Diagrammatic logic and exceptions: an introduction

D. Duval[1] and J.-C. Reynaud[2]

[1] Laboratoire de Modélisation et Calcul LMC-IMAG,
B.P. 53, 38041 Grenoble Cedex 9, France.
Dominique.Duval@imag.fr
[2] Laboratoire Logiciels Systèmes Réseaux LSR-IMAG,
BP. 72, 38402 Saint Martin d'Hères Cedex, France.
Jean-Claude.Reynaud@imag.fr

Abstract. For dealing with computational effects in computer science, it may be helpful to use several logics: typically, a logic with implicit effects for the language, and a more classical logic for the user. Hence, the study of computational effects should take place in a framework where distinct logics can be related. In this paper, such a framework is presented: it is a category, called the category of *propagators*. Each propagator defines a kind of logic, called a *diagrammatic logic*, which is endowed with a deduction system and a sound notion of models. Morphisms of propagators provide the required relationships between diagrammatic logics. The category of propagators has been introduced by Duval and Lair in 2002, it is based on the notion of *sketches*, which is due to Ehresmann in the 1960's. Then, the paper outlines how Duval and Reynaud in 2004 used the category of propagators for dealing with the computational effect of raising and handling *exceptions*. Another application of diagrammatic logic is presented by Domínguez *et al.* in the same conference.

1 Introduction

It is known from the pioneering work of Lawvere and Ehresmann in the 1960's that logic can be based on category theory. One major result of categorical logic is that simply typed lambda-calculus is equivalent, in some sense, to cartesian closed categories [11]. Similarly, many other results can be stated as "some logic is equivalent to some family of categories".

For dealing with computational effects, such as exceptions, overloading, state,..., it may be helpful to use several logics – schematically, at least, a logic (with implicit effects) for the language, and another logic (where the effects are made explicit) for the user. Thus, a *category of logics* is needed. In section 2, the category of *diagrammatic logics* is presented. In section 3, an application to *exceptions*, involving two morphisms of diagrammatic logics, is outlined. Moreover, diagrammatic logics are also used in [3].

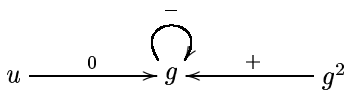
Dagstuhl Seminar Proceedings 05021
Mathematics, Algorithms, Proofs
http://drops.dagstuhl.de/opus/volltexte/2006/293

2 The category of diagrammatic logics

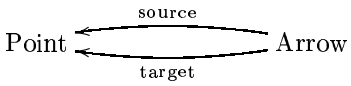
Let us begin with a well-known notion.

Definition 1. *A* (directed multi-)graph *is made of a set of* points*, a set of* arrows*, and* source *and* target *maps, both from the arrows to the points. An arrow f with source X and target Y is denoted $f : X \to Y$ or $X \xrightarrow{f} Y$.*

Example 1. The following graph illustrates a signature of groups, with one sort g, a binary operation $+ : g^2 \to g$, where g^2 stands for the list of sorts $g\,g$, a constant $0 : u \to g$, where u stands for the empty list of sorts, and a unary operation $- : g \to g$.



Example 2. Let us now look at a "meta" example: the definition of graphs can itself be illustrated by the following graph $\mathcal{S}_{\mathrm{Gr}}$:

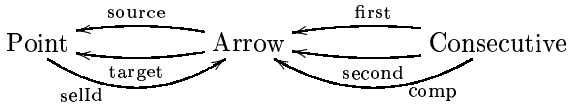


Definition 2. *A* category *is a graph where each point X has an* identity *arrow $X \xrightarrow{\mathrm{id}_X} X$, each pair of consecutive arrows $X \xrightarrow{f} Y \xrightarrow{g} Z$ has a* composed *arrow $X \xrightarrow{g \circ f} Z$, with the usual associativity and unitarity axioms.*

A fundamental reference for category theory is [12]. Some basic categories are:

- 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*.

