About raising and handling exceptions

Dominique Duval and Jean-Claude Reynaud

University of Grenoble

WADT'06 — June 1., 2006

- Motivation
- Extensivity and case distinction
- Exceptions: syntax and deduction
- Exceptions: models
- Conclusion

Many different frameworks for exceptions!

- Gogolla, Drosten, Lipeck, Ehrich (1983)
- Bernot, Bidoit, Choppy (1986)
- Schobbens (1993)
- Benton, Hughes, Moggi (2002)
- Laird (2002)
- Plotkin, Power (2003)
- Walter, Schröder, Mossakowski (2005)
- and many others . . .

Yet another framework for exceptions...

The treatment of exceptions is viewed as a
generalized case distinction mechanism,

and the extensivity property of sums [Carboni, Lack, Walters 1993]
is used for dealing with case distinctions and with exceptions.

Focus on exceptions

Looking for "the simplest logics" for dealing with exceptions

$\Rightarrow$ many features are omitted, e.g.,

all functions are assumed univariate

since the issue of dealing with multivariate functions is not specific to exceptions: it is similar for most computational effects (however, see [Duval, Reynaud 2004])

- Motivation
- Extensivity and case distinction
- Exceptions: syntax and deduction
- Exceptions: models
- Conclusion

Distributivity (and extensivity)

A distributive category is a category with finite sums and products such that for each Y_1, Y_2 and Z:

$$Z \times Y_1 + Z \times Y_2 \cong Z \times (Y_1 + Y_2)$$

(dashed arrows stand for the projections from the products,
dotted arrows for the coprojections into the sums,
and "$\equiv$" for commutative squares):

$$\begin{tikzcd}
 & Z \times Y_1 \arrow[dl, dotted] \arrow[rr, dashed] \arrow[dr, phantom, "\equiv"] & & Y_1 \arrow[dl, dotted] \\
Z \times (Y_1 + Y_2) \arrow[rr, dashed] & & Y_1 + Y_2 & \\
 & Z \times Y_2 \arrow[ul, dotted] \arrow[rr, dashed] \arrow[ur, phantom, "\equiv"] & & Y_2 \arrow[ul, dotted]
\end{tikzcd}$$

(Distributivity and) extensivity

[Carboni, Lack, Walters 1993] An extensive category is a category with finite sums such that for each Y_1, Y_2, X and $u : X \to Y_1 + Y_2$:

$$
\begin{tikzcd}
 & u^{-1}(Y_1) \arrow[rr] \arrow[dl, dotted] \arrow[dr, phantom, "\equiv"] & & Y_1 \arrow[dl, dotted] \\
X \arrow[rr, "u"] & & Y_1 + Y_2 & \\
 & u^{-1}(Y_2) \arrow[rr] \arrow[ul, dotted] \arrow[ur, phantom, "\equiv"] & & Y_2 \arrow[ul, dotted]
\end{tikzcd}
$$

Theorem. An extensive category with finite products is distributive.

Sums and "matches"

A (binary) sum (of types) is made of
a vertex $Y_1 + Y_2$ and coprojections $j_i : Y_i \to Y_1 + Y_2$ such that for all:

$$
\begin{tikzcd}
 & Y_1 \arrow[dl, dotted, "j_1"'] \arrow[dr, "f_1"] & \\
Y_1 + Y_2 & & Z \\
 & Y_2 \arrow[ul, dotted, "j_2"] \arrow[ur, "f_2"'] &
\end{tikzcd}
$$

there is a unique match $[j_1 \Rightarrow f_1 \mid j_2 \Rightarrow f_2]$ (or $[f_1 \mid f_2]$):

$$
\begin{tikzcd}
 & Y_1 \arrow[dl, dotted, "j_1"'] \arrow[dr, "f_1"] \arrow[d, phantom, "\equiv"] & \\
Y_1 + Y_2 \arrow[rr, "{[f_1 \,|\, f_2]}" description] & {} & Z \\
 & Y_2 \arrow[ul, dotted, "j_2"] \arrow[ur, "f_2"'] \arrow[u, phantom, "\equiv"] &
\end{tikzcd}
$$

Extensivity and case distinction

The match “$[j_1 \Rightarrow f_1 \mid j_2 \rightarrow f_2]$” corresponds to:

