States and exceptions considered as dual effects

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

LJK, University of Grenoble

March 27, 2011 – ACCAT 2011
dedicated to the memory of Prof. Jochen Pfalzgraf

Outline

Outline

1. Analyzing the semantics of exceptions yields
   a symmetry between states and exceptions
   at the semantics level.

2. States and exceptions are computational effects, but
   what is an effect?

3. Analyzing the syntax of exceptions as effects yields
   a symmetry between states and exceptions
   as computational effects.

Outline

Exceptions

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

- non-exceptional values
- exceptions

A function:

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

Exceptions: key operations

Exc = set of exceptions
$ExCstr$ = set of exception constructors

For each $i \in ExCstr$:

- Par_i = set of parameters
- $t_i : Par_i \to Exc$ = KEY throw function
- $c_i : Exc \to Par_i + Exc$ = KEY catch function

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

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

When $Exc = \sum_i Par_i$ with the key-throws as projections
this is an inductive definition of the key-catches

Exceptions: raise

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

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

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

Exceptions: handle

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

$$
\begin{tikzcd}
 & Par_i \arrow[d, "\subseteq"'] \arrow[dr, "g"] \arrow[dr, phantom, "=" ' near start] & \\
Exc \arrow[r, "c_i"] & Par_i + Exc \arrow[r] & Y + Exc \\
 & Exc \arrow[u, "\subseteq"] \arrow[ur, "\subseteq"'] \arrow[ur, phantom, "=" near start] &
\end{tikzcd}
$$

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

$$
\begin{tikzcd}
 & Y \arrow[d, "\subseteq"'] \arrow[dr, "\subseteq"] \arrow[dr, phantom, "=" ' near start] & \\
X \arrow[r, "f"] & Y + Exc \arrow[r] & Y + Exc \\
 & Exc \arrow[u, "\subseteq"] \arrow[ur, "catch\, i\, \{g\}"'] \arrow[ur, phantom, "=" near start] &
\end{tikzcd}
$$

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
- $u_i : Val_i \times St \to St$ = update function

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

When $St = \prod_i Val_i$ with the lookups as projections

this is a coinductive definition of the updates

Duality of semantics

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

So, there IS a duality between states and exceptions.

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

We will see that the duality of their semantics comes from a duality of states and exceptions seen as computational effects.

But. . .
what is a computational effect?

Outline

Monads for effects

[Moggi 1991]

The basic idea behind the categorical semantics of effects is that we distinguish the object A of values from the object TA of computations.

Programs of type B with a parameter of type A ought to be interpreted by morphisms with codomain TB,

but for their domain there are two alternatives, either A or TA.

We choose the first alternative, because it entails the second. Indeed computations of type A are the same as values of type TA.

a program: $A \to B$

is interpreted by a morphism: $A \to TB$

Monads for effects: exceptions

The monad of exceptions is $TA = A + Exc$.

A program of type B with a parameter of type A:

- throws an exception if it may map $x \in A$ to $e \in Exc$
- catches an exception if it may map $e \in Exc$ to $y \in B$

Monads for effects. A program of type B with a parameter of type A is interpreted by a morphism $A \to B + Exc$.

$\Longrightarrow$ it may throw an exception

$\Longrightarrow$ it cannot catch an exception

Second alternative. A program of type B with a parameter of type A is interpreted by a morphism $A + Exc \to B + Exc$.

$\Longrightarrow$ it may throw an exception

$\Longrightarrow$ it may catch an exception

What is an effect?

Claim. A computational effect is

an apparent lack of soundness.

There is a computational effect when:

- at first sight, the intended denotational semantics is not a model of the syntax,
- but the syntax may be "decorated" so as to recover soundness.

States as effect

The intended denotational 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 apparent syntax
but it is a model of the explicit syntax

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

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

Decorations for states

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
  - $f^{(2)} : X \to Y$ 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
$l : 1 \to V$
$u : V \to 1$
$l \circ u = id_V : V \to V$

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

Meaning of the decorations for states

The decorated syntax may be explicited

$$\begin{array}{lll} f^{(0)} : X \to Y & \text{as} & f : X \to Y \\ f^{(1)} : X \to Y & \text{as} & f : X \times S \to Y \\ f^{(2)} : X \to Y & \text{as} & f : X \times S \to Y \times S \end{array}$$

$$\begin{array}{lll} f =^{(sg)} g : X \to Y & \text{as} & f = g : X \times S \to Y \times S \\ f =^{(wk)} g : X \to Y & \text{as} & pr_Y \circ f = pr_Y \circ g : X \times S \to Y \end{array}$$

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

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

States as effect: decorations

Decorated

$l^{(1)} : 1 \to V$

$u^{(2)} : V \to 1$

$l \circ u =^{(wk)} id_V$

Apparent

$l : 1 \to V$

$u : V \to 1$

$l \circ u = id_V$

Explicit

$l : S \to V$

$u : V \times S \to S$

$l \circ u = pr_V$

Three syntaxes for one effect

$$\begin{tikzcd}[column sep=small]
 & \boxed{\text{Decorated}} \arrow[dl, mapsto] \arrow[dr, mapsto] & \\
\boxed{\text{Apparent}} & & \boxed{\text{Explicit}}
\end{tikzcd}$$

The intended semantics

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

A framewok 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
- categories of fractions
- limit sketches

Operations and equations

Our approach generalizes algebraic specifications

$\Longrightarrow$ it involves (decorated) operations and equations

handling exceptions is "symmetric" to updating states

The monads approach leads to Lawvere theories
for getting operations and equations [Plotkin&Power 2001] but

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

The approach of monads and Lawvere theories can be extended for handling exceptions

- with exception monads [Schroeder&Mossakowski 2004]
- with coalgebras [Levy 2006]
- with handlers [Plotkin&Pretnar 2009]

Outline

States

$$
\begin{tikzcd}
 & \boxed{\begin{array}{l} \text{Decorated} \\ \hline l^{(1)} : 1 \to V \\ u^{(2)} : V \to 1 \\ l \circ u =^{(wk)} id_V \end{array}} \arrow[dl, mapsto] \arrow[dr, mapsto] & \\
\boxed{\begin{array}{l} \text{Apparent} \\ \hline l : 1 \to V \\ u : V \to 1 \\ l \circ u = id_V \end{array}} & & \boxed{\begin{array}{l} \text{Explicit} \\ \hline l : S \to V \\ u : V \times S \to S \\ l \circ u = pr_V \end{array}}
\end{tikzcd}
$$

Exceptions: decorations

$f^{(0)} : X \to Y$	pure	$f : X \to Y$
$f^{(1)} : X \to Y$	thrower	$f : X \to Y + E$
$f^{(2)} : X \to Y$	catcher	$f : X + E \to Y + E$
$f =^{(sg)} g : X \to Y$	strong	$f = g : X + E \to Y + E$
$f =^{(wk)} g : X \to Y$	weak	$f \circ in_X = g \circ in_X : X \to Y + E$

Exceptions: key operations

Decorated

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

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

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

Apparent

$t : P \to 0$

$c : 0 \to P$

$c \circ t = id_P$

Explicit

$t : P \to E$

$c : E \to P + E$

$c \circ t = in_P$

Duality of effects

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

Outline

Conclusion

- An effect is an apparent lack of soundness
- a span of diagrammatic logics for each effect
- a new point of view on states
- a completely new point of view on exceptions
- a duality between states and exceptions

Future work

- combining effects
- operational semantics