Example 3. The definition of categories can also be illustrated by a graph $\mathcal{S}_{\mathrm{Cat},0}$:



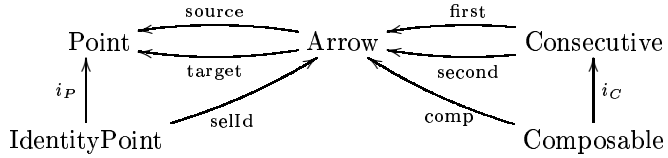
However, this graph illustrates only part of the definition. For instance, it does not mention the fact that the point Consecutive stands precisely for the set of pairs of consecutive arrows. This will be improved later.

Let us now introduce an intermediate notion between graphs and categories.

Definition 3. *A* compositive graph *is a graph where* some *points X have an identity arrow $X \xrightarrow{\mathrm{id}_X} X$,* some *pairs of consecutive arrows $X \xrightarrow{f} Y \xrightarrow{g} Z$ have a composed arrow $X \xrightarrow{g \circ f} Z$.*

The compositive graphs with their morphisms form another category Comp, which is "between" the categories Gr and Cat, in the sense that every category is a compositive graph and every compositive graph is a graph.

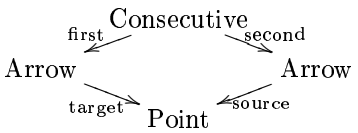
Example 4. The definition of compositive graphs can also be partially illustrated by a graph $\mathcal{S}_{\mathrm{Comp},0}$:



The notion of compositive graphs is now enriched, in order to get a better illustration of the categories Cat and Comp.

Definition 4. *A* projective sketch *is a compositive graph together with some* distinguished cones *(or potential limits).*

Example 5. The definition of categories can be illustrated by the projective sketch $\mathcal{S}_{\mathrm{Cat}}$ made of the graph $\mathcal{S}_{\mathrm{Cat},0}$ as above, together with one distinguished cone, which states that the point Consecutive stands precisely for the set of pairs of consecutive arrows:



and with several equalities involving identities and composed arrows:

$$\mathrm{source} \circ \mathrm{selId} = \mathrm{id}_{\mathrm{Point}} \ , \ \mathrm{target} \circ \mathrm{selId} = \mathrm{id}_{\mathrm{Point}} \ ,$$

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

Similarly, distinguished cones and equalities of arrows can be added to the compositive graph $\mathcal{S}_{\mathrm{Comp},0}$, in order to get a projective sketch $\mathcal{S}_{\mathrm{Comp}}$ which illustrates the definition of compositive graphs.
Of course, the graph $\mathcal{S}_{\mathrm{Gr}}$ is a projective sketch.

Definition 5. *The* realizations *of a projective sketch $\mathcal{S}$ are the morphims from $\mathcal{S}$ to* Set*, they form a category* $\mathrm{Real}(\mathcal{S})$.

Example 6. $\mathrm{Real}(\mathcal{S}_{\mathrm{Gr}}) = \mathrm{Gr}$ and (up to equivalence) $\mathrm{Real}(\mathcal{S}_{\mathrm{Comp}}) = \mathrm{Comp}$ and $\mathrm{Real}(\mathcal{S}_{\mathrm{Cat}}) = \mathrm{Cat}$.

The projective sketches with their morphisms form a category, and every morphism of projective sketches $M : \mathcal{S} \to \mathcal{S}'$ gives rise to a functor $\mathrm{U}_M : \mathrm{Real}(\mathcal{S}') \to \mathrm{Real}(\mathcal{S})$, called the *omitting (or forgetful) functor* with respect to M. The following major theorem is due to Ehresmann [7].

Theorem 1. *The functor* $\mathrm{U}_M : \mathrm{Real}(\mathcal{S}') \to \mathrm{Real}(\mathcal{S})$ *has a left adjoint* $\mathrm{F}_M : \mathrm{Real}(\mathcal{S}) \to \mathrm{Real}(\mathcal{S}')$*, called the* freely generating functor *with respect to* M*.*

$$\mathrm{Real}(\mathcal{S}) \underset{\mathrm{U}_M}{\overset{\mathrm{F}_M}{\rightleftarrows}} \mathrm{Real}(\mathcal{S}')$$

This theorem implies that there is natural bijection, for all Γ in $\mathcal{S}$ and Δ in $\mathcal{S}'$:

$$\mathrm{Hom}_{\mathcal{S}}(\Gamma, \mathrm{U}\Delta) \cong \mathrm{Hom}_{\mathcal{S}'}(\mathrm{F}\Gamma, \Delta) \ .$$

