HAL
open science

Breaking a monad-comonad symmetry between computational effects

Jean-Guillaume Dumas, Dominique Duval, Jean-Claude Reynaud

To cite this version:

Jean-Guillaume Dumas, Dominique Duval, Jean-Claude Reynaud. Breaking a monad-comonad symmetry between computational effects. [Research Report] arXiV. 2014. hal-00941653

HAL Id: hal-00941653

https://hal.science/hal-00941653v1

Submitted on 4 Feb 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Breaking a monad-comonad symmetry between computational effects

Jean-Guillaume Dumas
Laboratoire Jean Kuntzmann, University of Grenoble, France

Dominique Duval
Laboratoire Jean Kuntzmann, University of Grenoble, France

Jean-Claude Reynaud
Reynaud Consulting, Claix, France

February 4., 2014

Abstract

Computational effects may often be interpreted in the Kleisli category of a monad or in the coKleisli category of a comonad. The duality between monads and comonads corresponds, in general, to a symmetry between construction and observation, for instance between raising an exception and looking up a state. Thanks to the properties of adjunction one may go one step further: the coKleisli-on-Kleisli category of a monad provides a kind of observation with respect to a given construction, while dually the Kleisli-on-coKleisli category of a comonad provides a kind of construction with respect to a given observation. In the previous examples this gives rise to catching an exception and updating a state. However, the interpretation of computational effects is usually based on a category which is not self-dual, like the category of sets. This leads to a breaking of the monad-comonad duality. For instance, in a distributive category the state effect has much better properties than the exception effect. This remark provides a novel point of view on the usual mechanism for handling exceptions. The aim of this paper is to build an equational semantics for handling exceptions based on the coKleisli-on-Kleisli category of the monad of exceptions. We focus on n-ary functions and conditionals. We propose a programmer's language for exceptions and we prove that it has the required behaviour with respect to n-ary functions and conditionals.

Keywords. Computational effects; monads and comonads; duality; decorated logics.

1 Introduction

Categorical semantics for programming languages interprets types as objects and terms as morphisms; in this setting, substitution is composition, categorical

products are used for dealing with n-ary operations and coproducts for conditionals. The most famous result in this direction is the Curry-Howard-Lambek correspondence which relates intuitionistic logic, simply typed lambda calculus and cartesian closed categories. The *algebraic effects* challenge is the search for some extension of this correspondence to a categorical framework corresponding to *computational effects*, which means, roughly, to non-functional features of programming languages. Moggi proposed to use the categorical notion of *monad* for this purpose [18], then monads were popularized by Wadler [28] and implemented in Haskell and F♯. Related categorical notions like Freyd categories, arrows, Lawvere theories, were also proposed [25, 10, 22, 12]. Moreover, the dual notion of *comonad* can also be used for dealing with computational effects [27, 4, 20]. This gives rise to a three-tier classification of terms which is similar to the one in [29]. Some effects, like the state effect, can be seen both as monads and as comonads [18, 4]. Other effects, like the handling of exceptions, do not fit easily in the monad approach [23, 24, 1]. However the use of the co-Eilenberg-Moore-on-Eilenberg-Moore category of the monad of exceptions was successfully used by Levy for adapting the monad approach to the handling of exceptions [15]; in this paper we follow a similar line.

The aim of this paper is to build an equational semantics for handling exceptions based on the coKleisli-on-Kleisli category of the monad of exceptions. We focus on n-ary functions and conditionals because they correspond to the dual categorical notions of products and coproducts, although their behaviour with respect to effects is quite different: in general there is no ambiguity in using conditionals with effects, whereas the value of an expression involving n-ary functions may depend on the order of evaluation of the arguments. The equational semantics we use are *decorated*: the terms and equations are annotated, in a way similar to the type-and-effect systems [16], in order to classify them according to their interaction with the effect. Typically, for exceptions, terms are classified as pure, propagators (which must propagate exceptions) and catchers (which may recover from exceptions); thus, there is no need for an explicit "type of exceptions", and we get a clear dictinction between a coproduct type $A+B$ in the syntax and a coproduct $A+E$ (where E is the "object of exceptions") which may be used for interpreting terms involving the type A. In the equational semantics we use coproduct types $A+B$ but we never use coproducts involving E. In order to get an equational semantics for exceptions, we start from two facts: first, the Kleisli-on-coKleisli category of a comonad can be used for building an equational semantics for states [4]; secondly, there is a duality between the denotational semantics of the state effect and the denotational semantics of the core operations for the exception effect [5]. We adapt this duality to the equational level, then we build the programmer's language for exceptions by adding some control to the core operations. Finally we propose a programmer's language for exceptions, built from categorical products and coproducts, and we prove that it satisfies equations providing the required behaviour (as explained above) with respect to n-ary functions and conditionals. The equational semantics for states is implemented in Coq [8] and the one for exceptions is in progress.

The duality between monads and comonads corresponds, in general, to a

symmetry between construction and observation: raising an exception is a construction, reading the value of a location is an observation. As recalled in Section 2, thanks to the properties of adjunction one may go one step further: the coKleisli-on-Kleisli category of a monad provides a kind of observation with respect to a given construction, while dually the Kleisli-on-coKleisli category of a comonad provides a kind of construction with respect to a given observation. In the previous examples this gives rise to catching an exception and updating a state, respectively. The coKleisli-on-Kleisli category of a monad, as well as the Kleisli-on-coKleisli category of a comonad, provide a classification of terms and equations. In Section 3 we define variants of the equational logic for dealing with this classification. These variants are called *decorated logics*: there is a decorated logic $\mathcal{L}_{mon}$ for a monad and dually a decorated logic $\mathcal{L}_{comon}$ for a comonad. When the monad is the exception monad, we can add to the decorated logic $\mathcal{L}_{mon}$ the *core* operations for exceptions: the *tagging* operations for encapsulating an ordinary value into an exception, and the *untagging* operations for recovering the ordinary value which has been previously encapsulated in an exception. Dually, When the comonad is the state comonad, we can add the basic operations for states: the *lookup* operations which observe the state and the *update* operations which modify it. In Section 4 we assume that the category $\mathbf{C}$ has some distributivity or extensivity property, like for instance the category of sets. This breaks the monad-comonad duality: the state effect gets better properties with respect to coproducts, while the exception effect does not get better properties with respect to products. On the comonad side, we check that the side-effects due to the evolution of state do not perturb the case-distinction features, and we provide decorated equations for imposing an order on the interpretation of the arguments of multivariate functions. On the monad side, we check that the properties of operations for catching exceptions are quite poor. This is circumvented by encapsulating the catching operations in *try-catch* blocks. This provides a novel point of view on the formalization of the usual mechanism for handling exceptions. We get a programmer's language for exceptions which has the required behaviour with respect to n-ary functions and conditionals.

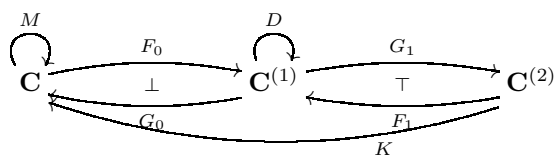
2 Preliminaries

We present some well-known results about monads and comonads in Section 2.1) and (independently) about equational logic with conditionals in Section 2.2.

2.1 CoKleisli-on-Kleisli category

This Section relies on [17]. A similar construction is used in [15, 14], with "Kleisli" replaced by "Eilenberg-Moore". Let $\mathbf{C}$ be a category and (M, η, μ) a monad on $\mathbf{C}$. Let $\mathbf{C}^{(1)}$ be the Kleisli category of this monad and $F_0 \dashv G_0 \colon \mathbf{C}^{(1)} \to \mathbf{C}$ the corresponding adjunction. Then $M = G_0 \circ F_0 \colon \mathbf{C} \to \mathbf{C}$. Let $D = F_0 \circ G_0 \colon \mathbf{C}^{(1)} \to \mathbf{C}^{(1)}$, it is the endofunctor of a comonad (D, ε, δ) on $\mathbf{C}^{(1)}$.

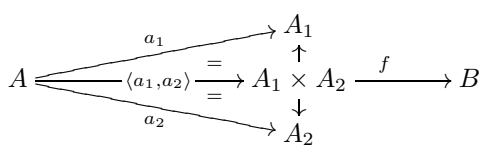
Let $\mathbf{C}^{(2)}$ be the coKleisli category of this comonad and $F_1 \dashv G_1 \colon \mathbf{C}^{(1)} \to \mathbf{C}^{(2)}$ the corresponding adjunction. Then $D = F_1 \circ G_1 \colon \mathbf{C}^{(1)} \to \mathbf{C}^{(1)}$. In such a situation, there is a unique functor $K \colon \mathbf{C}^{(2)} \to \mathbf{C}$ such that $K \circ G_1 = G_0$ and $F_0 \circ K = F_1$.



