\[ \text{S} \xrightarrow{L} \text{T} \xleftarrow{R} \]

- A model \( M \) of a specification \( \Sigma \) with values in a theory \( \Theta \) is a morphism \( L\Sigma \rightarrow \Theta \) in \( \text{T} \), i.e., a morphism \( \Sigma \rightarrow R\Theta \) in \( \text{S} \)

[Gabriel-Zisman 1967] \( R \) is full and faithful \( \iff \)
(up to equiv.) \( L \) is a localization:
\( L \) makes some morphisms in \( \text{S} \) invertible in \( \text{T} \)

- A proof is a morphism in \( \text{T} \) [...]

Ex. Monadic equational logic

- \( \text{T} \): categories
- \( \text{S} \): “linear” sketches (＝ graphs with some composition)
Morphisms of logics

Based on arrow categories

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

This provides the category of diagrammatic logics
A zoom for states

- $L_1$ 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 $C^{(0)} \subseteq C^{(1)} \subseteq C^{(2)}$, with 1,...
- $F_1$ omits the decorations: it maps $C^{(0)} \subseteq C^{(1)} \subseteq C^{(2)}$ to $C^{(2)}$
- $F_2$ provides the meaning of the decorations
Outline

Introduction

States

Diagrammatic logics

Exceptions

Conclusion
Exceptions as dual of states?

Monads:

<table>
<thead>
<tr>
<th></th>
<th>( T(X) = (X \times St)^{St} )</th>
<th>( T(X) = X + Exc )</th>
</tr>
</thead>
<tbody>
<tr>
<td>states</td>
<td></td>
<td></td>
</tr>
<tr>
<td>exceptions</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lawvere theories:

<table>
<thead>
<tr>
<th></th>
<th>( lookup : Val \rightarrow Loc )</th>
</tr>
</thead>
<tbody>
<tr>
<td>states</td>
<td>( update : 1 \rightarrow Loc \times Val )</td>
</tr>
<tr>
<td></td>
<td>with 7 equations</td>
</tr>
<tr>
<td>exceptions</td>
<td>( raise_e : 0 \rightarrow 1 ) for ( e \in Exc )</td>
</tr>
<tr>
<td></td>
<td>with no equation</td>
</tr>
</tbody>
</table>
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

\( \textit{Etype} = \) the set of \textit{exceptional types} \\
\( P_i = \) the type of \textit{parameters} of type \( i \), for each \( i \in \textit{Etype} \) \\
\( 0 = \) the \textit{empty type} \\
\( E = \) the “type of \textit{exceptions}”

\[
\begin{align*}
\text{DECORATED: } \Sigma_0 & \\
& \begin{align*}
  r^{(1)}_i : P_i & \rightarrow 0 \\
  h^{(2)}_i : 0 & \rightarrow P_i
\end{align*}
\end{align*}
\]

\[
\begin{align*}
\text{APPARENT: } \Sigma_1 & \\
& \begin{align*}
  r_i : P_i & \rightarrow 0 \\
  h_i : 0 & \rightarrow P_i
\end{align*}
\end{align*}
\]

\[
\begin{align*}
\text{EXPLICIT: } \Sigma_2 & \\
& \begin{align*}
  r_i : P_i & \rightarrow E \\
  h_i : E & \rightarrow P_i + E
\end{align*}
\end{align*}
\]
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**

\[
\begin{array}{c}
X \xrightarrow{f^{(0)}} Y \\
\downarrow F_2 \\
X \xrightarrow{f} Y
\end{array}
\]

**propagator**

\[
\begin{array}{c}
X \xrightarrow{f^{(1)}} Y \\
\downarrow F_2 \\
X \xrightarrow{f} Y + E
\end{array}
\]

**handler**

\[
\begin{array}{c}
X \xrightarrow{f^{(2)}} Y \\
\downarrow F_2 \\
X + E \xrightarrow{f} Y + E
\end{array}
\]
Dual of states: a zoom for exceptions

$F_1$ \text{DECORATED: } L_0 \quad F_2$

\text{APPARENT: } L_1 \quad \text{EXPLICIT: } L_2

- $L_1$ 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 $C^{(0)} \subseteq C^{(1)} \subseteq C^{(2)}$, with $0,...$

- $F_1$ omits the decorations: it maps $C^{(0)} \subseteq C^{(1)} \subseteq C^{(2)}$ to $C^{(2)}$

- $F_2$ 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 $\text{raise}_{i,X}^{(1)}$ and $\text{handle}_{i,f,g}^{(1)}$ which are expanded as the usual operations $\text{raise}$ and $\text{handle}$
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 $\text{raise}_{i,X}^{(1)}$ and $\text{handle}_{i,f,g}^{(1)}$ which are expanded as the usual operations $\text{raise}$ and $\text{handle}$

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

<table>
<thead>
<tr>
<th>$r_i : P_i \rightarrow E$</th>
<th>$p \mapsto e = r_i(p)$</th>
</tr>
</thead>
</table>
| $h_i : E \rightarrow P_i + E$ | \(\left\{\begin{array}{l}
e = r_i(p) \mapsto p \\
e = r_j(p) \mapsto e \quad (j \neq i)\end{array}\right.\) |
Exceptions: encapsulation of $r_i^{(1)}$

In raising an exception, the empty type is hidden

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

- 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 $f^{(1)} : X \rightarrow Y$, using $g^{(1)} : P_i \rightarrow Y$:

- $f^{(1)}(x)$ 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 $e$

\[
\begin{array}{ccc}
X & \xrightarrow{f^{(1)}} & Y \\
& \downarrow{\text{id}^{(0)}} & \\
0 & \xrightarrow{h_i^{(2)}} & P_i \\
& \downarrow{\text{id}^{(0)}} & \\
Y & \xrightarrow{[\text{id}|g \circ h_i]^{(2)}} & Y
\end{array}
\]
Exceptions: encapsulation of $h_i^{(2)}$

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

- $f^{(1)}(x)$ 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 $e$

![Diagram](diagram.png)

- finally, this handler $[\text{id}\mid g \circ h_i]^{(2)} \circ f^{(1)}$ is encapsulated in a propagator $\text{handle}_{i, f, g}$
Outline

Introduction

States

Diagrammatic logics

Exceptions

Conclusion
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
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