Diagrammatic logic and effects: 
the example of exceptions

Dominique Duval
joint work with Christian Lair and Jean-Claude Reynaud

MAP — january 13, 2005
Logic and categories

- Logic can be based on category theory: (Lawvere, Ehresmann, 1960’s).

- One major result: simply typed lambda-calculus is equivalent to cartesian closed categories (Lambek and Scott).

- Many other results: “some logic is equivalent to some family of categories”.

For dealing (also) with computational effects, such as exceptions, overloading, . . ., several logics are needed – schematically, at least:

— a logic (with effects) for the language,

— a logic (explicit) for the user.

Thus, a category of logics is needed.
This talk

1. The category of diagrammatic logics
   (also used by César Domínguez in the next talk).

2. An application to exceptions
   (involving three diagrammatic logics).
1. The category of diagrammatic logics
Graphs

A (directed multi-)graph is made of:

- a set of points, a set of arrows,
- source and target maps, both from the arrows to the points.

Example:

\[ u \xrightarrow{0} g \xleftarrow{+} g^2 \]

The definition of graphs can be illustrated by the following graph \( S_{Gr} \):
Categories

A category is a graph where:

- each point $X$ has an identity arrow $\text{id}_X : X \to X$,
- each pair of consecutive arrows $f : X \to Y$, $g : Y \to Z$ has a composed arrow $g \circ f : X \to Z$,
- with the usual associativity and unitarity axioms.

Basic examples.

**Set**: points are sets and arrows are maps.

**Gr**: points are graphs and arrows are morphisms of graphs.

**Cat**: points are categories and arrows are functors.
The definition of categories can be illustrated by the following graph $S_{\text{Cat}}$:

with the additional information that the cone below represents a limit:

and with several equalities of maps:

$$\text{source} \circ \text{comp} = \text{source} \circ \text{first}, \text{target} \circ \text{comp} = \text{target} \circ \text{second}, \ldots$$

A graph together with some distinguished cones (potential limits) and with some equalities among terms, is a **projective sketch**.
Graphs and categories

From the definition, every category is a graph; there is an omitting (or forgetful) functor:

\[
\begin{array}{ccc}
\text{Gr} & \xrightarrow{U} & \text{Cat} \\
\end{array}
\]

which corresponds to the inclusion of projective sketches:

\[
\begin{array}{ccc}
\mathcal{S}_\text{Gr} & \subseteq & \mathcal{S}_\text{Cat} \\
\end{array}
\]

On the other hand, every graph generates a category; there is a generating functor:

\[
\begin{array}{ccc}
\text{Gr} & \xrightarrow{F} & \text{Cat} \\
\end{array}
\]

The pair \((F, U)\) is an adjunction:

\[
\text{Hom}_{\text{Gr}}(\Gamma, U\Delta) \cong \text{Hom}_{\text{Cat}}(F\Gamma, \Delta) .
\]
Composative graphs

A compositive graph is a graph where:

- *some* points have an identity arrow,
- *some* pairs of consecutive arrows have a composed arrow.

This definition can be illustrated by a projective sketch $\mathcal{S}_{\text{Comp}}$:

$$
\begin{array}{ccc}
\mathcal{S}_{\text{Gr}} & \longrightarrow & \mathcal{S}_{\text{Comp}} & \longrightarrow & \mathcal{S}_{\text{Cat}} \\
\end{array}
$$

So, there are adjunctions:

$$
\begin{array}{ccc}
\text{Gr} & \overset{F'}{\longrightarrow} & \text{Comp} & \overset{F''}{\longrightarrow} & \text{Cat} \\
\overset{U'}{\longleftarrow} & & \overset{U''}{\longleftarrow} & \\
\end{array}
$$

Moreover, for every graph $\Gamma$ and every category $\Delta$:

$$
U'F'\Gamma \cong \Gamma \text{ and } F''U''\Delta \cong \Delta :
$$

an instance of the decomposition theorem (Duval, Lair, 2002).
Propagators

The realizations of a projective sketch $\mathcal{S}$ are the morphisms from $\mathcal{S}$ to Set, they form a category $\text{Real}(\mathcal{S})$. E.g., $\text{Real}(\mathcal{S}_{\text{Gr}}) = \text{Gr}$, \ldots 