The three categories $\mathbf{C}$, $\mathbf{C}^{(1)}$ and $\mathbf{C}^{(2)}$ have the same objects. There is a morphism $g^{(1)} \colon A \to B$ in $\mathbf{C}^{(1)}$ for each morphism $g_1 \colon A \to MB$ in $\mathbf{C}$, and there is a morphism $h^{(2)} \colon A \to B$ in $\mathbf{C}^{(2)}$ for each morphism $h_2 \colon MA \to MB$ in $\mathbf{C}$. The functor K maps A to MA and $h^{(2)} \colon A \to B$ to $h_2 \colon MA \to MB$. We are mainly interested in the functors F_0 and G_1. They are the identity on objects, F_0 maps $f_0 \colon A \to B$ in $\mathbf{C}$ to $f^{(1)} \colon A \to B$ in $\mathbf{C}^{(1)}$ corresponding to $f_1 = \eta_B \circ f_0 \colon A \to MB$ in $\mathbf{C}$, and G_1 maps $g^{(1)} \colon A \to B$ in $\mathbf{C}^{(1)}$ corresponding to $g_1 \colon A \to MB$ in $\mathbf{C}$ to $g^{(2)} \colon A \to B$ in $\mathbf{C}^{(2)}$ corresponding to $g_2 = \mu_B \circ Mg_1 \colon MA \to MB$ in $\mathbf{C}$. Thus, $G_1 \circ F_0$ maps $f_0 \colon A \to B$ in $\mathbf{C}$ to $f^{(2)} \colon A \to B$ in $\mathbf{C}^{(2)}$ corresponding to $f_2 = Mf_0 \colon MA \to MB$ in $\mathbf{C}$.

2.2 Equational logic with conditionals

We choose a categorical presentation of logic as for instance in [21], in a *bicartesian* category (i.e., a category with finite products and coproducts). In a functional programming language, from this point of view, types are objects, terms are morphisms and substitution is composition. Each term f has a source type A and a target type B, this is denoted $f \colon A \to B$. A term has precisely one source type, which can be a product type or the unit type $\mathbb{1}$. A n-ary operation $f : A_1, \ldots, A_n \to B$ corresponds to a morphism $f : A_1 \times \cdots \times A_n \to B$ (this holds for every $n \geq 0$, with $f : \mathbb{1} \to B$ when $n = 0$). Typically, when $n = 2$, the substitution of terms $a_1 \colon A \to A_1, a_2 \colon A \to A_2$ for the variables x_1, x_2 in $f(x_1, x_2)$ is the composition of the pair $\langle a_1, a_2 \rangle \colon A \to A_1 \times A_2$ with $f : A_1 \times A_2 \to B$.



Conditionals corresponds to copairs: a command like *if* b *then* f *else* g corresponds to the morphism $[f|g] \circ b$, where $[f|g]$ is the *copair* of $f \colon \mathbb{1} \to B$ and $g \colon \mathbb{1} \to B$, i.e., the unique morphism $h \colon \mathbb{1} + \mathbb{1} \to B$ such that $h \circ \mathit{true} = f$ and

$h \circ \mathit{false} = g$.

$$\begin{tikzcd}
 & \mathbb{1} \arrow[dd, "\mathit{true}"'] \arrow[ddrr, "f"] & & \\
A \arrow[r, "b"] & \mathbb{1} + \mathbb{1} \arrow[rr, "{[f|g]}", "="'] & & B \\
 & \mathbb{1} \arrow[uu, "\mathit{false}"] \arrow[urr, "g"'] & &
\end{tikzcd}$$

The grammar and the rules of the equational logic with conditionals are recalled in Fig. 1. For short, rules with the same premisses may be grouped together: $\frac{H_1 \ldots H_n}{C_1}, \ldots, \frac{H_1 \ldots H_n}{C_p}$ may be written $\frac{H_1 \ldots H_n}{C_1 \ldots C_p}$.

3 The duality

In Sections 3.1 and 3.2 we define decorated logics $\mathcal{L}_{mon}$ and $\mathcal{L}_{comon}$, together with their interpretation in a category with a monad and with a comonad, respectively. Then in Sections 3.3 and 3.4 we extend $\mathcal{L}_{mon}$ and $\mathcal{L}_{comon}$ into $\mathcal{L}_{exc}$ and $\mathcal{L}_{st}$ which are dedicated to the monad of exception and to the comonad of states, respectively. The interpretations of these logics provide the duality between the denotational semantics of states and exceptions mentioned in [5]. All these logics are called *decorated logics* because their grammar and inference rules are essentially the grammar and inference rules for the logic $\mathcal{L}_{eq}$ (from Section 2.2) together with *decorations* for the terms and for the equations. The decorations for the terms are similar to the *annotations* of the types and effects systems [16]. Decorated logics are introduced in [3] in an abstract categorical framework which will not be explicitly used in this paper.

3.1 A decorated logic for a monad

In the logic $\mathcal{L}_{mon}$ for monads, each term has a decoration which is denoted as a superscript (0), (1) or (2): a term is *pure* when its decoration is (0), it is a *constructor* when its decoration is (1) and a *modifier* when its decoration is (2). Each equation has a decoration which is denoted by replacing the symbol $\equiv$ either by $\cong$ or by $\sim$: an equation with $\cong$ is called *strong*, with $\sim$ it is called *weak*. In order to give a meaning to the logic $\mathcal{L}_{mon}$, let us consider a bicartesian category $\mathbf{C}$ with a monad (M, η, μ). The categories $\mathbf{C}^{(0)} = \mathbf{C}$, $\mathbf{C}^{(1)}$, $\mathbf{C}^{(2)}$ and the functors $F_0 : \mathbf{C}^{(0)} \to \mathbf{C}^{(1)}$ and $G_1 : \mathbf{C}^{(1)} \to \mathbf{C}^{(2)}$ are defined as in Section 2.1. Then we get an interpretation $\mathbf{C}_M$ of the grammar and the conversion rules of $\mathcal{L}_{mon}$ as follows.

- A type A is interpreted as an object A of $\mathbf{C}$.
- A term $f^{(d)} \colon A \to B$ is interpreted as a morphism $f \colon A \to B$ in $\mathbf{C}^{(2)}$; if $d = 0$ then f must be in the image of $\mathbf{C}^{(0)}$ by $G_1 \circ F_0$, and if $d = 1$ then f must be in the image of $\mathbf{C}^{(1)}$ by G_1. This means that all terms are interpreted as morphisms of $\mathbf{C}$: a pure term $f^{(0)} \colon A \to B$ as a morphism $f_0 \colon A \to B$ in $\mathbf{C}$; a constructor $g^{(1)} \colon A \to B$ as a morphism $g_1 \colon A \to MB$ in $\mathbf{C}$; and a modifier $h^{(2)} \colon A \to B$ as a morphism $h_2 \colon MA \to MB$ in $\mathbf{C}$.

- A strong equation $f^{(d)} \cong g^{(d)} \colon A \to B$ is interpreted as an equality $f = g \colon A \to B$ in $\mathbf{C}^{(2)}$, i.e., as an equality $f_2 = g_2 \colon MA \to MB$ in $\mathbf{C}$.
- A weak equation $f^{(d)} \sim g^{(d)} \colon A \to B$ is interpreted as an equality $f_2 \circ \eta_A = g_2 \circ \eta_A \colon A \to MB$ in $\mathbf{C}$.

Example 3.1. Let us consider the monad of lists (or words), and its interpretation in the category of sets. Then a term $f : A \to B$ is interpreted as a code, i.e., as a map $f : A^* \to B^*$ from the words on A to the words on B. The classification of the terms provided by the decorations corresponds to a well-known classification of the codes: if f is constructor then for each word $u = x_1 \dots x_n$ on A the word $f(u) = f(x_1) \dots f(x_n)$ is the concatenation of the images of the letters in u, and if f is pure then in addition for each letter x in A the word $f(x)$ is a letter in B.

The inference rules of $\mathcal{L}_{mon}$ are decorated versions of the rules of the equational logic with conditionals. The main rules are given in Fig. 2, and all rules in Appendix A. When a decoration is clear from the context, it is often omitted.

- The conversion rules are decorated versions of rules of the form $\frac{H}{H}$.
- All rules of $\mathcal{L}_{eq}$ are decorated with (0) for terms and $\cong$ for equations: the pure terms with the strong equations form a sublogic of $\mathcal{L}_{mon}$, which is the same as $\mathcal{L}_{eq}$. Thus, the structural operations like id, pr, $\langle\ \rangle$, in, $[\ ]$, are pure.
- The congruence rules for equations are decorated with all decorations for terms and for equations, with one notable exception: the substitution rule holds only when the substituted term is pure.
- The categorical rules hold for all decorations and the decoration of a composed term is the maximum of the decorations of its components.
- The product rules are decorated only as pure.
- For the coproduct rules, the terms in rules (copair) and (copair-u) can be decorated as pure or constructors, and the decoration of a copair is the maximum of the decorations of its components. Thus, conditionals can be built from constructors, but not from modifiers. The decorated rule (initial-u) states that $[\ ]_B$ is the unique term from $\mathbb{0}$ to B, up to weak equality.