Example 7. The enrichment from $\mathcal{S}_{\mathrm{Gr}}$ to $\mathcal{S}_{\mathrm{Cat}}$ is a morphism of projective sketches:

$$\mathcal{S}_{\mathrm{Gr}} \to \mathcal{S}_{\mathrm{Cat}}$$

which gives rise to an adjunction pair:

$$\mathrm{Gr} \underset{\mathrm{U}}{\overset{\mathrm{F}}{\rightleftarrows}} \mathrm{Cat}$$

such that:

$$\mathrm{Hom}_{\mathrm{Gr}}(\Gamma, \mathrm{U}\Delta) \cong \mathrm{Hom}_{\mathrm{Cat}}(\mathrm{F}\Gamma, \Delta) \ .$$

The omitting functor maps each category to its underlying graph, and the freely generating functor maps each graph to its freely generated category. It is worth noting that, for most graphs Γ and categories Δ:

$$\mathrm{UF}\Gamma \ncong \Gamma \text{ and } \mathrm{FU}\Delta \ncong \Delta \ :$$

Example 8. The enrichment from $\mathcal{S}_{\mathrm{Gr}}$ to $\mathcal{S}_{\mathrm{Cat}}$ can be decomposed in two morphisms of projectives sketches:

$$\mathcal{S}_{\mathrm{Gr}} \to \mathcal{S}_{\mathrm{Comp}} \to \mathcal{S}_{\mathrm{Cat}}$$

The morphism from $\mathcal{S}_{\mathrm{Gr}}$ to $\mathcal{S}_{\mathrm{Comp}}$ is the enrichment, while the morphism P_{Comp} from $\mathcal{S}_{\mathrm{Comp}}$ to $\mathcal{S}_{\mathrm{Cat}}$ maps both Point and IdentityPoint to Point and i_P to $\mathrm{id}_{\mathrm{Point}}$, and similarly it maps both Consecutive and Composable to Consecutive and i_C to $\mathrm{id}_{\mathrm{Consecutive}}$. These morphisms give rise to two adjunction pairs:

$$\mathrm{Gr} \underset{\mathrm{U}'}{\overset{\mathrm{F}'}{\rightleftarrows}} \mathrm{Comp} \underset{\mathrm{U}''}{\overset{\mathrm{F}''}{\rightleftarrows}} \mathrm{Cat}$$

Moreover, for every graph Γ and every category Δ:

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

This property of compositive graphs is an instance of the *decomposition theorem*, which is stated now, and which is proved in an effective way in [5, 4].

Theorem 2. *Let $M : \mathcal{S}_0 \to \overline{\mathcal{S}}$ be a morphism of projective sketches. There are a projective sketch $\mathcal{S}$ and two morphisms $D : \mathcal{S}_0 \to \mathcal{S}$ and $P : \mathcal{S} \to \overline{\mathcal{S}}$, such that (up to equivalence) $M = P \circ D$ and both functors $\mathrm{U}_D \circ \mathrm{F}_D$ and $\mathrm{F}_P \circ \mathrm{U}_P$ are identities.*

The morphisms like $P : \mathcal{S} \to \overline{\mathcal{S}}$ in this theorem, play a basic rôle in the definition of diagrammatic logics. Such a P describes a logic: roughly speaking, the projective sketch $\mathcal{S}$ describes the syntax for expressing the specifications (i.e., the sets of axioms), while the morphism P corresponds to the rules of the logic, and the projective sketch $\overline{\mathcal{S}}$ states the properties that have to be satisfied by the theories generated from the specifications (i.e., by the sets of theorems), and by the models. The fact that $\mathrm{F}_P \circ \mathrm{U}_P$ ensures that every theorem that has been proved from the axioms can be added to the specification without changing its meaning. For this reason, such morphisms are given a name.

Definition 6. *A* propagator *is a morphism of projective sketches $P : \mathcal{S} \to \overline{\mathcal{S}}$ such that (up to equivalence) the functor $\mathrm{F}_P \circ \mathrm{U}_P$ is the identity of* $\mathrm{Real}(\overline{\mathcal{S}})$.

