HAL
open science

Patterns for computational effects arising from a monad or a comonad

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

To cite this version:

Jean-Guillaume Dumas, Dominique Duval, Jean-Claude Reynaud. Patterns for computational effects arising from a monad or a comonad. [Research Report] LJK. 2013. hal-00868831v2

HAL Id: hal-00868831

https://hal.science/hal-00868831v2

Submitted on 14 Oct 2013

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.

Patterns for computational effects arising from a monad or a comonad

Jean-Guillaume Dumas* Dominique Duval*

Jean-Claude Reynaud†

October 14, 2013

Abstract

This paper presents equational-based logics for proving first order properties of programming languages involving effects.

We propose two dual inference system patterns that can be instanciated with monads or comonads in order to be used for proving properties of different effects. The first pattern provides inference rules which can be interpreted in the Kleisli category of a monad and the coKleisli category of the associated comonad. In a dual way, the second pattern provides inference rules which can be interpreted in the coKleisli category of a comonad and the Kleisli category of the associated monad. The logics combine a 3-tier effect system for terms consisting of pure terms and two other kinds of effects called 'constructors/observers' and 'modifiers', and a 2-tier system for 'up-to-effects' and 'strong' equations. Each pattern provides generic rules for dealing with any monad (respectively comonad), and it can be extended with specific rules for each effect. The paper presents two use cases: a language with exceptions (using the standard monadic semantics), and a language with state (using the less standard comonadic semantics). Finally, we prove that the obtained inference system for states is Hilbert-Post complete.

1 Introduction

A *software design pattern* is not a finished design, it is a description or template that can be instanciated in order to be used in many different situations. In this paper, we propose *inference system patterns* that can be instanciated with monads or comonads in order to be used for proving properties of different effects.

*Laboratoire J. Kuntzmann, Université de Grenoble. 51, rue des Mathématiques, umr CNRS 5224, bp 53X, F38041 Grenoble, France, {Jean-Guillaume.Dumas,Dominique.Duval}@imag.fr.

†Reynaud Consulting (RC), Jean-Claude.Reynaud@imag.fr.

In order to formalize computational effects one can choose between types and effects systems [12], monads [14] and their associated Lawvere theories [17], comonads [22], or decorated logics [1]. Starting with Moggi's seminal paper [14] and its application to Haskell [23], various papers deal with the effects arising from a monad, for instance [17, 19, 11, 18].

Each of these approaches rely on some classification of the syntactic expressions according to their interaction with effects. In this paper we use decorated logics which, by extending this classification to equations, provide a proof system adapted to each effect.

This paper presents equational-based logics for proving first order properties of programming languages involving effects. We propose two dual patterns, consisting in a language with an inference system, for building such a logic.

The first pattern provides inference rules which can be interpreted in the coKleisli category of a comonad and the Kleisli category of the associated monad. In a dual way, the second pattern provides inference rules which can be interpreted in the Kleisli category of a monad and the coKleisli category of the associated comonad. The logics combine a three-levels effect system for terms consisting of pure terms and two other kinds of effects called observers/constructors and modifiers, and a two-levels system for strong and weak equations.

Each pattern provides generic rules for dealing with any comonad (respectively monad), and it can be extended with specific rules for each effect. The paper presents two use cases: a language with state and a language with exceptions. For the language with state we use a comonadic semantics and we prove that the equational theory obtained is Hilbert-Post complete, which provides a new proof for a result in [16]. For the language with exceptions we extend the standard monadic semantics in order to catch exceptions; this relies on the duality between states and exceptions from [3].

We do not claim that each effect arises either from a comonad or from a monad, but this paper only deals with such effects. Intuitively, an effect which observes features may arise from a comonad, while an effect which constructs features may arise from a monad [10]. However, some interesting features in the comonad pattern stem from the well-known fact that each comonad determines a monad on its coKleisli category, and dually for the monad pattern. More precisely, on the monads side, let (M, η, μ) be a monad on a category $\mathbf{C}^{(0)}$ and let $\mathbf{C}^{(1)}$ be the Kleisli category of (M, η, μ) on $\mathbf{C}^{(0)}$. Then M can be seen as the endofunctor of a comonad (M, ε, δ) on $\mathbf{C}^{(1)}$, so that we may consider the coKleisli category $\mathbf{C}^{(2)}$ of (M, ε, δ) on $\mathbf{C}^{(1)}$. The canonical functors from $\mathbf{C}^{(0)}$ to $\mathbf{C}^{(1)}$ and from $\mathbf{C}^{(1)}$ to $\mathbf{C}^{(2)}$ give rise to a hierarchy of terms: pure terms in $\mathbf{C}^{(0)}$, constructors in $\mathbf{C}^{(1)}$, modifiers in $\mathbf{C}^{(2)}$. This corresponds to the three translations of a typed lambda calculus into a monadic language [23].