It is easy to check that these rules are satisfied by the interpretation $\mathbf{C}_M$ of $\mathcal{L}_{mon}$. Each $f^{(0)}$ may be converted to $f^{(1)} = F_0 f^{(0)}$ and to $f^{(2)} = G_1 F_0 f^{(0)}$, and each $g^{(1)}$ to $g^{(2)} = G_1 g^{(1)}$. Each strong equality $f = g$ gives rise to an equality $f_2 \circ \eta_A = g_2 \circ \eta_A$, and both equalities are equivalent when f and g are in $\mathbf{C}^{(1)}$. Products and coproducts in $\mathcal{L}_{mon}$ are interpreted as products and coproducts in $\mathbf{C}$. For instance, the pair of two constructors $f^{(1)} \colon A \to B_1$ and $g^{(1)} \colon A \to B_2$ is interpreted as the pair $\langle f_1, g_1 \rangle \colon MA \to B_1 \times B_2$ in $\mathbf{C}$.

3.2 A decorated logic for a comonad

The dual of the decorated logic $\mathcal{L}_{mon}$ for a monad is the decorated logic $\mathcal{L}_{comon}$ for a comonad. Thus, the grammar of $\mathcal{L}_{comon}$ is the same as the grammar of $\mathcal{L}_{mon}$, but a term with decoration (1) is now called an *accessor* (or an *observer*). The conversion rules are the same as those in $\mathcal{L}_{mon}$. Let $\mathbf{C}$ be a bicartesian category with a comonad (D, ε, δ). The categories $\mathbf{C}^{(0)} = \mathbf{C}$, $\mathbf{C}^{(1)}$, $\mathbf{C}^{(2)}$ and the functors $F_0 : \mathbf{C}^{(0)} \to \mathbf{C}^{(1)}$ and $G_1 : \mathbf{C}^{(1)} \to \mathbf{C}^{(2)}$ are defined dually to Section 2.1. Then we get an interpretation $\mathbf{C}_D$ of the grammar of $\mathcal{L}_{comon}$ as follows.

- A type A is interpreted as an object A of $\mathbf{C}$.
- A term $f^{(d)} \colon A \to B$ is interpreted as a morphism $f \colon A \to B$ in $\mathbf{C}^{(2)}$, which can be expressed as a morphism in $\mathbf{C}$: a pure term $f^{(0)} \colon A \to B$ as a morphism $f_0 \colon A \to B$ in $\mathbf{C}$; an accessor $g^{(1)} \colon A \to B$ as a morphism $g_1 \colon DA \to B$ in $\mathbf{C}$; and a modifier $h^{(2)} \colon A \to B$ as a morphism $h_2 \colon DA \to DB$ in $\mathbf{C}$.
- A strong equation $f^{(d)} \cong g^{(d)} \colon A \to B$ is interpreted as an equality $f_2 = g_2 \colon DA \to DB$ in $\mathbf{C}$.
- A weak equation $f^{(d)} \sim g^{(d)} \colon A \to B$ is interpreted as an equality $\varepsilon_B \circ f_2 = \varepsilon_B \circ g_2 \colon A \to DB$ in $\mathbf{C}$.

The rules for $\mathcal{L}_{comon}$ are nearly the same as the corresponding rules for $\mathcal{L}_{mon}$, except that for weak equations the substitution rule always holds while the replacement rule holds only when the replaced term is pure, and in the rules for products and coproducts the decorations are permuted, see Fig. 3 for the main rules. The logic $\mathcal{L}_{comon}$ can be interpreted dually to $\mathcal{L}_{mon}$. Let $\mathbf{C}$ be a bicartesian category and (D, ε, δ) a comonad on $\mathbf{C}$. Then we get a model $\mathbf{C}_D$ of the decorated logic $\mathcal{L}_{comon}$, where an accessor $f^{(1)} \colon A \to B$ is interpreted as a morphism $f_1 \colon DA \to B$ in $\mathbf{C}$, a weak equation $f^{(2)} \sim g^{(2)} \colon A \to B$ as an equality $\varepsilon_B \circ f_2 = \varepsilon_B \circ g_2 \colon DA \to B$ in $\mathbf{C}$ and a copair of two accessors $f^{(1)} \colon A_1 \to B$ and $g^{(1)} \colon A_2 \to B$ as the copair $[f_1|g_1] \colon A_1 + A_2 \to DB$ in $\mathbf{C}$.

3.3 A decorated logic for the monad of exceptions

Let us assume that there is in $\mathbf{C}$ a distinguished object E called the *object of exceptions*. The *monad of exceptions* on $\mathbf{C}$ is the monad (M, η, μ) with endofunctor $MX = X + E$, its unit η is made of the coprojections $\eta_X \colon X \to X + E$ and its multiplication μ is defined by $\mu_X = [id_{X+E}|in_X] \colon (X+E)+E \to X+E$ where $in_X \colon E \to X+E$ is the coprojection. As in Section 3.1, the category $\mathbf{C}$ with the monad of exceptions provides a model $\mathbf{C}_M$ of the decorated logic $\mathcal{L}_{mon}$. The name of the decorations can be adapted to the monad of exceptions: a constructor is called a *propagator*: it may raise an exception but cannot recover from an exception, so that it has to propagate all exceptions; a modifier is called a *catcher*.

For this specific monad, it is possible to extend the logic $\mathcal{L}_{mon}$ as $\mathcal{L}_{exc}$, called the *decorated logic for exceptions*, so that $\mathbf{C}_M$ can be extended as a model $\mathbf{C}_{exc}$ of $\mathcal{L}_{exc}$. First, we get copairs of a propagator and a modifier, as in the first part of Fig. 4 for the left copairs (the rules for the right copairs are symmetric). The interpretation of the left copair $[f|g]_l^{(2)} : A_1 + A_2 \to B$ of $f^{(1)} : A_1 \to B$ and $g^{(2)} : A_2 \to B$ is the copair $[f_1|g_2] : A_1+A_2+E \to B+E$ of $f_1 : A_1 \to B+E$ and $g_2 : A_2+E \to B+E$ in $\mathbf{C}$. This is possible because $(A_1+A_2)+E$ is canonically isomorphic to $A_1 + (A_2 + E)$, whereas for a monad generally $M(A_1 + A_2)$ is not isomorphic to $A_1 + MA_2$. For instance, the coproduct of $A \cong A + \mathbb{0}$, with coprojections $id_A^{(0)} : A \to A$ and $[\ ]_A^{(0)} : \mathbb{0} \to A$, gives rise to the left copair $[f|g]_l^{(2)} : A \to B$ of any propagator $f^{(1)} \colon A \to B$ with any modifier $g^{(2)} \colon \mathbb{0} \to B$, which is characterized up to strong equations by $[f|g]_l \sim f$ and $[f|g]_l \cong g$. The construction of $[f|g]_l^{(2)}$ and its interpretation can be illustrated as follows:

$$\begin{tikzcd} A \arrow[d, "id^{(0)}"'] \arrow[drr, "f^{(1)}"] & & \\ A \arrow[rr, "\sim", "\cong"', "{[f|g]_l^{(2)}}"] & & B \\ \mathbb{0} \arrow[u, "{[\ ]^{(0)}}"] \arrow[urr, "g^{(2)}"'] & & \end{tikzcd} \qquad \begin{tikzcd} A \arrow[d] \arrow[drr, "f_1"] & & \\ A+E \arrow[rr, "=", "="', "{([f|g]_l)_2}"] & & B+E \\ E \arrow[u] \arrow[urr, "g_2"'] & & \end{tikzcd}$$

Moreover, the rule (effect) expresses the fact that, when $MX = X + E$, two modifiers coincide as soon as they coincide on ordinary values and on exceptions, whereas for a monad generally the morphisms $\eta_X \colon X \to MX$ and $M[\ ]_X \colon M\mathbb{0} \to MX$ do not form a coproduct. For each set *Exn* of *exception names*, additional grammar and rules for the logic $\mathcal{L}_{exc}$ are given in Fig. 4. We extend the grammar with a type V_T, a propagator $\mathtt{tag}_T^{(1)} : V_T \to \mathbb{0}$ and a catcher $\mathtt{untag}_T^{(2)} : \mathbb{0} \to V_T$ for each exception name T, and we also extend its rules. The logic $\mathcal{L}_{exc}$ obtained performs the *core* operations on exceptions: the *tagging* operations encapsulate an ordinary value into an exception, and the *untagging* operations recover the ordinary value which has been encapsulated in an exception. This may be generalized by assuming a hierarchy of exception names [7]. In Fig. 4, the rule (exc-coprod-u) is a decorated rule for coproducts. It asserts that two functions without argument coincide as soon as they coincide on each exception. Together with the rule (effect) this implies that two functions coincide as soon as they coincide on their argument and on each exception. For each family of objects $(V_T)_{T \in Exn}$ in $\mathbf{C}$ such that $E \cong \sum_{T \in Exn} V_T$ we build a model $\mathbf{C}_{exc}$ of $\mathcal{L}_{exc}$, which extends the model $\mathbf{C}_M$ of $\mathcal{L}_{mon}$ with functions for tagging and untagging the exceptions. The types V_T are interpreted as the objects V_T and the propagators $\mathtt{tag}_T^{(1)} : V_T \to \mathbb{0}$ as the coprojections from V_T to E. Then the interpretation of each catcher $\mathtt{untag}_T^{(2)} : \mathbb{0} \to V_T$ is the function $\mathtt{untag}_T : E \to V_T + E$ defined as the cotuple (or case distinction) of the functions $f_{T,R} : V_R \to V_T + E$ where $f_{T,T}$ is the coprojection of V_T in $V_T + E$ and $f_{T,R}$ is made of $\mathtt{tag}_R : V_T \to E$ followed by the coprojection of E in $V_T + E$ when $R \neq T$. This can be illustrated, in an informal way, as follows: $\mathtt{tag}_T$ encloses its argument a in a box with name T, while $\mathtt{untag}_T$ opens every box with name T to recover its argument and returns

every box with name $R \neq T$ without opening it:

3.4 A decorated logic for the comonad of states

Let us assume that there is in $\mathbf{C}$ a distinguished object S called the *object of states*. The *comonad of states* on $\mathbf{C}$ is the comonad (D, ε, δ) with endofunctor $DX = X \times S$, its counit ε is made of the projections $\varepsilon_X \colon X \times S \to X$ and its comultiplication δ is defined by $\delta_X = \langle id_{X \times S}, pr_X \rangle \colon X \times S \to (X \times S) \times S$ where $pr_X \colon X \times S \to S$ is the projection. This comonad is sometimes called the *product comonad*; it is different from the *costate comonad* or *store comonad* with endofuntor $DA = S \times A^S$ [9]. As in Section 3.2, the category $\mathbf{C}$ with the comonad of states provides a model $\mathbf{C}_D$ of the decorated logic $\mathcal{L}_{comon}$.