Example 9. The morphism $P_{\mathrm{Comp}} : \mathcal{S}_{\mathrm{Comp}} \to \mathcal{S}_{\mathrm{Cat}}$ is a propagator. In this basic example, the "axioms" are simply operations, and the "theorems" are the terms which are generated from the operations. The propagator P_{Comp} can be enriched in order to give rise to the equational logic [4]; then the axioms are equations and the theorems are all the equations in the congruence which is generated from the axioms.

Propagators can be described by the following result [9].

Theorem 3. *A propagator consists (up to equivalence) of adding inverses to arrows.*

Now, it is quite easy to define the *diagrammatic logic* which is associated to a propagator.

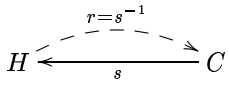
Definition 7. *Let $P : \mathcal{S} \to \overline{\mathcal{S}}$ be a propagator.*

- *The category of* P-specifications *is the category of realizations of $\mathcal{S}$:* $\mathrm{Spec}(P) = \mathrm{Real}(\mathcal{S})$.
- *The category of* P-domains *is the category of realizations of $\overline{\mathcal{S}}$:* $\mathrm{Dom}(P) = \mathrm{Real}(\overline{\mathcal{S}})$.
- *The* P-deduction rules *are the arrows in $\overline{\mathcal{S}}$.*
- *The* P-entailments *are the morphisms of P-specifications σ such that $\mathrm{F}_P(\sigma)$ is an isomorphim of P-domains.*
- *The set of* P-models *of a P-specification Σ with values in a P-domain Δ is (using the bijection that comes from the adjunction):*

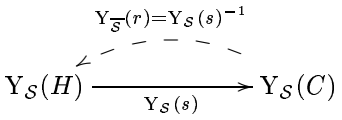
$$\mathrm{Mod}(\Sigma, \Delta) = \mathrm{Hom}(\Gamma, \mathrm{U}_P \Delta) \cong \mathrm{Hom}(\mathrm{F}_P \Gamma, \Delta) \ .$$

From the theorem above, the deduction rules are of two kinds: the *passive* ones are simply the arrows of $\mathcal{S}$, while the *active* ones are inverses of arrows of $\mathcal{S}$. For instance, with respect to the propagator P_{Comp}, there is a passive deduction rule which states that both elements in a consecutive pair are terms, and there is an active deduction rule which states that every consecutive pair gives rise to a composed term. Clearly, the active rules are the most important ones.

An active deduction rule $\frac{H}{C}$ can be illustrated as follows, where the continuous arrow is an arrow in $\mathcal{S}$ and in $\overline{\mathcal{S}}$, while the dashed arrow is an arrow in $\overline{\mathcal{S}}$ only:

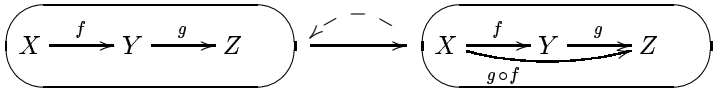


The *Yoneda contravariant morphism* of a projective sketch is defined in [10]. With respect to $\mathcal{S}$, the Yoneda morphism $\mathrm{Y}_{\mathcal{S}}$ maps each point in $\mathcal{S}$ to a P-specification and each arrow in $\mathcal{S}$ to a morphism of P-specifications, in a contravariant way. With respect to $\overline{\mathcal{S}}$, the Yoneda morphism $\mathrm{Y}_{\overline{\mathcal{S}}}$ maps each point in $\overline{\mathcal{S}}$ to a P-domain and each arrow in $\overline{\mathcal{S}}$ to a morphism of P-domains, in a contravariant way. An active rule, as above, gives rise to a morphism of P-specifications $\mathrm{Y}_{\mathcal{S}}(s)$, which becomes an isomorphism between the generated P-domains, with inverse $\mathrm{Y}_{\overline{\mathcal{S}}}(r)$ (as in *D-algebras* [2]):



Hence, $\mathrm{Y}_{\mathcal{S}}(s)$ is an entailment.

Example 10. The rule for composition in categories is illustrated as follows:



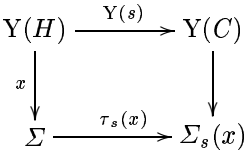
The next result derives easily from the definitions, it states that every diagrammatic logic is *sound.*

Theorem 4. *If σ is a P-entailment, and Δ is a P-domain, then* $\mathrm{Mod}(\sigma, \Delta)$ *is a bijection.*

Clearly, entailments can be composed. The elementary entailments are the deduction steps, which are obtained from the deduction rules, as explained now.