On the comonads side, we get a dual hierarchy: pure terms in $\mathbf{C}^{(0)}$, observers in $\mathbf{C}^{(1)}$, modifiers in $\mathbf{C}^{(2)}$.

We instanciate these patterns with two fundamental examples of effects: state and exceptions.

Following [3], we consider that the states effect arise from the comonad

$A \times S$ (where S is the set of states), thus a decorated logic for states is built by extending the pattern for comonads. The comonad itself provides a decoration for the lookup operation, which observes the state, while the monad on its coKleisli category provides a decoration for the update operation.

Following [14], we consider that the exceptions effect arise from the monad $A + E$ (where E is the set of exceptions), thus a decorated logic for exceptions is built by extending the pattern for monads. The monad itself provides a decoration for the raising operation, which constructs an exception, while the comonad on its Kleisli category provides a decoration for the handling operation.

In fact the decorated logic for exceptions is not exactly dual to the decorated logic for states if we assume that the intended interpretation takes place in a distributive category, like the category of sets, which is not codistributive.

Other effects would lead to other additional rules, but we have chosen to focus on two effects which are well known from various points of view. Our goal is to enligthen the contributions of each approach: the annotation system from the types and effects systems [12], the major role of monads for some effects [14], and the dual role of comonads [22], as well as the flexibility of decorated logics [1]. Moreover, proofs in decorated logics can be checked with the Coq proof assistant; a library for states is available there: `http://coqeffects.forge.imag.fr`.

In this paper we focus on finite products and coproducts; from a programming point of view this means that we are considering languages with n-ary operations and with case distinction, but without loops or higher-order functions. In a language with effects there is a well-known issue with n-ary operations: their interpretation may depend on the order of evaluation of their arguments. In this paper we are looking for languages with case distinction and with *sequential products*, which allows to force the order of evaluation of the arguments, whenever this is required.

It is well known that (co)monads fit very well with composition but require additional assumptions for being fully compatible with products and coproducts. This corresponds to the fact that in the patterns from Section 2, which are valid for any (co)monad, the rules for products and coproducts hold only under some decoration constraints. However, such assumptions are satisfied for several (co)monads. This is in particular the case for the state comonad and the exceptions monad.

In Section 2 we describe the patterns for a comonad and for a monad. The first pattern is instanciated with the comonad for state in Section 3, and we prove the Hilbert-Post completeness of the decorated theory for state. In Section 4 we instanciate the second pattern to the monad for exceptions.

2 Patterns for comonads and for monads

2.1 Equational logic with conditionals

In this Section we define a grammar and an inference system for two logics

$\mathcal{L}_{com}$ and $\mathcal{L}_{mon}$, then we define an interpretation of these logics in a category with a comonad and a monad, respectively.

The logics $\mathcal{L}_{com}$ and $\mathcal{L}_{mon}$ are called *decorated* logics because their grammar and inference rules are essentially the grammar and inference rules for a "usual" logic, namely the equational logic with conditionals (denoted $\mathcal{L}_{eq}$), 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 [12].

Decorated logics are introduced in [1] in an abstract categorical framework, which will not be explicitly used in this paper.

The grammar of the equational logic with conditionals is reminded in Figure 1. Each term has a source type and a target type. As usual in categorical presentations of equational logic, a term has precisely one source type, which can be a product type or the unit type. Each equation relates two parallel terms, i.e., two terms with the same source and the same target. This grammar will be extended with decorations

in order to get the grammar of the logics $\mathcal{L}_{com}$ and $\mathcal{L}_{mon}$.