For this specific comonad, it is possible to extend the logic $\mathcal{L}_{comon}$ as $\mathcal{L}_{st}$, called the *decorated logic for states*, so that $\mathbf{C}_D$ can be extended as a model $\mathbf{C}_{st}$ of $\mathcal{L}_{st}$. In Fig. 5, the rule (st-prod-u) is a decorated rule for coproducts. It asserts that two functions without result coincide as soon as they coincide when observed at each location. Together with the rule (st-effect) this implies that two functions coincide as soon as they return the same value and coincide on each location. For each family of objects $(V_T)_{T \in Loc}$ in $\mathbf{C}$ such that $S \cong \prod_{T \in Loc} V_T$ we build a model $\mathbf{C}_{st}$ of $\mathcal{L}_{st}$, which extends the model $\mathbf{C}_D$ of $\mathcal{L}_{comon}$ with functions for looking up and updating the locations. The types V_T are interpreted as the objects V_T and the accessors $\mathtt{lookup}_T^{(1)} : \mathbb{1} \to V_T$ as the projections from S to V_T. Then the interpretation of each modifier $\mathtt{update}_T^{(2)} : V_T \to \mathbb{1}$ is the function $\mathtt{update}_T : V_T \times S \to S$ defined as the tuple of the functions $f_{T,R} : V_T \times S \to V_R$ where $f_{T,T}$ is the projection of $V_T \times S$ to V_T and $f_{T,R}$ is made of the projection of $V_T \times S$ to S followed by $\mathtt{lookup}_R : S \to V_R$ when $R \neq T$.

4 Breaking the duality

In Section 4.1 we discuss the behaviour of conditionals and n-ary operations with respect to effects. In Section 4.2 the decorated logic for states is extended under the assumption that $\mathbf{C}$ is distributive, and we easily get Theorem 4.3 about conditionals and sequential pairs. In Section 4.3 the decorated logic for exceptions is extended, in a way which is *not* dual to the extension for states. It happens that catchers do not have "good" properties with respect to conditionals and sequential pairs. Thus, we define a new language, called the *programmer's language* for exceptions, in order to encapsulate the catchers in *try-catch* blocks. This corresponds to the usual way to deal with exceptions in a computer language. Then, under the assumption that $\mathbf{C}$ satisfies a limited form of extensivity, we get Theorem 4.9 about conditionals and sequential pairs for the programmer's language for exceptions. Note that distributivity and

extensivity are related notions [2], and that each of them breaks the duality between products and coproducts.

4.1 Effects: conditionals and sequential pairs

When there are effects, for a binary operation $f : A_1 \times A_2 \to B$, the fact that the substitution of terms a_1, a_2 in f is $f \circ \langle a_1, a_2 \rangle$ is no more valid: indeed, because of the effects, the result of applying f to a_1, a_2 may depend on the evaluation order of a_1 and a_2. This means that there is no "good" pair $\langle a_1, a_2 \rangle$. However, it is usually possible to give a meaning to "$f(a_1, a_2)$ with a_1 evaluated before a_2", or symmetrically to "$f(a_1, a_2)$ with a_2 evaluated before a_1". This means that there are "good" tuples $\langle a_1 \circ v_1, v_2 \rangle$ and $\langle w_1, a_2 \circ v_2 \rangle$ when v_1, v_2, w_1 and w_2 are either identities or projections. Then, for "a_1 before a_2" one can use $\langle pr_1, a_2 \circ pr_2 \rangle \circ \langle a_1, id_A \rangle$ (which coincides with $\langle a_1, a_2 \rangle$ when this pair does exist). Such a notion of *sequential pair* is studied in [6], where several effects are considered. There are other ways to formalize the fact of first evaluating a_1 then a_2: for instance by using a strong monad [18] or productors [26]; a comparison with strong monads is done in [6].

$$\begin{tikzcd}
 & A_1 & & A_1 & \\
A \arrow[ur, "a_1"] \arrow[r, "\langle a_1, id_A \rangle", "="'] \arrow[dr, "id_A"'] & A_1 \times A \arrow[u] \arrow[d] & A \arrow[ur, "pr_1"] \arrow[r, "\langle pr_1, a_2 \circ pr_2 \rangle", "="'] \arrow[dr, "a_2 \circ pr_2"'] & A_1 \times A_2 \arrow[u] \arrow[d] \arrow[r, "f"] & B \\
 & A & & A_2 &
\end{tikzcd}$$

For conditionals, the fact that `if b then f else g` corresponds to $[f|g] \circ b$ usually remains valid when there are effects.

In this paper, we consider a *language with effects* as a language with (at least) two levels of terms, similar to the *values* and *computations* in [18]: the *pure* terms form the morphisms of a category $\mathbf{C}$ with finite products and coproducts and the *general* terms form a larger category $\mathbf{C}^{(g)}$ with the same objects as $\mathbf{C}$.

Definition 4.1. A language with effects *is compatible with conditionals* when the category $\mathbf{C}$ has finite coproducts and when the copairs of general terms are defined: for each $f_1^{(g)} \colon A_1 \to B$ and $f_2^{(g)} \colon A_2 \to B$ there exists a unique $[f_1|f_2]^{(g)} \colon A_1 + A_2 \to B$ such that $[f_1|f_2] \circ in_1 = f_1$ and $[f_1|f_2] \circ in_2 = f_2$ (where $in_1^{(0)} \colon A_1 \to A_1 + A_2$ and $in_2^{(0)} \colon A_2 \to A_1 + A_2$ are the coprojections).

Definition 4.2. Let $\gg$ be a relation between pure terms and general terms which is the equality when both terms are pure. A language with effects *is compatible with sequential pairs, with respect to* $\gg$, when the category $\mathbf{C}$ has finite products and when the left and right pairs of a pure term and a general term are defined, in the following sense: for each $f_1^{(0)} \colon A \to B_1$ and $f_2^{(g)} \colon A \to B_2$ there exists a unique $\langle f_1, f_2 \rangle_l^{(g)} \colon A \to B_1 \times B_2$ such that $pr_1 \circ \langle f_1, f_2 \rangle_l \gg f_1$ and $pr_2 \circ \langle f_1, f_2 \rangle_l = f_2$ (where $pr_1^{(0)} \colon B_1 \times B_2 \to B_1$ and $pr_2^{(0)} \colon B_1 \times B_2 \to B_2$ are the projections), and symmetrically for $\langle f_1^{(g)}, f_2^{(0)} \rangle_r^{(g)}$.

4.2 States

Let us assume that the category $\mathbf{C}$ is distributive. This means that the canonical morphism from $A \times B + A \times C$ to $A \times (B + C)$ is an isomorphism. Then we get new decorations for the coproduct rules, because the copair of two modifiers now exists, see Fig. 6. The interpretation of the modifier $[f|g]^{(2)}$, when both $f^{(2)}$ and $g^{(2)}$ are modifiers, is the composition of the inverse of the canonical morphism $(A_1 \times S) + (A_2 \times S) \to (A_1 + A_2) \times S$ with $[f_2|g_2] \colon (A_1 \times S) + (A_2 \times S) \to B \times S$.

Theorem 4.3. *Let us consider the language for states with modifiers as general terms (decoration $g = 2$). When the category $\mathbf{C}$ is distributive, the language for states is compatible with conditionals and sequential pairs with respect to $\sim$.*

Proof. The left and right pairs of an accessor and a modifier in the logic $\mathcal{L}_{st}$ (Fig. 5) provide sequential pairs. The rules for copairs in the logic $\mathcal{L}_{st}^{+}$ (Fig. 6) provide conditionals. □

Remark 4.4. An advantage of using the comonad of states $X \times S$ rather than the usual monad of states $(X \times S)^S$ is that sequential pairs for states are defined without any new ingredient: no kind of strength, in contrast with the approach using the strong monad of states $(A \times S)^S$ [18], and no "external" decoration for equations, in contrast with [6].