Definition 8. *Let P be a propagator, and $r = s^{-1} : H \to C$ an active P-deduction rule. Let Σ be a P-specification and let $x \in \Sigma(H)$. The* P-deduction step *associated to the rule r applied to x is the morphism $\tau_s(x)$ in the* pushout

of $\mathrm{Y}(s)$ *and* x*:*

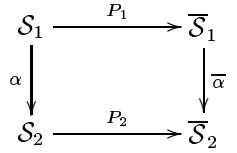


It can be proved that $\mathrm{F}_P(\tau_s(x))$ is an isomorphism, as required:

Theorem 5. *Each deduction step is an entailment.*

In the introduction, we claimed that we were able to build some "category of logics". The objects of this category are the diagrammatic logics, i.e., the propagators. The morphisms are now quite easy to define.

Definition 9. *Let* $P_1 : \mathcal{S}_1 \to \overline{\mathcal{S}}_1$ *and* $P_2 : \mathcal{S}_2 \to \overline{\mathcal{S}}_2$ *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 the following diagram is commutative:*



The next result derives immediately from the definitions.

Theorem 6. *The propagators together with their morphisms form a category.*

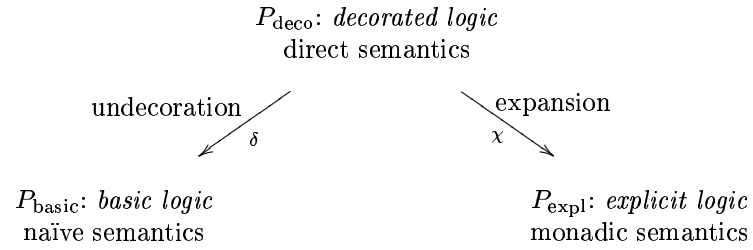
3 An application to exceptions

The opportunity of building easily non-trivial morphisms between logics is now applied to the issue of formalizing the exceptions mechanism. The details can be found in [6]. Previous work include algebraic specifications and monads [13]. However, as quoted from [14]:

> "*Evident futher 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. In the treatment of computational effects, three denotational semantics can be considered, according to [8]: the *naïve*, *direct* and *monadic* semantics. We introduce three diagrammatic logics for dealing with exceptions: the *basic*, *decorated* and

explicit logic, linked by two morphisms. Each logic corresponds to a denotational semantics, according to the following scheme:



The *decorated logic* is the most important one among the three logics. Its syntax is the syntax of a language with exceptions (here in an ML style): so, the keywords `raise` and `handle` are defined in this logic. Its deduction rules correspond to the computations with exceptions, and its models provide the required meaning of the exceptions mechanism. But it is an unusual kind of logic. On the contrary, the basic logic and the explicit logic are quite usual ones, without exceptions.

The *basic logic* provides a simplified view of the syntax, but its models are wrong.

The *explicit logic* has the right models, but it makes the exceptions totally explicit, so that the interesting part of the exceptions mechanism is lost: the type of exceptions has to appear explicitly, so that the composition cannot any more be easily written.

Example 11. The running example is over the naturals, which are defined from z (for 0) and s (for *successor*), with the sum $\text{Nat} = 1 + \text{Nat}$ and the coprojections $z : 1 \to \text{Nat}$ and $s : \text{Nat} \to \text{Nat}$. The type 1 is the "product of no type", i.e., a terminal object; it is often called `Unit` or `Void`. In the category of sets, 1 is interpreted as a singleton.