Grammar for the equational logic with conditionals:	
Types:	$t ::= A \mid B \mid \cdots \mid t + t \mid \mathbb{0} \mid t \times t \mid \mathbb{1}$
Terms:	$f ::= id_t \mid f \circ f \mid$
	$\langle f, f \rangle \mid pr_{t,t,1} \mid pr_{t,t,2} \mid \langle\ \rangle_t$
	$[f \mid f] \mid in_{t,t,1} \mid in_{t,t,2} \mid [\ ]_t \mid$
Equations:	$e ::= f \equiv f$

Figure 1: Equational logic with conditionals: grammar

2.2 Patterns

The rules in Figure 2 are *patterns*, in the following sense: when the boxes in the rules are removed, we get usual rules for the logic $\mathcal{L}_{eq}$, which may be interpreted in any bicartesian category. When the boxes are replaced by decorations, we get a logic which, according to the choice of decorations, may be interpreted in a bicartesian category with a comonad or a monad. There may be other ways to decorate the rules for $\mathcal{L}_{eq}$, but this is beyond the scope of this paper.

2.3 A decorated logic for a comonad

In the logic $\mathcal{L}_{com}$ for comonads, 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

congruence rules

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

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

categorical rules

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

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

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

product rules

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

(pair) $\dfrac{f_1^{\square} \colon A \to B_1 \quad f_2^{\square} \colon A \to B_2}{\langle f_1, f_2 \rangle^{\square} \colon A \to B_1 \times B_2}$

(pair-eq) $\dfrac{f_1^{\square} \colon A \to B_1 \quad f_2^{\square} \colon A \to B_2}{pr_1 \circ \langle f_1, f_2 \rangle \boxed{\equiv} f_1 \quad pr_2 \circ \langle f_1, f_2 \rangle \boxed{\equiv} f_2}$

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

(unit) $\dfrac{}{\mathbb{1}}$ (final) $\dfrac{A}{\langle\,\rangle_A^{\square} \colon A \to \mathbb{1}}$ (final-u) $\dfrac{f^{\square} \colon A \to \mathbb{1}}{f \boxed{\equiv} \langle\,\rangle_A}$

coproduct rules

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

(copair) $\dfrac{f_1^{\square} \colon A_1 \to B \quad f_2^{\square} \colon A_2 \to B}{[f_1 | f_2]^{\square} \colon A_1 + A_2 \to B}$

(copair-eq) $\dfrac{f_1^{\square} \colon A_1 \to B \quad f_2^{\square} \colon A_2 \to B}{[f_1 | f_2] \circ in_1 \boxed{\equiv} f_1 \quad [f_1 | f_2] \circ in_2 \boxed{\equiv} f_2}$

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

(empty) $\dfrac{}{\mathbb{0}}$ (initial) $\dfrac{B}{[\,]_B^{\square} \colon \mathbb{0} \to B}$ (initial-u) $\dfrac{g^{\square} \colon \mathbb{0} \to B}{g \boxed{\equiv} [\,]_B}$

Figure 2: Patterns: rules

an *accessor* (or an *observer*) 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*.

The inference rules of $\mathcal{L}_{com}$ are obtained by introducing some *conversion* rules and by decorating the rules in Figure 2.

When writing terms, if a decoration does not matter or if it is clear from the context, it may be omitted.

- The conversion rules are:

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

  The conversions for terms are *upcasting* conversions.

  We will always use them in a *safe* way, by interpreting them as injections. This allows to avoid any specific notation for these conversions; an accessor $a^{(1)}$ may be converted to a modifier which is denoted $a^{(2)}$: both have the same name although they are distinct terms; similarly, a pure term $v^{(0)}$ may be converted to $v^{(1)}$ or to $v^{(2)}$. An equation between terms with distinct decorations does not imply any downcasting of its members; for instance, if $f^{(2)} \cong g^{(0)}$ then it does not follow that f is downcasted to $f^{(0)}$. The conversions for equations mean that strong and weak equations coincide on pure terms and accessors and that each strong equation between modifiers can be seen as a weak one.

- 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 $\mathcal{L}_{com}$ which is isomorphic to $\mathcal{L}_{eq}$. Thus we get $id^{(0)}$, $pr^{(0)}$, $\langle\ \rangle^{(0)}$, $in^{(0)}$, $[\ ]^{(0)}$.