4.3 Exceptions

Since we do not assume that the category $\mathbf{C}$ is codistributive we do not get pairs of catchers in a way dual to the copairs of modifiers for states. In fact the decorated logic $\mathcal{L}_{exc}$ for exceptions, with the core operations for tagging and untagging, remains *private*, while there is a *programmer's* language, which is *public*, with no direct access to the catchers. The programmer's language for exceptions provides the operations for *raising* and *handling* exceptions, which are defined in terms of the core operations. This language does not include the private tagging and untagging operations, but the public `throw` and `try/catch` constructions, which are defined in terms of `tag` and `untag`. It has no catcher: the only way to catch an exception is by using a `try/catch` expression, which itself propagates exceptions. This corresponds to the usual mechanism of exceptions in programming languages. For the sake of simplicity we assume that only one type of exception is handled in a `try/catch` expression, the general case is treated in Appendix B.

The main ingredients for building the programmer's language from the core language are the coproducts $A \cong A + \mathbb{0}$ and the fact of decorating the composition: in addition to the basic composition "$\circ$" we introduce a second composition, called the *propagator composition* and denoted "$\odot$", subject to the rules in Fig. 8. Both compositions "$\circ$" and "$\odot$" coincide on propagators, but they are interpreted differently when a propagator is composed with a modifier. This is an instance of the two ways to compose oblique morphisms related to an adjunction [19].

Remark 4.5. In fact, this new composition can be defined for any monad, but until now it has not been needed: Let $f^{(1)} : A \to B$ and $k^{(2)} : B \to C$ then $(k \odot f)^{(1)} : A \to C$ is interpreted as $k_2 \circ f_1 : A \to MC$; then it can be checked that $f \sim g$ if and only if $id \odot f \cong id \odot g$. In contrast, $(k \circ f)^{(2)} : A \to C$ is interpreted as $k_2 \circ f_2 = k_2 \circ \mu_B \circ Mf_1 : MA \to MC$. Dually, such a new composition could be defined for any comonad.

Now, we come back to exceptions and we define the `throw` and `try`/`catch` constructions.

Definition 4.6. For each type B and each exception name T, the propagator $\mathtt{throw}^{(1)}_{B,T}$ is:

$$\mathtt{throw}^{(1)}_{B,T} = [\ ]^{(0)}_B \circ \mathtt{tag}^{(1)}_T : V_T \to B$$

For each each propagator $f^{(1)} : A \to B$, each exception name T and each propagator $g^{(1)} : V_T \to B$, the propagator $\mathtt{try}(f)\mathtt{catch}(T \Rightarrow g)^{(1)}$ is defined as follows, in two steps:

$$\mathtt{catch}(T \Rightarrow g)^{(2)} = [\ g^{(1)} \mid [\ ]^{(0)}_B\ ]^{(1)} \circ \mathtt{untag}^{(2)}_T : \mathbb{0} \to B$$
$$\mathtt{try}(f)\mathtt{catch}(T \Rightarrow g)^{(1)} = [\ id_B \mid \mathtt{catch}(T \Rightarrow g)\ ]^{(2)}_l \odot f^{(1)} : A \to B$$

This means that raising an exception with name T in a type B consists in tagging the given ordinary value (in V_T) as an exception and coerce it to B. For handling an exception, the intermediate expression $\mathtt{catch}(T \Rightarrow g)$ is a private catcher while the expression $\mathtt{try}(f)\mathtt{catch}(T \Rightarrow g)$ is a public propagator: the propagator composition "$\odot$" prevents this expression from catching exceptions with name T which might have been raised before the $\mathtt{try}(f)\mathtt{catch}(T \Rightarrow g)$ block is considered. The definition of $\mathtt{try}(f)\mathtt{catch}(T \Rightarrow g)$ corresponds to the Java mechanism for exceptions [11, 13], which may be described by the control flow in Fig. 7, where "*exc*?" means "*is this value an exception?*", an *abrupt* termination returns an uncaught exception and a *normal* termination returns an ordinary value. Now, let us assume that the category $\mathbf{C}$ is *extensive with respect to* E, by which we mean that the pullbacks of the coprojections $in_1 : B \to B+E$ and $in_2 : E \to B + E$ along an arbitrary morphism $f : A \to B + E$ exist and form a coproduct $A = \mathcal{D}_f + \mathcal{E}_f$:

$$\begin{tikzcd}
\mathcal{D}_f \arrow[r, "f_{normal}"] \arrow[d, "i_f"'] & B \arrow[d, "in_1"] \\
A \arrow[r, "f_1"] & B + E \\
\mathcal{E}_f \arrow[r, "f_{abrupt}"] \arrow[u, "j_f"] & E \arrow[u, "in_2"']
\end{tikzcd}$$

Informally, this implies that any morphism $f_1 : A \to B + E$ can be seen as a partial morphism form A to B with domain of definition the vertex $\mathcal{D}_f$ of the pullback on in_1 and f_1. We get a decorated logic $\mathcal{L}^+_{exc}$ by extending $\mathcal{L}_{exc}$ with the propagator composition and with left pairs (and right ones, omitted here)

as in Fig. 8. We define a relation $\gg$ between pure terms and propagators, which can be seen as (a restriction of) the usual order between partial functions.

Definition 4.7. Let $v^{(0)}\colon A \to B$ be a pure term and $f^{(1)}\colon A \to B$ a propagator, corresponding respectively to $v_0\colon A \to B$ and $f_1\colon A \to B + E$ in $\mathbf{C}$. Then $v^{(0)} \gg f^{(1)}$ if and only if the restrictions of v_0 and f_1 to the domain of definition of f_1 coincide, which means, if and only if $v_0 \circ i_f = f_{normal}\colon \mathcal{D}_f \to B$.

Now we can interpret the left pair of a pure term and a propagator.

Definition 4.8. Let $v^{(0)}\colon A \to B_1$ be a pure term and $f^{(1)}\colon A \to B_2$ a propagator, corresponding respectively to $v_0\colon A \to B_1$ and $f_1\colon A \to B_2 + E$ in $\mathbf{C}$. Let $h_{normal} = \langle v \circ i_f, f_{normal}\rangle\colon \mathcal{D}_g \to B_1 \times B_2$ and $h_{abrupt} = f_{abrupt}\colon \mathcal{E}_g \to E$, then let $h = h_{normal} + h_{abrupt}\colon A \to (B_1 \times B_2) + E$ in $\mathbf{C}$. The morphism h in $\mathbf{C}$ corresponds to a propagator $h^{(1)}\colon A \to B_1 \times B_2$, which is the interpretation of the *left pair* $\langle v, f\rangle_l^{(1)}$ of $v^{(0)}$ and $f^{(1)}$.

It is easy to check that indeed $h^{(1)}$ satisfies the properties required of left pairs in Fig. 8. The right pair of a propagator and a pure term is defined in a symmetric way. It can easily be checked that the core language for exceptions with catchers as general terms (decoration $g = 2$) is not compatible with conditionals and sequential pairs (with respect to any relation $\gg$).

Theorem 4.9. *Let us consider the programmer's language for exceptions with propagators as general terms (decoration $g = 1$). When the category $\mathbf{C}$ is extensive with respect to E, the programmer's language for exceptions is compatible with conditionals and sequential pairs with respect to $\gg$ as in Definition 4.7.*

Proof. The left and right pairs of a pure term and a propagator in the logic $\mathcal{L}_{exc}^{+}$ (Fig. 8) provide sequential pairs. The rules for copairs in the logic $\mathcal{L}_{mon}$ (Fig. 2) provide conditionals. □

References

[1] Andrej Bauer, Matija Pretnar. An Effect System for Algebraic Effects and Handlers. CALCO 2013: 1-16

[2] Aurelio Carboni, Steve Lack, R.F.C. Walters. Introduction to extensive and distributive categories. Journal of Pure and Applied Algebra 84(2):145-158, 1993.

[3] César Domínguez, Dominique Duval. Diagrammatic logic applied to a parameterization process. Mathematical Structures in Computer Science 20, p. 639-654 (2010).

[4] Jean-Guillaume Dumas, Dominique Duval, Laurent Fousse, Jean-Claude Reynaud. Decorated proofs for computational effects: States. ACCAT 2012. Electronic Proceedings in Theoretical Computer Science 93, p. 45-59 (2012).

[5] Jean-Guillaume Dumas, Dominique Duval, Laurent Fousse, Jean-Claude Reynaud. A duality between exceptions and states. Mathematical Structures in Computer Science 22, p. 719-722 (2012).

[6] Jean-Guillaume Dumas, Dominique Duval, Jean-Claude Reynaud. Cartesian effect categories are Freyd-categories. Journal of Symbolic Computation 46, p. 272-293 (2011).

[7] Jean-Guillaume Dumas, Dominique Duval, Jean-Claude Reynaud. A decorated proof system for exceptions. arXiv:1310.2338 (2013).

[8] Jean-Guillaume Dumas, Dominique Duval, Burak Ekici, Damien Pous. Formal verification in Coq of program properties involving the global state effect. JFLA'2014: Journes Francophones des Langages Applicatifs, Frjus, France, Janvier 2014. arXiv:1310.0794.

[9] Jeremy Gibbons, Michael Johnson. Relating Algebraic and Coalgebraic Descriptions of Lenses BX 2012. ECEASST 49 (2012).