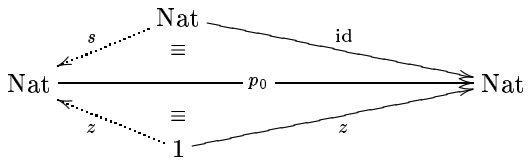
The aim is to define a *predecessor* p, such that the computation of $p(0)$ first raises an exception e, and then handles this exception in order to return 0:

```
Exception e
p(x) = case x of [ s(y)=>y | z=>raise(e) ] handle [ e=>z ]
```

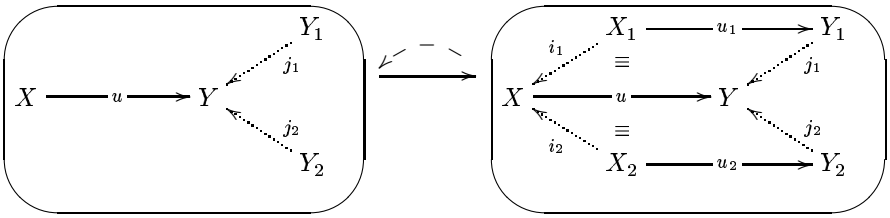
The *basic logic* has *sum* types, but no exceptions.

Example 12. In the basic logic, the predecessor p cannot be defined as above, since there are no exceptions. But the following p_0, which maps directly 0 to 0, can be defined.

```
p0(x) = case x of [ s(y)=>y | z=>z ]
```



Let us look closer at the `case` construction in the basic logic, since it plays a major rôle in the exceptions mechanism. It is based on the *extensivity* property of sums [1]. Here we study the binary case, this would easily be generalized to the n-ary case for any n. The extensivity property states that, from each sum (j_1, j_2) and each term u such that the type of u is the vertex of the sum, there is an *inverse image* of (j_1, j_2) by u, in the sense of the diagram below, and that this sum is unique up to equivalence:



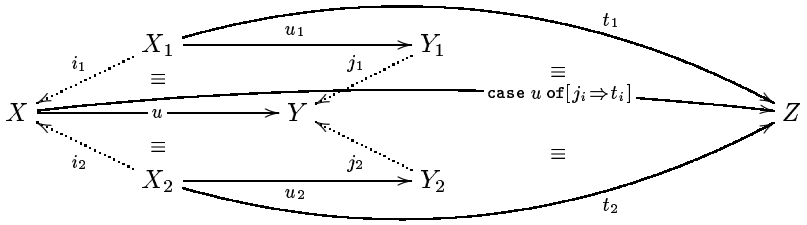
In the basic logic, the `case` construction is defined as follows:

```
t(x) = case u(x) of [ j1(y)=>t1(x,y) | j2(y)=>t2(x,y) ]
```

means that:

```
t(x) = [ i1(y)=>t1(x,y) | i2(y)=>t2(x,y) ]
```

where the sum (i_1, i_2) is the inverse image of the sum (j_1, j_2) by u:



In the *decorated logic*, the terms are decorated, which means that they are classified, either as *values* or as *computations*. This idea is due to Moggi [13], but we go somewhat further, by decorating also the terms in the deduction rules. In this way, the main features of the exception mechanism appear as several way of decorating the rules for extensivity. The decorations are represented as superscripts, $-^v$ for values and $-^c$ for computations. They may be omitted when there is no ambiguity about them. They do not appear in usual programming languages, since they can easily be recovered as follows: every term which involves some exception is a computation, the other terms are values.

Let us consider an exception declaration;

```
Exception e
```

For simplicity, here, the exception is a constant, which means that it does not depend on any parameter. We consider that such a declaration builds a computation:

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

The context of e is 1, since here the exception e is a constant. The type of e is 0, which denotes the "sum of no type", i.e., an initial object; in the category of sets, 0 is interpreted as the empty set. The reason for choosing 0 as the type of exceptions can be explained as follows. In a language with exceptions, a term $f : X \to Y$, when applied to an argument of type X, either returns a value of type Y or raises an exception. So, when $Y = 0$, such a term must raise an exception; and clearly, an exception does satisfy this property.

If the decoration of e is forgotten, we are left with a term $e : 1 \to 0$, which should be interpreted, in any set-valued model, as a map from a singleton to the empty set: such a map does not exist.

Now, we present on our example the way the keywords `raise` and `handle` are introduced.