- The congruence rules for equations take all decorations for terms and for equations, with one notable exception: the replacement rule for weak equations holds only when the replaced term is pure:

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

- The categorical rules hold for all decorations and the decoration of a composed terms is the maximum of the decorations of its components.

- The product rules hold only when the given terms are pure or accessors and the decoration of a pair is the maximum of the decorations of its components. Thus, n-ary operations can be used only when their arguments are accessors.

- The coproduct rules hold only when the given terms are pure and a copair is always pure, which is the maximum of the decorations of its components. Thus, case distinction can be done only for pure terms.

2.4 The interpretation of $\mathcal{L}_{com}$ by a comonad

In order to give a meaning to the logic $\mathcal{L}_{com}$, let us consider a bicartesian category $\mathbf{C}$ with a comonad (T, ε, δ) satisfying the epi requirement, i.e., $\varepsilon_A : TA \to A$ is an epimorphism for each object A (the dual assumption is discussed in [14]).

Then we get a model $\mathbf{C}_T$ of the decorated logic $\mathcal{L}_{com}$ as follows.

- The types are interpreted as the objects of $\mathbf{C}$.
- The terms are interpreted as morphisms of $\mathbf{C}$: a pure term $f^{(0)} \colon A \to B$ as a morphism $f \colon A \to B$ in $\mathbf{C}$; an accessor $f^{(1)} \colon A \to B$ as a morphism $f \colon TA \to B$ in $\mathbf{C}$; and a modifier $f^{(2)} \colon A \to B$ as a morphism $f \colon TA \to TB$ in $\mathbf{C}$.
- The conversion from pure terms to accessors is interpreted by mapping $f \colon A \to B$ to $f \circ \varepsilon_A \colon TA \to B$. The epi requirement implies that this conversion is safe.
- The conversion from accessors to modifiers is interpreted by mapping $f \colon TA \to B$ to $Tf \circ \delta_A \colon TA \to TB$. It is easy to check that this conversion is safe.
- When a term f has several decorations (because it is pure or accessor, and thus can be upcasted) we will denote by f any one of its interpretations: a pure term $f^{(0)} \colon A \to B$ may be interpreted as $f \colon A \to B$ and as $f \colon TA \to B$ and as $f \colon TA \to TB$, and an accessor $f^{(1)} \colon A \to B$ as $f \colon TA \to B$ and as $f \colon TA \to TB$. The choice will be clear from the context, and when several choices are possible they will give the same result, up to conversions. For this reason, we will describe the interpretation of the rules only for the largest possible decorations.
- The identity $id_A^{(0)} \colon A \to A$ is interpreted as $id_A \colon A \to A$ in $\mathbf{C}$;
- The composition of two modifiers $f^{(2)} \colon A \to B$ and $g^{(2)} \colon B \to C$ is interpreted as $g \circ f \colon TA \to TB$ in $\mathbf{C}$.
- An equation between modifiers $f^{(2)} \cong g^{(2)} \colon A \to B$ is interpreted by an equality $f = g \colon TA \to TB$ in $\mathbf{C}$.
- A weak equation between modifiers $f^{(2)} \sim g^{(2)} \colon A \to B$ is interpreted by an equality $\varepsilon_B \circ f = \varepsilon_B \circ g \colon TA \to B$ in $\mathbf{C}$.
- The unit type is interpreted as the final object of $\mathbf{C}$ and the term $\langle\,\rangle_A^{(0)} \colon A \to \mathbb{1}$ as the unique morphism from A to $\mathbb{1}$ in $\mathbf{C}$.
- The product $B_1 \times B_2$ with its projections is interpreted as the binary product in $\mathbf{C}$ and the pair of $f_1^{(0)} \colon A \to B_1$ and $f_2^{(0)} \colon A \to B_2$ as the pair $\langle f_1, f_2 \rangle \colon A \to B_1 \times B_2$ in $\mathbf{C}$.