“`if` $y \in Y_1$ `then` $f_1(y)$ `else` $f_2(y)$”

Let $u : X \rightarrow Y$, then “`case` u `of` $[j_1 \Rightarrow f_1 \mid j_2 \Rightarrow f_2]$” is defined as:

$$\texttt{case}\ u\ \texttt{of}\ [\, j_1 \Rightarrow f_1 \mid j_2 \Rightarrow f_2 \,] \ = \ [\, u^{-1}(j_1) \Rightarrow f_1 \mid u^{-1}(j_2) \Rightarrow f_2 \,]$$

$$\begin{tikzcd}
u^{-1}(Y_1) \arrow[r] \arrow[d, dotted, "u^{-1}(j_1)"'] \arrow[dr, phantom, "\equiv"] & Y_1 \arrow[d, dotted, "j_1"] \\
X \arrow[r, "u"] & Y_1 + Y_2 \\
u^{-1}(Y_2) \arrow[r] \arrow[u, dotted, "u^{-1}(j_2)"] \arrow[ur, phantom, "\equiv"] & Y_2 \arrow[u, dotted, "j_2"']
\end{tikzcd}
\qquad
\begin{tikzcd}
& u^{-1}(Y_1) \arrow[dl, dotted, "u^{-1}(j_1)"'] \arrow[dr, "f_1"] \arrow[d, phantom, "\equiv"] & \\
X \arrow[rr, "\texttt{case}\ u\ \texttt{of}\ [j_1 \Rightarrow f_1 \mid j_2 \Rightarrow f_2]"] & & Z \\
& u^{-1}(Y_2) \arrow[ul, dotted, "u^{-1}(j_2)"] \arrow[ur, "f_2"'] \arrow[u, phantom, "\equiv"] &
\end{tikzcd}$$

It corresponds to:

“`if` $u(y) \in Y_1$ `then` $f_1(y)$ `else` $f_2(y)$”

Example (1)

$\mathbb{N}$ is the set of naturals, and $\mathrm{succ} : \mathbb{N} \to \mathbb{N}$ the successor map.
The predecessor map $\mathrm{pred} : \mathbb{N} \to \mathbb{N}$ is defined as:

$$\mathrm{pred}(\mathrm{succ}(n)) = n \text{ and } \mathrm{pred}(0) = 0$$

Here is a specification "of naturals" Σ_{nat}:

$$\begin{tikzcd} U \arrow[r, "z"] & N \arrow[loop right, "s"] \end{tikzcd} \qquad \text{with a sum} \qquad \begin{tikzcd} U \arrow[r, dotted, "z"] & N & N \arrow[l, dotted, "s"'] \end{tikzcd}$$

and the model "of naturals" M_{nat} of Σ_{nat}:

$$\begin{tikzcd} \{*\} \arrow[r, "0"] & \mathbb{N} \arrow[loop right, "\mathrm{succ}"] \end{tikzcd}$$

A predecessor function can be generated from Σ_{nat}:

$$p = \texttt{case id of } [\, s \Rightarrow \texttt{id} \mid z \Rightarrow z \,] = [\, s \Rightarrow \texttt{id} \mid z \Rightarrow z \,] : N \to N \,.$$

- Motivation
- Extensivity and case distinction
- Exceptions: syntax and deduction
- Exceptions: models
- Conclusion

Three keywords for exceptions

The predecessor map $\text{pred} : \mathbb{N} \to \mathbb{N}$ can also be formalised in the following way, if some (SML-like) mechanism for exceptions is available:

First, an exception e is created:

$$\texttt{Exception}\ e$$

Then, a function $p' : N \to N$ is generated,
such that $p'(z)$ raises the exception e:

$$p'(x) = \texttt{case}\ x\ \texttt{of}\ [\, s(y) \Rightarrow y \mid z \Rightarrow \texttt{raise}\ e\,]$$

Finally, a function $p'' : N \to N$ is generated,
that calls p' and handles the exception e:

$$p''(x) = p'(x)\ \texttt{handle}\ [\, e \Rightarrow z\,]$$

Exception

The functions are decorated as follows
(this is borrowed from the monads [Moggi 1991]):
— a value is a function that does not raise any exception
— a computation is a function that may raise an exception
(so, every value can be coerced to a computation)

Let 0 be the empty sum, both for values and for computations.