Every morphism of projective sketches $M : \mathcal{S} \rightarrow \mathcal{S}'$ defines an omitting functor:

$$\text{Real}(\mathcal{S}) \xrightarrow{U_M} \text{Real}(\mathcal{S}')$$

**Theorem** (*Ehresmann*, 1965) The functor $U_M$ has a left-adjoint:

$$\text{Real}(\mathcal{S}) \xleftrightarrow{F_M} \text{Real}(\mathcal{S}')$$

A propagator is a morphism of projective sketches $P : \mathcal{S} \rightarrow \overline{\mathcal{S}}$ such that, for each realization $\Delta$ of $\overline{\mathcal{S}}$:

$$F_P U_P \Delta \cong \Delta.$$
Diagrammatic logics

Let $P : \mathcal{S} \to \overline{\mathcal{S}}$ be a propagator.

- **$P$-specifications**: $\text{Spec}(P) = \text{Real}(\mathcal{S})$.
- **$P$-domains**: $\text{Dom}(P) = \text{Real}(\overline{\mathcal{S}})$.
- **$P$-deduction rules**: arrows from $\overline{\mathcal{S}}$.
- **$P$-deduction steps** are morphisms $\sigma : \Sigma \to \Sigma'$ such that $F_P(\sigma)$ is an isomorphism (see next slides).
- **$P$-models** of a specification $\Sigma$ with values in a domain $\Delta$:
  $\text{Mod}(\Sigma, \Delta) = \text{Hom}(\Gamma, U_P \Delta) \cong \text{Hom}(F_P \Gamma, \Delta)$.

**Soundness.** If $\sigma$ is a deduction step, then $\text{Mod}(\sigma, \Delta)$ is a bijection.
Deduction rules

**Theorem** (*Hébert, Adámek and Rosický, 2001*)
A propagator consists of adding inverses to arrows.

The corresponding rules $\frac{H}{C}$ are illustrated as follows (→ only in $\overline{\mathcal{S}}$):

\[ H \xleftarrow{s} C \]

Or via the Yoneda contravariant morphism (→ only in $\text{Dom}(P)$):

\[ Y \mathcal{S}(H) \xrightarrow{Y \mathcal{S}(s)} Y \mathcal{S}(C) \]

Example:

\[ X \xrightarrow{f} Y \xrightarrow{g} Z \xleftarrow{} \]

\[ X \xrightarrow{f} Y \xrightarrow{g \circ f} Z \]

\[ Y \mathcal{S}(H) \xrightarrow{Y \mathcal{S}(s)} Y \mathcal{S}(C) \]
Deduction steps

Let $\Sigma$ be a $P$-specification. The $P$-deduction step associated to the $P$-deduction rule $H \xleftarrow{s} C$, applied to an $x \in \Sigma(H)$, is the morphism $\tau_s(x)$ in the pushout of $Y(s)$ and $x$:

$$
\begin{array}{ccc}
Y(H) & \xrightarrow{Y(s)} & Y(C) \\
\downarrow{x} & & \downarrow{c_s(x)} \\
\Sigma & \xrightarrow{\tau_s(x)} & \Sigma_s(x)
\end{array}
$$

**Proposition.** $F_P(\tau_s(x))$ is an isomorphism.
Morphisms of diagrammatic logics

Let $P_1 : S_1 \to \overline{S}_1$ and $P_2 : S_2 \to \overline{S}_2$ be two propagators. A morphism of propagators $P_1 \to P_2$ is a pair $(\alpha, \overline{\alpha})$ of morphisms of projective sketches such that:

\[
\begin{array}{ccc}
S_1 & \xrightarrow{P_1} & \overline{S}_1 \\
\downarrow{\alpha} & = & \downarrow{\overline{\alpha}} \\
S_2 & \xrightarrow{P_2} & \overline{S}_2
\end{array}
\]

In this way, we get a category of diagrammatic logics, as required.
2. An application to exceptions
The issue

Formalizing the exception mechanism.