Example 13. The starting point is a specification of naturals, similar to the basic one, where the terms z and s are values z^v and s^v. As explained above, the declaration:

```
Exception e
```

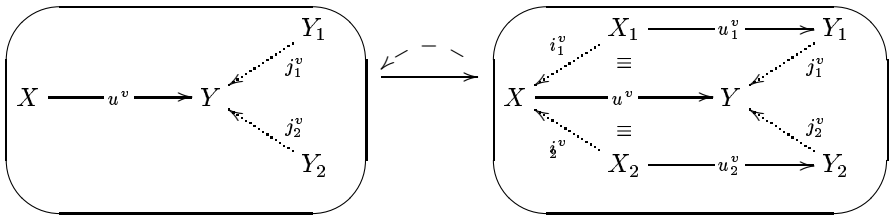
adds to the specification a constant computation:

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

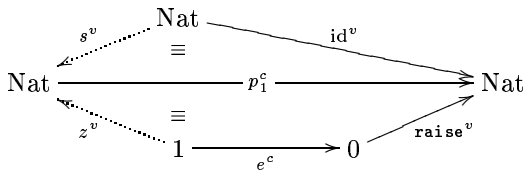
Let us define the computation p_1 as follows:

```
p1(x) = case x of [ s(y)=>y | z=>raise(e) ]
```

This `case` construction has the form "`case` $u(x)$ `of` ..." where $u(x) = x$, so that u is a value. Because u is a value, the construction is quite similar to the `case` construction in the basic logic, it uses the following decorated version of the extensivity property:



This `case` construction in the decorated logic can be illustrated as:



where `raise` (or $\mathtt{raise}_{\mathrm{Nat}}$) is the unique value from 0 to Nat:

$$\mathtt{raise}^v = [\,]^v_{\mathrm{Nat}} : 0 \to \mathrm{Nat}\ .$$

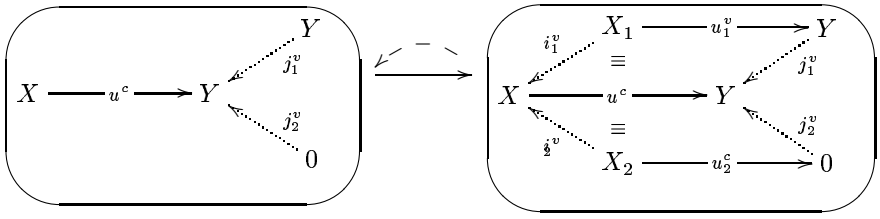
Now, the computation p can be defined as follows:

```
p(x) = p1(x) handle [ e=>z ]
```

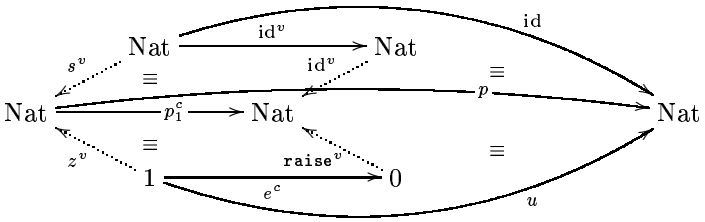
We consider that the `handle` construction is made of two different `case` constructions. Roughly speaking, the first `case` (which is written $\mathtt{case}^c$) tests whether $p_1(x)$ raises an exception, and when this happens, the second `case` (which is written $\mathtt{case}^e$) tests which one is this exception. With the notations from [6], this is written as:

$$p(x) = \mathtt{case}^c\ p_1(x)\ \mathtt{of}\ [\,x \Rightarrow x \mid \mathtt{raise} \Rightarrow w\,]\ \text{ where }\ w = \mathtt{case}^e\ e\ \mathtt{of}\ [\,e \Rightarrow z\,]$$

and where the $\mathtt{case}^c$ and $\mathtt{case}^e$ constructions correspond to variants of the basic `case` construction, which differ in the way the rules are decorated. The $\mathtt{case}^c$ construction has the form "$\mathtt{case}^c\ u(x)\ \mathtt{of}\ \ldots$" where u is a computation, not a value (here, $u = p_1$). This construction uses the following decorated version of the extensivity property, which means (because u_1 is a value and u_2 is a computation) that X_1 stands for the set of x's such that $u(x)$ does not raise an exception, while X_2 stands for the set of x's such that $u(x)$ does raise an exception:



The $\mathtt{case}^c$ construction in our example can be illustrated as:



Then, in the decorated logic, it is easily proved that:

$$p^c \equiv [\, s \Rightarrow \mathrm{id} \mid z \Rightarrow u \,] \text{ and } u^c \equiv z \ ,$$

which finally proves that, as expected:

$$p \equiv p_0 \ .$$

More generally, in the decorated logic, as in any diagrammatic logic, proofs can be made and models can be defined, although there is no canonical interesting domain of sets.