[10] Hughes, J.. Generalising monads to arrows. Sci. of Comput. Program. 37(13) (2000), pp. 67111.

[11] James Gosling, Bill Joy, Guy Steele, Gilad Bracha. The Java Language Specification, Third Edition. Addison-Wesley Longman (2005).

[12] Martin Hyland, Gordon D. Plotkin, John Power. Combining effects: Sum and tensor. Theor. Comput. Sci. 357(1-3): 70-99 (2006)

[13] Bart Jacobs. A Formalisation of Java's Exception Mechanism. ESOP 2001. LNCS, Vol. 2028, p. 284-301 Springer (2001).

[14] Bart Jacobs. Bases as Coalgebras. Logical Methods in Computer Science 9(3) (2013).

[15] Paul Blain Levy. Monads and adjunctions for global exceptions. MFPS 2006. Electronic Notes in Theoretical Computer Science 158, p. 261-287 (2006).

[16] John M. Lucassen, David K. Gifford. Polymorphic effect systems. POPL 1988. ACM Press, p. 47-57.

[17] Saunders Mac Lane. Categories for the Working Mathematician. Springer, 2nd ed. 1978.

[18] Eugenio Moggi. Notions of Computation and Monads. Information and Computation 93(1), p. 55-92 (1991).

[19] Guillaume Munch. Models of a Non-Associative Composition. To appear in Proc. FoSSaCS 2014.

[20] Tomas Petricek, Dominic A. Orchard, Alan Mycroft: Coeffects: Unified Static Analysis of Context-Dependence. ICALP (2) 2013: 385-397

[21] Andrew M. Pitts. Categorical Logic. Chapter 2 of S. Abramsky and D. M. Gabbay and T. S. E. Maibaum (Eds) Handbook of Logic in Computer Science, Volume 5. Algebraic and Logical Structures, Oxford University Press, 2000.

[22] Gordon D. Plotkin, John Power. Notions of Computation Determine Monads. FoSSaCS 2002. LNCS, Vol. 2620, p. 342-356, Springer (2002).

[23] Gordon D. Plotkin, John Power: Algebraic Operations and Generic Effects. Applied Categorical Structures 11(1): 69-94 (2003)

[24] Gordon D. Plotkin, Matija Pretnar. Handlers of Algebraic Effects. ESOP 2009. LNCS, Vol. 5502, p. 80-94, Mpringer (2009).

[25] Power, J. and E. Robinson. Premonoidal categories and notions of computation. Math. Structures in Comput. Sci. 7(5) (1997), pp. 453468.

[26] Ross Tate. The sequential semantics of producer effect systems. POPL 2013. ACM Press, p. 15-26 (2013).

[27] Tarmo Uustalu, Varmo Vene. Comonadic Notions of Computation. CMCS 2008. ENTCS 203, p. 263-284 (2008).

[28] Philip Wadler. The essence of functional programming. POPL 1992. ACM Press, p. 1-14 (1992).

[29] Philip Wadler. Call-by-Value Is Dual to Call-by-Name - Reloaded. RTA 2005: 185-203

A The decorated logic for a monad

The decorated logic $\mathcal{L}_{mon}$ for a monad when $\mathbf{C}$ is bicartesian.

Grammar

Types: $t ::= A \mid B \mid \cdots \mid t \times t \mid 1 \mid t + t \mid 0$

Terms: $f ::= id_t \mid f \circ f \mid \langle f, f \rangle \mid pr_{t,t,1} \mid pr_{t,t,2} \mid \langle\ \rangle_t \mid [f|f] \mid in_{t,t,1} \mid in_{t,t,2} \mid [\,]_t$

Decoration for terms: $(d) ::= (0) \mid (1) \mid (2)$

Equations: $e ::= f \cong f \mid f \sim f$

Conversion rules

$$\frac{f^{(0)}}{f^{(1)}} \qquad \frac{f^{(1)}}{f^{(2)}} \qquad \frac{f^{(d)} \cong g^{(d')}}{f \sim g} \qquad \frac{f^{(d)} \sim g^{(d')}}{f \cong g} \text{ if } d, d' \leq 1$$

Equivalence rules

$$\text{(s-refl)}\ \frac{f^{(d)}}{f \cong f} \qquad \text{(s-sym)}\ \frac{f^{(d)} \cong g^{(d')}}{g \cong f} \qquad \text{(s-trans)}\ \frac{f^{(d)} \cong g^{(d')} \quad g^{(d')} \cong h^{(d'')}}{f \cong h}$$

$$\text{(w-refl)}\ \frac{f^{(d)}}{f \sim f} \qquad \text{(w-sym)}\ \frac{f^{(d)} \sim g^{(d')}}{g^{(d')} \sim f} \qquad \text{(w-trans)}\ \frac{f^{(d)} \sim g^{(d')} \quad g^{(d')} \sim h^{(d'')}}{f \sim h}$$

Categorical rules

$$\text{(id)}\ \frac{A}{id_A^{(0)} : A \to A} \qquad \text{(comp)}\ \frac{f^{(d)} : A \to B \quad g^{(d')} : B \to C}{(g \circ f)^{(max(d,d'))} : A \to C}$$

$$\text{(id-source)}\ \frac{f^{(d)} : A \to B}{f \circ id_A \cong f} \qquad \text{(id-target)}\ \frac{f^{(d)} : A \to B}{id_B \circ f \cong f}$$

$$\text{(assoc)}\ \frac{f^{(d)} : A \to B \quad g^{(d')} : B \to C \quad h^{(d'')} : C \to D}{h \circ (g \circ f) \cong (h \circ g) \circ f}$$

Congruence rules

$$\text{(s-repl)}\ \frac{f_1^{(d_1)} \cong f_2^{(d_2)} : A \to B \quad g^{(d)} : B \to C}{g \circ f_1 \cong g \circ f_2} \qquad \text{(s-subs)}\ \frac{f^{(d)} : A \to B \quad g_1^{(d_1)} \cong g_2^{(d_2)} : B \to C}{g_1 \circ f \cong g_2 \circ f}$$

$$\text{(w-repl)}\ \frac{f_1^{(d_1)} \sim f_2^{(d_2)} : A \to B \quad g^{(d)} : B \to C}{g \circ f_1 \sim g \circ f_2} \qquad \text{(w-subs)}\ \frac{f^{(0)} : A \to B \quad g_1^{(d_1)} \sim g_2^{(d_2)} : B \to C}{g_1 \circ f \sim g_2 \circ f}$$

Product rules

$$\text{(prod)}\ \frac{B_1 \quad B_2}{pr_1^{(0)} : B_1 \times B_2 \to B_1 \quad pr_2^{(0)} : B_1 \times B_2 \to B_2}$$

$$\text{(pair)}\ \frac{f_1^{(0)} : A \to B_1 \quad f_2^{(0)} : A \to B_2}{\langle f_1, f_2 \rangle^{(0)} : A \to B_1 \times B_2 \quad pr_1 \circ \langle f_1, f_2 \rangle \cong f_1 \quad pr_2 \circ \langle f_1, f_2 \rangle \cong f_2}$$

$$\text{(pair-u)}\ \frac{f_1^{(0)} : A \to B_1 \quad f_2^{(0)} : A \to B_2 \quad g^{(0)} : A \to B_1 \times B_2 \quad pr_1 \circ g \cong f_1 \quad pr_2 \circ g \cong f_2}{g \cong \langle f_1, f_2 \rangle}$$

$$\text{(final)}\ \frac{A}{\langle\ \rangle_A^{(0)} : A \to 1} \qquad \text{(final-u)}\ \frac{f^{(0)} : A \to 1}{f \cong \langle\ \rangle_A}$$

Coproduct rules

$$\text{(coprod)}\ \frac{A_1 \quad A_2}{in_1^{(0)} : A_1 \to A_1 + A_2 \quad in_2^{(0)} : A_2 \to A_1 + A_2}$$

$$\text{(copair)}\ \frac{f_1^{(d_1)} : A_1 \to B \quad f_2^{(d_2)} : A_2 \to B}{[f_1|f_2]^{(max(d_1,d_2))} : A_1 + A_2 \to B \quad [f_1|f_2] \circ in_1 \cong f_1 \quad [f_1|f_2] \circ in_2 \cong f_2} (d_1, d_2 \leq 1)$$

$$\text{(copair-u)}\ \frac{f_1^{(d_1)} : A_1 \to B \quad f_2^{(d_2)} : A_2 \to B \quad g^{(d)} : A_1 + A_2 \to B \quad g \circ in_1 \cong f_1 \quad g \circ in_2 \cong f_2}{g \cong [f_1|f_2]} (d_1, d_2, d \leq 1)$$

$$\text{(initial)}\ \frac{B}{[\,]_B^{(0)} : 0 \to B} \qquad \text{(initial-u)}\ \frac{f^{(d)} : 0 \to B}{f \sim [\,]_B}$$

B Catching several exception names

The handling process is easily extended to several exception names, as follows. The index T_i is simplified as i: $V_i = V_{T_i}$, $\mathtt{tag}_i = \mathtt{tag}_{T_i}$, $\mathtt{untag}_i = \mathtt{untag}_{T_i}$.