Previous work:
algebraic specifications, monads (Moggi, 1996), (Haskell).
Plotkin and Power, 2001: “Evident further work is to consider how other operations such as those for handling exceptions should be modelled. That might involve going beyond monads, as Moggi has suggested to us.”

Our approach is influenced by the monads approach, although quite different.
Three logics
Three denotational semantics

\[ P_{\text{deco}}: \text{decorated logic} \]
\[ \text{direct semantics} \]
\[ \text{undecoration} \]
\[ \delta \]
\[ \text{expansion} \]
\[ \chi \]
\[ P_{\text{basic}}: \text{basic logic} \]
\[ \text{naïve semantics} \]
\[ P_{\text{expl}}: \text{explicit logic} \]
\[ \text{monadic semantics} \]

Example, over the naturals, with \( z \) (for 0) and \( s \) (for successor):

Exception \( e \)

\[ p(x) = \text{case } x \text{ of } [ s(y) \Rightarrow y \mid z \Rightarrow \text{raise } e ] \text{ handle } [ e \Rightarrow z ] \]
The basic logic

Without any exceptions.
With sum types.

\[ p_0(x) = \text{case } x \text{ of } [ s(y) \Rightarrow y \mid z \Rightarrow z ] : \text{Nat} \rightarrow \text{Nat} \]
**The case construction**

More generally, the case construction uses the extensivity property of sums (Carboni, Lack and Walters, 1993):

\[
    t(x) = \text{case } u(x) \text{ of } [ j_1(y) \Rightarrow t_1(x,y) \mid j_2(y) \Rightarrow t_2(x,y) ] \\
    = [ i_1(x) \Rightarrow t_1(x,y) \mid i_2(x) \Rightarrow t_2(x,y) ]
\]
**The decorated logic:** raise

Exception $e$

$$1 \xrightarrow{e^c} 0$$

$p_1(x) = \text{case } x \text{ of } [ s(y) \Rightarrow y | z \Rightarrow \text{raise } e ]$

![Diagram of decorated logic](image)

where:

$$\text{raise}^v_X = [ \_ ]^v_X : 0 \to X.$$
The decorated logic: handle

\[ p(x) = p_1(x) \text{ handle } [e \Rightarrow z] \]
\[ = \text{case}^c p_1 \text{ of } [\text{id} \Rightarrow \text{id} \mid \text{raise} \Rightarrow u] \]
where \( u = \text{case}^e e \text{ of } [e \Rightarrow z] \)

So, \( p^c \equiv [s \Rightarrow \text{id} \mid z \Rightarrow u] \)
and on the other hand \( u^c \equiv z \).
Finally:
\[ p \equiv [s \Rightarrow \text{id} \mid z \Rightarrow z] \equiv p_0. \]
The decorated logic: proofs and models

- Proofs can be made in the decorated logic.
- Models can be defined in the decorated logic, although there is no canonical interesting domain of sets.
- Decorations for arrows: \( v, c \),
  this is rather similar to the monads approach.
- Decorations for the case rule: \( v, c, e \),
  this is new.
- The morphism \( \delta : P_{\text{deco}} \to P_{\text{basic}} \) is simply the undecoration.
The explicit logic

Similar to the basic logic, but with a distinguished type $E$ for exceptions: “exceptions are explicit”.

A (partial) description of the expansion morphism $\chi : P_{\text{deco}} \to P_{\text{expl}}$:

- A value $t^v : X \to Y$ becomes a term $t : X \to Y$.

- A computation $t^c : X \to Y$ becomes a term $t : X \to Y + E$. For example, an exception $e^c : 1 \to 0$ becomes $e : 1 \to E$.

- The composition of these terms is done in the Kleisli way, as in the monads approach.
Conclusion

**Mathematics:**
categories, adjunction, sketches,…

**Algorithms:**
can be formalized, even when they are not functional.

**Proofs:**
a framework for proofs of programs using effects.