Definition. An exception is a computation with type 0:

$$e^c : P \to 0$$

Example (2)

It generates a predecessor value:

$$p^v = (\texttt{case id of}\ [\, s \Rightarrow \texttt{id} \mid z \Rightarrow z\,])^v = [\, s \Rightarrow \texttt{id} \mid z \Rightarrow z\,]^v : N \to N$$

so that $p.s \equiv^v$ `id` and $p.z \equiv^v z$.

`raise`

Claim. When a function $f : X \to Y$ *raises* an exception e, this means that the exception e (of type 0) can be viewed as an expression of type Y.

Definition. The keyword `raise` constructs a value ${\tt raise}_Y{}^v$ for every type Y:

$$\texttt{raise}_Y{}^v = [\,]_Y{}^v : 0 \longrightarrow Y \ .$$

Let $e^c : P \to 0$ be an exception and Y a type.
To raise the exception e^c in the type Y is to build:

$$(\texttt{raise}_Y.e)^c : P \longrightarrow Y \ .$$

Theorem. The exceptions do propagate:
let $e^c : P \to 0$ and $g^c : Y \to Z$, then

$$g.\texttt{raise}_Y.e \equiv^c \texttt{raise}_Z.e \ .$$

Example (3)

$\Sigma_{\mathrm{nat,deco}}$:

$$\begin{tikzcd} U \arrow[r, dotted, "z^v"] \arrow[dr, "e^c"] & N \arrow[loop right, dotted, "s^v"] \\ & 0 \end{tikzcd}$$

generates a predecessor computation:

$$p'^{c} = (\,\texttt{case}\ \texttt{id}\ \texttt{of}\ [\, s \Rightarrow \texttt{id} \mid z \Rightarrow \texttt{raise}.e\,]\,)^{c}$$

$$= [\, s \Rightarrow \texttt{id} \mid z \Rightarrow \texttt{raise}.e\,]^{c} : N \to N$$

so that $p'.s \equiv^{c} \texttt{id}$ and $p'.z \equiv^{c} \texttt{raise}.e$.

`handle`

The keyword `handle` has two arguments.
For instance, "p' `handle` $[\, e \Rightarrow z \,]$" has arguments p' and $[\, e \Rightarrow z \,]$.

There are two nested cases in a handling expression "f `handle` g":
— the first one tests whether f raises an exception,
— when true, the second one tests which is the raised exception.

The handling construction is defined from these two kinds of "cases",
which correspond to two decorations of extensivity,
on top of the "ordinary" decoration of extensivity.

1st decoration of extensivity

For "ordinary" case distinction, when $u^v : X \to Y_1 + Y_2$ is a value:

$$
\begin{tikzcd}
 & u^{-1}(Y_1) \arrow[rr, "u_1^v"] \arrow[dl, dotted, "u^{-1}(j_1)^v"'] \arrow[drr, phantom, "\equiv"] & & Y_1 \arrow[dl, dotted, "j_1^v"] \\
X \arrow[rr, "u^v"] & & Y_1 + Y_2 & \\
 & u^{-1}(Y_2) \arrow[rr, "u_2^v"'] \arrow[ul, dotted, "u^{-1}(j_2)^v"] \arrow[urr, phantom, "\equiv"] & & Y_2 \arrow[ul, dotted, "j_2^v"']
\end{tikzcd}
$$

2nd decoration of extensivity

For testing whether a computation $u^c : X \to Y$ raises an exception:

$$\begin{tikzcd}
 & X_{u,1} \arrow[rr, "u_1^v"] \arrow[dl, dotted, "u^{-1}(\mathtt{id})^v"'] \arrow[dr, phantom, "\equiv"] & & Y \arrow[dl, dotted, "\mathtt{id}_Y^v"] \\
X \arrow[rr, "u^c"] & & Y & \\
 & X_{u,0} \arrow[rr, "u_0^c"'] \arrow[ul, dotted, "u^{-1}(\mathtt{raise})^v"] \arrow[ur, phantom, "\equiv"] & & 0 \arrow[ul, dotted, "\mathtt{raise}_Y^v"']
\end{tikzcd}$$