The morphism $\delta : P_{\mathrm{deco}} \to P_{\mathrm{basic}}$ is simply the *undecoration*: the decorations are forgotten, so that all the meaningful interpretations of the computations are lost.

The *explicit logic* is similar to the basic logic, but with a distinguished type E for exceptions: "exceptions are explicit".

The *expansion* morphism $\chi : P_{\mathrm{deco}} \to P_{\mathrm{expl}}$ makes the exceptions explicit, in the way that can be guessed:

- A value $t^v : X \to Y$ becomes a term $t : X \to Y$.
- A computation $t^c : X \to Y$ becomes a term $t : X \to Y + E$.
- The composition of these terms propagates the exceptions; this means that it is done in the Kleisli way, as in the monads approach.

For example, an exception $e^c : 1 \to 0$ becomes $e : 1 \to E$. So, our notion of morphisms between diagrammatic logics allows to distinguish the decorated and the explicit points of view, and to give a precise status to the relationship between both.

Let Σ_{deco} be a decorated specification, and let Σ_{expl} be the explicit specification which is freely generated by Σ_{deco}, with respect to the expansion morphism χ. The meaning of Σ_{deco} can be given either by a set-valued model of Σ_{expl}, or by a model of Σ_{deco}. Both coincide, thanks to the adjunction which is associated to χ.

4 Conclusion

This paper corresponds to a talk given at the MAP conference in Dagstuhl, in january, 2005. Its subject is related to the three keywords of the MAP project:

- **M***athematics*: we use mathematical tools like categories, adjunction, sketches,...
- **A***lgorithms*: we can formalize algorithms, even when they are written in a non-functional way.
- **P***roofs*: diagrammatic logic could become a framework for proofs of programs using computational effects.

References

1. A. Carboni, S. Lack, R.F.C. Walters. Introduction to extensive and distributive categories, *J. Pure Appl. Algebra* **84**145-158 (1993) .
2. L. Coppey. Théories algébriques et extensions de pré-faisceaux, *Cahiers de Topologie et Géométrie Différentielle* **13** (1972).
3. C. Domínguez, D. Duval, L. Lambán, J. Rubio. Towards Diagrammatic Specifications of Symbolic Computation Systems. This conference.
4. D. Duval. Diagrammatic specifications. *Mathematical Structures in Computer Science* **13** 857-890 (2003). This is the journal version of [5].
5. D. Duval, C. Lair. Diagrammatic specifications. *Rapport de recherche IMAG-LMC* **1043** (2002). This is a preliminary version of [4].
6. D. Duval, J.-C. Reynaud. Diagrammatic logic and effects : the example of exceptions. Submitted for publication.
7. C. Ehresmann. Introduction to the theory of structured categories. Report **10**, University of Kansas, Lawrence (1966).
8. C. Führmann. The structure of call-by-value. PhD thesis, Division of Informatics, University of Edinburgh (2000).
9. M. Hébert, J. Adámek, J. Rosický. More on orthogonality in locally presentable categories. *Cahiers de Topologie et Géométrie Différentielle Catégoriques* **XLII-1**, 51–80 (2001).
10. C. Lair, D. Duval. Esquisses et spécifications. Manuel de référence, 4ème partie : Fibrations et Eclatements, Lemmes de Yoneda et Modèles Engendrés. *Rapport de recherche du LACO, Université de Limoges* **2001-03**.
11. J. Lambek, P.J. Scott. *Introduction to higher order categorical logic.* Cambridge University Press (1986).
12. S. Mac Lane. *Categories for the working mathematician*, Springer-Verlag (1971).
13. E. Moggi. Notions of computation and monads, *Information and Computation* **93**, 55–92 (1991).
14. G. Plotkin, J. Power. Semantics for algebraic operations. *Electronic Notes in Theoretical Computer Science* **45**, 1–14 (2001).
15. G. Plotkin and J. Power. Algebraic Operations and Generic Effects. *Applied Categorical Structures* **11**, 69–94 (2003). This is the journal version of [14].