Definition B.1. For each each propagator $f^{(1)} \colon A \to B$, each list of exception names $(T_1, \ldots, T_n)$ and each propagators $g_j^{(1)} \colon V_i \to B$ for $i = 1, \ldots, n$, the propagator $\mathtt{try}(f)\mathtt{catch}\,(T_1 \Rightarrow g_1 | \ldots | T_n \Rightarrow g_n)^{(1)} \colon A \to B$ is defined as follows, in two steps:

- the catcher $\mathtt{catch}\,(T_1 \Rightarrow g_1 | \ldots | T_n \Rightarrow g_n)^{(2)} : \mathbb{0} \to B$ is obtained by setting $i = 1$ in the family of catchers $k_i^{(2)} = \mathtt{catch}\,(T_i \Rightarrow g_i | \ldots | T_n \Rightarrow g_n) : \mathbb{0} \to B$ (for $i = 1, \ldots, n$) which are defined recursively by:

$$k_i^{(2)} \;=\; \begin{cases} [\; g_n^{(1)} \mid [\;]_B^{(0)} \;]^{(1)} \circ \mathtt{untag}_n^{(2)} & \text{when } i = n \\ [\; g_i^{(1)} \mid k_{i+1}^{(2)} \;]_l^{(2)} \circ \mathtt{untag}_i^{(2)} & \text{when } i < n \end{cases}$$

- then the required propagator is:

$$\begin{aligned} &\mathtt{try}(f)\mathtt{catch}\,(T_1 \Rightarrow g_1 | \ldots | T_n \Rightarrow g_n)^{(1)} = \\ &\qquad [\; id_B \mid \mathtt{catch}\,(T_1 \Rightarrow g_1 | \ldots | T_n \Rightarrow g_n) \;]_l^{(2)} \odot f^{(1)} \colon A \to B \end{aligned}$$

The handling process is also easily extended to all exception names. This *catch-all* construction is similar to `catch(...)` in C++ or to `(except, else)` in Python. We add a catcher $\mathtt{untag}_{\mathtt{all}}^{(2)} \colon \mathbb{0} \to \mathbb{1}$ with the equations

$$\mathtt{untag}_{\mathtt{all}} \circ \mathtt{tag}_T \sim \langle\;\rangle_T$$

for every exception name T, which means that $\mathtt{untag}_{\mathtt{all}}$ catches exceptions of the form $\mathtt{tag}_T(a)$ for every T and forgets the value a.

Definition B.2. For each propagators $f^{(1)} \colon A \to B$ and $g^{(1)} \colon \mathbb{1} \to B$, the propagator $\mathtt{try}(f)\mathtt{catch}(\mathtt{all} \Rightarrow g)^{(1)} \colon A \to B$ is:

$$\mathtt{try}(f)\mathtt{catch}(\mathtt{all} \Rightarrow g)^{(1)} = [\; id_B \,|\, g \circ \mathtt{untag}_{\mathtt{all}} \;]_l^{(2)} \odot f^{(1)} \colon A \to B$$

The interpretation of $\mathtt{try}(f)\mathtt{catch}(\mathtt{all} \Rightarrow g)$ is "*handle the exception e raised in f, if any, with g*". This may be combined with other catchers, and every catcher following a *catch-all* is syntactically allowed, but never executed.

Grammar

Types: $t ::= A \mid B \mid \cdots \mid t \times t \mid \mathbb{1} \mid t + t \mid \mathbb{0}$

Terms: $f ::= id_t \mid f \circ f \mid \langle f, f \rangle \mid pr_{t,t,1} \mid pr_{t,t,2} \mid \langle\ \rangle_t \mid [f|f] \mid in_{t,t,1} \mid in_{t,t,2} \mid [\,]_t$

Equations: $e ::= f \equiv f$

Equivalence rules

(refl) $\dfrac{f}{f \equiv f}$ (sym) $\dfrac{f \equiv g}{g \equiv f}$ (trans) $\dfrac{f \equiv g \quad g \equiv h}{f \equiv h}$

Categorical rules

(id) $\dfrac{A}{id_A \colon A \to A}$ (comp) $\dfrac{f \colon A \to B \quad g \colon B \to C}{(g \circ f) \colon A \to C}$

(id-source) $\dfrac{f \colon A \to B}{f \circ id_A \equiv f}$ (id-target) $\dfrac{f \colon A \to B}{id_B \circ f \equiv f}$

(assoc) $\dfrac{f \colon A \to B \quad g \colon B \to C \quad h \colon C \to D}{h \circ (g \circ f) \equiv (h \circ g) \circ f}$

Congruence rules

(repl) $\dfrac{f_1 \equiv f_2 \colon A \to B \quad g \colon B \to C}{g \circ f_1 \equiv g \circ f_2}$ (subs) $\dfrac{f \colon A \to B \quad g_1 \equiv g_2 \colon B \to C}{g_1 \circ f \equiv g_2 \circ f}$

Product rules

(prod) $\dfrac{B_1 \quad B_2}{pr_1 \colon B_1 \times B_2 \to B_1 \quad pr_2 \colon B_1 \times B_2 \to B_2}$

(pair) $\dfrac{f_1 \colon A \to B_1 \quad f_2 \colon A \to B_2}{\langle f_1, f_2 \rangle \colon A \to B_1 \times B_2 \quad pr_1 \circ \langle f_1, f_2 \rangle \equiv f_1 \quad pr_2 \circ \langle f_1, f_2 \rangle \equiv f_2}$

(pair-u) $\dfrac{f_1 \colon A \to B_1 \quad f_2 \colon A \to B_2 \quad g \colon A \to B_1 \times B_2 \quad pr_1 \circ g \equiv f_1 \quad pr_2 \circ g \equiv f_2}{g \equiv \langle f_1, f_2 \rangle}$

(final) $\dfrac{A}{\langle\ \rangle_A \colon A \to \mathbb{1}}$ (final-u) $\dfrac{f \colon A \to \mathbb{1}}{f \equiv \langle\ \rangle_A}$

Coproduct rules

(coprod) $\dfrac{A_1 \quad A_2}{in_1 \colon A_1 \to A_1 + A_2 \quad in_2 \colon A_2 \to A_1 + A_2}$

(copair) $\dfrac{f_1 \colon A_1 \to B \quad f_2 \colon A_2 \to B}{[f_1|f_2] \colon A_1 + A_2 \to B \quad [f_1|f_2] \circ in_1 \equiv f_1 \quad [f_1|f_2] \circ in_2 \equiv f_2}$

(copair-u) $\dfrac{f_1 \colon A_1 \to B \quad f_2 \colon A_2 \to B \quad g \colon A_1 + A_2 \to B \quad g \circ in_1 \equiv f_1 \quad g \circ in_2 \equiv f_2}{g \equiv [f_1|f_2]}$

(initial) $\dfrac{B}{[\,]_B \colon \mathbb{0} \to B}$ (initial-u) $\dfrac{f \colon \mathbb{0} \to B}{f \equiv [\,]_B}$

Figure 1: $\mathcal{L}_{eq}$: the equational logic with conditionals

Conversion rules

(pure-acc) $\dfrac{f^{(0)}}{f^{(1)}}$ (acc-mod) $\dfrac{f^{(1)}}{f^{(2)}}$

(strong-weak) $\dfrac{f^{(d)} \cong g^{(d')}}{f \sim g}$ (weak-strong) $\dfrac{f^{(d)} \sim g^{(d')}}{f \cong g}(d, d' \leq 1)$

Weak substitution rule

(w-subs) $\dfrac{f^{(0)} \colon A \to B \quad g_1^{(d)} \sim g_2^{(d')} \colon B \to C}{g_1 \circ f \sim g_2 \circ f}$

Coproduct rules

(coprod) $\dfrac{A_1 \quad A_2}{in_1^{(0)} \colon A_1 \to A_1 + A_2 \quad in_2^{(0)} \colon A_2 \to A_1 + A_2}$

(copair) $\dfrac{f_1^{(d_1)} \colon A_1 \to B \quad f_2^{(d_2)} \colon A_2 \to B}{[f_1|f_2]^{(max(d_1,d_2))} \colon A_1 + A_2 \to B \quad [f_1|f_2] \circ in_1 \cong f_1 \quad [f_1|f_2] \circ in_2 \cong f_2}(d_1, d_2 \leq 1)$

(copair-u) $\dfrac{f_1^{(d_1)} \colon A_1 \to B \quad f_2^{(d_2)} \colon A_2 \to B \quad g^{(d)} \colon A_1 + A_2 \to B \quad g \circ in_1 \cong f_1 \quad g \circ in_2 \cong f_2}{g \cong [f_1|f_2]}(d_1, d_2, d \leq 1)$

(initial) $\dfrac{B}{[\,]_B^{(0)} \colon \mathbb{0} \to B}$ (initial-u) $\dfrac{f^{(2)} \colon \mathbb{0} \to B}{f \sim [\,]_B}$

Figure 2: $\mathcal{L}_{mon}$: some decorated rules for a monad

Conversion rules

$$\frac{f^{(0)}}{f^{(1)}} \qquad \frac{f^{(1)}}{f^{(2)}} \qquad \frac{f^{(d)} \cong g^{(d')}}{f \sim g} \text{ for all } d, d' \qquad \frac{f^{(d)} \sim g^{(d')}}{f \cong g} \text{ for all } d, d' \leq 1$$