3rd decoration of extensivity

For testing which exception has been raised:

$$
\begin{tikzcd}
 & u^{-1}(P_1) \arrow[rr, "u_1^v"] \arrow[dl, dotted, "u^{-1}(e_1)^v"'] \arrow[dr, phantom, "\equiv"] & & P_1 \arrow[dl, dotted, "e_1^c"] \\
X \arrow[rr, "u^c" description] & & 0 & \\
 & u^{-1}(P_2) \arrow[rr, "u_2^v"'] \arrow[ul, dotted, "u^{-1}(e_2)^v"] \arrow[ur, phantom, "\equiv"] & & P_2 \arrow[ul, dotted, "e_2^c"']
\end{tikzcd}
$$

`handle` (more precisely)

The handling construction is defined from the 2nd and 3rd decorations of extensivity:

$$(u\;\mathtt{handle}\;[\,e_i \Rightarrow f_i\,]_{i\in I})^c$$

$$= (\mathtt{case}^2\;u\;\mathtt{of}\;[\,\mathtt{id}_Y \Rightarrow u_1 \mid \mathtt{raise}_Y \Rightarrow f\,])^c : X \longrightarrow Y\;,$$

where f is the computation:

$$f^c = (\mathtt{case}^3\;u_0\;\mathtt{of}\;[\,e_i \Rightarrow f_i\,]_{i\in I})^c : X_{u,0} \longrightarrow Y\;.$$

Example (4)

We have yet a predecessor value:

$$p^v = [\, s \Rightarrow \mathtt{id} \mid z \Rightarrow z \,] : N \to N$$

and a predecessor computation:

$$p'^c = [\, s \Rightarrow \mathtt{id} \mid z \Rightarrow \mathtt{raise}.e \,] : N \to N$$

Now, we get another predecessor computation:

$$p''^c = p' \ \mathtt{handle} \ [\, e \Rightarrow z \,] : N \to N$$

Using the rules of the decorated logic, it can be proved that $p'' \equiv^c p$.

It follows that p'', actually, never raises an exception.

- Motivation
- Extensivity and case distinction
- Exceptions: syntax and deduction
- Exceptions: models
- Conclusion

The expansion of the decorations

The expansion gets rid of the decorations.
(this is borrowed from the monads [Moggi 1991]).

Definition. Let E be a distinguished type;
— the expansion of a value $f^v : X \to Y$
is a function $f : X \to Y$,
— the expansion of a computation $f^c : X \to Y$
is a function $f : X \to Y + E$.

Models

Let Σ_{deco} be a decorated specification, and Σ_{expl} its expansion.

Definition. A model of Σ_{deco} is a model of Σ_{expl}.

Theorem. The decorated deduction rules are sound with respect to the explicit models.

This means that if $f \equiv^v g$ (for values) or $f \equiv^c g$ (for computations) in Σ_{deco}, then $M(f) = M(g)$ in every model of Σ_{expl}.

A proof of this result relies upon the fact that the expansion is a morphism between diagrammatic logics [Duval, Reynaud 2004].

Example (5)

The expansion $\Sigma_{\mathrm{nat,expl}}$ of $\Sigma_{\mathrm{nat,deco}}$ is Σ_{nat} with a function $e : U \to E$:

$$\begin{tikzcd} U \arrow[r, dotted, "z"] \arrow[dr, "e"] & N \arrow[loop right, dotted, "s"] \\ & E \end{tikzcd}$$

Then $\Sigma_{\mathrm{nat,expl}}$ has a model $M_{\mathrm{nat,expl}}$:

$$\begin{tikzcd} \{*\} \arrow[r, dotted, "0"] \arrow[dr, "\varepsilon"] & \mathbb{N} \arrow[loop right, dotted, "\mathrm{succ}"] \\ & \{\varepsilon\} \end{tikzcd}$$

In this model, both the functions p and p'' are interpreted as the predecessor map $\mathrm{pred} : \mathbb{N} \to \mathbb{N}$.

- Motivation
- Extensivity and case distinction
- Exceptions: syntax and deduction
- Exceptions: models
- Conclusion

What has been done:

The deduction system of a language with exceptions
(without any explicit "type of exceptions")
and its set-valued interpretation
(with an explicit "set of exceptions")
are related by a morphism of diagrammatic logics.

What has to be done:

This approach has to be embedded into some model of computation:
maybe distributive computability? [Walters 1992, Vigna 1995]