Weak replacement rule

$$\text{(w-repl)} \quad \frac{f_1^{(d)} \sim f_2^{(d')} \colon A \to B \quad g^{(0)} \colon B \to C}{g \circ f_1 \sim g \circ f_2}$$

Product rules

$$\text{(prod)} \quad \frac{B_1 \quad B_2}{pr_1^{(0)} \colon B_1 \times B_2 \to B_1 \quad pr_2^{(0)} \colon B_1 \times B_2 \to B_2}$$

$$\text{(pair)} \quad \frac{f_1^{(d_1)} \colon A \to B_1 \quad f_2^{(d_2)} \colon A \to B_2}{\langle f_1, f_2 \rangle^{(max(d_1,d_2))} \colon A \to B_1 \times B_2 \quad pr_1 \circ \langle f_1, f_2 \rangle \cong f_1 \quad pr_2 \circ \langle f_1, f_2 \rangle \cong f_2} (d_1, d_2 \leq 1)$$

$$\text{(pair-u)} \quad \frac{f_1^{(d_1)} \colon A \to B_1 \quad f_2^{(d_2)} \colon A \to B_2 \quad g^{(d)} \colon A \to B_1 \times B_2 \quad pr_1 \circ g \cong f_1 \quad pr_2 \circ g \cong f_2}{g \cong \langle f_1, f_2 \rangle} (d_1, d_2, d \leq 1)$$

$$\text{(final)} \quad \frac{A}{\langle\, \rangle_A^{(0)} \colon A \to \mathbb{1}} \qquad \text{(final-u)} \quad \frac{f^{(2)} \colon A \to \mathbb{1}}{f \sim \langle\, \rangle_A}$$

Figure 3: $\mathcal{L}_{comon}$: some decorated rules for a comonad

Additional (left) coproduct rules

(l-copair) $$\frac{f_1^{(1)}\colon A_1 \to B \quad f_2^{(2)}\colon A_2 \to B}{[f_1|f_2]_l^{(2)}\colon A_1 + A_2 \to B \quad [f_1|f_2]_l \circ in_1 \sim f_1 \quad [f_1|f_2]_l \circ in_2 \cong f_2}$$

(l-copair-u) $$\frac{g^{(2)}\colon A_1 + A_2 \to B \;\; f_1^{(1)}\colon A_1 \to B \;\; f_2^{(2)}\colon A_2 \to B \;\; g \circ in_1 \sim f_1 \;\; g \circ in_2 \cong f_2}{g \cong [f_1|f_2]_l}$$

Effect rule

(effect) $$\frac{f, g\colon A \to B \quad f \sim g \quad f \circ [\,]_A \cong g \circ [\,]_A}{f \cong g}$$

Additional grammar (for each $T \in Exn$)

Types: V_T

Terms: $\mathtt{tag}_T^{(1)}\colon V_T \to \mathbb{0} \mid \mathtt{untag}_T^{(2)}\colon \mathbb{0} \to V_T$

Axioms (for each $T \in Exn$)

$\mathtt{untag}_T \circ \mathtt{tag}_T \sim id_{V_T}$

$\mathtt{untag}_T \circ \mathtt{tag}_R \sim [\,]_{V_T} \circ \mathtt{tag}_R$ for each $R \neq T,\ R \in Exn$

A specific coproduct rule

(exc-coprod-u) $$\frac{f, g\colon \mathbb{0} \to B \quad \text{for all } T \in Exn\ f \circ \mathtt{tag}_T \sim g \circ \mathtt{tag}_T}{f \cong g}$$

Figure 4: From $\mathcal{L}_{mon}$ to $\mathcal{L}_{exc}$: additional features for the monad of exceptions

Additional (left) product rules

(l-pair) $$\frac{f_1^{(1)}\colon A \to B_1 \quad f_2^{(2)}\colon A \to B_2}{\langle f_1, f_2\rangle_l^{(2)}\colon A \to B_1 \times B_2 \quad pr_1 \circ \langle f_1, f_2\rangle_l \sim f_1 \quad pr_2 \circ \langle f_1, f_2\rangle_l \cong f_2}$$

(l-pair-u) $$\frac{g^{(2)}\colon A \to B_1 \times B_2 \;\; f_1^{(1)}\colon A \to B_1 \;\; f_2^{(2)}\colon A \to B_2 \;\; pr_1 \circ g \sim f_1 \;\; pr_2 \circ g \cong f_2}{g \cong \langle f_1, f_2\rangle_l}$$

Effect rule

(st-effect-u) $$\frac{f, g\colon A \to B \quad f \sim g \quad \langle\,\rangle_A \circ f \cong \langle\,\rangle_A \circ g}{f \cong g}$$

Additional grammar (for each $T \in Loc$)

Types: V_T

Terms: $\mathtt{lookup}_T^{(1)}\colon \mathbb{1} \to V_T \mid \mathtt{update}_T^{(2)}\colon V_T \to \mathbb{1}$

Axioms (for each $T \in Loc$)

$\mathtt{lookup}_T \circ \mathtt{update}_T \sim id_{V_T}$

$\mathtt{lookup}_R \circ \mathtt{update}_T \sim \mathtt{lookup}_R \circ \langle\,\rangle_{V_T}$ for each $R \neq T,\ R \in Loc$

A specific product rule

(st-prod-u) $$\frac{f, g\colon A \to \mathbb{1} \quad \text{for all } T \in Loc\ \mathtt{lookup}_T \circ f \sim \mathtt{lookup}_T \circ g}{f \cong g}$$

Figure 5: From $\mathcal{L}_{comon}$ to $\mathcal{L}_{st}$: additional features for the comonad of states

Additional coproduct rules

(copair) $$\frac{f_1^{(2)}\colon A_1 \to B \quad f_2^{(2)}\colon A_2 \to B}{[f_1|f_2]^{(2)}\colon A_1 + A_2 \to B \quad [f_1|f_2] \circ in_1 \cong f_1 \quad [f_1|f_2] \circ in_2 \cong f_2}$$

(copair-u) $$\frac{f_1^{(2)}\colon A_1 \to B \;\; f_2^{(2)}\colon A_2 \to B \;\; g^{(2)}\colon A_1 + A_2 \to B \;\; g \circ in_1 \cong f_1 \;\; g \circ in_2 \cong f_2}{g \cong [f_1|f_2]}$$

Figure 6: From $\mathcal{L}_{st}$ to $\mathcal{L}_{st}^{+}$: additional rules for states, when $\mathbf{C}$ is distributive

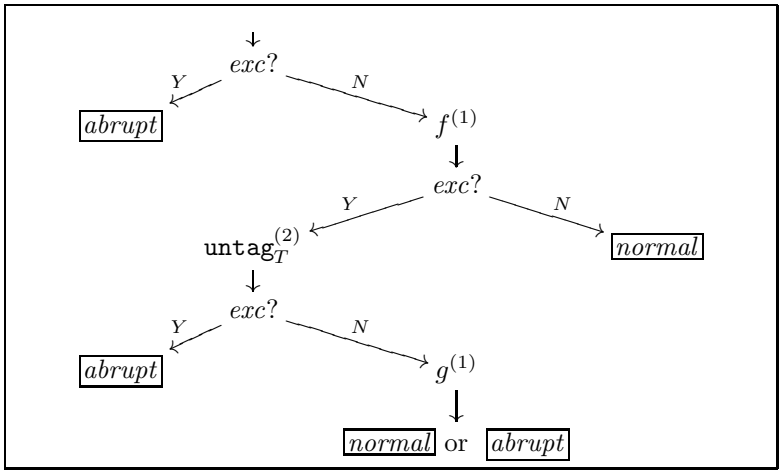


Figure 7: The control flow for $\texttt{try}(f)\texttt{catch}(T \Rightarrow g)$

Propagator composition

$$\text{(prop-comp)} \quad \frac{f^{(1)} \colon A \to B \quad g^{(2)} \colon B \to C}{(g \odot f)^{(1)} \colon A \to C \quad g \odot f \sim g \circ f}$$

Additional (left) product rules

$$\text{(l-pair)} \quad \frac{f_1^{(0)} \colon A \to B_1 \quad f_2^{(1)} \colon A \to B_2}{\langle f_1, f_2 \rangle_l^{(1)} \colon A \to B_1 \times B_2 \quad pr_1 \circ \langle f_1, f_2 \rangle_l \ll f_1 \quad pr_2 \circ \langle f_1, f_2 \rangle_l \sim f_2}$$

$$\text{(l-pair-u)} \quad \frac{f_1^{(0)} \colon A \to B_1 \quad f_2^{(1)} \colon A \to B_2 \quad g^{(1)} \colon A \to B_1 \times B_2 \quad pr_1 \circ g \ll f_1 \quad pr_2 \circ g \cong f_2}{g \cong \langle f_1, f_2 \rangle_l}$$

Figure 8: From $\mathcal{L}_{exc}$ to $\mathcal{L}_{exc}^{+}$: additional rules for exceptions, when $\mathbf{C}$ is extensive wrt E