- The empty type is interpreted as the initial object of $\mathbf{C}$ and the term $[\,]_A^{(0)} \colon \mathbb{0} \to A$ as the unique morphism from $\mathbb{0}$ to A in $\mathbf{C}$.
- The coproduct $A_1 + A_2$ with its coprojections is interpreted as the binary coproduct in $\mathbf{C}$ and the copair of $f_1^{(1)} \colon A_1 \to B$ and $f_2^{(1)} \colon A_2 \to B$ as the copair $[f_1|f_2] \colon A_1 + A_2 \to TB$ in $\mathbf{C}$.

2.5 A decorated logic for a monad

The dual of the decorated logic $\mathcal{L}_{com}$ for a comonad is the decorated logic $\mathcal{L}_{mon}$ for a monad.

Thus, the grammar of $\mathcal{L}_{mon}$ is the same as the grammar of $\mathcal{L}_{com}$, but a term with decoration (1) is now called a *constructor*.

The rules for $\mathcal{L}_{mon}$ are nearly the same as the corresponding rules for $\mathcal{L}_{com}$, except that for weak equations the replacement rule always holds while the substitution rule holds only when the substituted term is pure:

$$\boxed{\text{(subs)} \quad \frac{f^{(0)} \colon A \to B \quad g_1^{(d)} \sim g_2^{(d')} \colon B \to C}{g_1 \circ f \sim g_2 \circ f}}$$

In the rules for pairs and copairs, the decorations are permuted.

The logic $\mathcal{L}_{mon}$ can be interpreted dually to $\mathcal{L}_{com}$. Let $\mathbf{C}$ be a bicartesian category and (M, η, μ) a monad on $\mathbf{C}$ satisfying the mono requirement, which means that $\eta_A : A \to MA$ is a monomorphism for each object A. Then we get a model $\mathbf{C}_M$ of the decorated logic $\mathcal{L}_{mon}$, where

a constructor $f^{(1)} \colon A \to B$ is interpreted as a morphism $f \colon A \to MB$ in $\mathbf{C}$

and a weak equation $f^{(2)} \sim g^{(2)} \colon A \to B$ is interpreted as an equality $f \circ \eta_A = g \circ \eta_A \colon A \to TB$ in $\mathbf{C}$.

3 States: an instance of the pattern for comonads

3.1 A decorated logic for state

Let us consider a distributive category $\mathbf{C}$ with epimorphic projections and with a distinguished object S called the *object of states*. We consider the comonad (T, ε, δ) with endofunctor $TA = A \times S$, with counit ε made of the projections $\varepsilon_A \colon A \times S \to A$, and with comultiplication δ which "duplicates" the states, in the sense that $\delta_A = \langle id_{A \times S} | pr_A \rangle \colon A \times S \to (A \times S) \times S$ where $pr_A \colon A \times S \to A$ is the projection.

We call this comonad the *comonad of state*. It is sometimes called the *product comonad*, and it is different from the *costate comonad* or *store comonad* with endofuntor $TA = S \times A^S$ [7].

The category $\mathbf{C}$ with the comonad of states provides a model of the logic $\mathcal{L}_{com}$. We can extend $\mathcal{L}_{com}$ into a logic $\mathcal{L}_{st}$ dedicated to the state comonad.

First, because of the specific choice of the comonad $TA = A \times S$, we can add new decorations to the rule patterns for pairs in $\mathcal{L}_{com}$, involving modifiers: there is a *left pair* $\langle f_1, f_2 \rangle_l^{(2)}$ of an accessor $f_1^{(1)}$ and a modifier $f_2^{(2)}$, satisfying the first three rules in Figure 3. There are also three rules (omitted), symmetric to these ones, for the *right pair* $\langle f_1, f_2 \rangle_r^{(2)}$ of a modifier $f_1^{(2)}$ and an accessor $f_2^{(1)}$.

The interpretation of the left pair $\langle f_1, f_2 \rangle_l^{(2)} : A \to B_1 \times B_2$ is the pair $\langle f_1, f_2 \rangle : A \times S \to B_1 \times B_2 \times S$ of $f_1 : A \times S \to B_1$ and $f_2 : A \times S \to B_2 \times S$.

Moreover, the rule (effect) expresses the fact that, when $TA = A \times S$, two modifiers coincide as soon as they return the same result and modify the state in the same way.

(l-pair)	$\dfrac{f_1^{(1)} \colon A \to B_1 \quad f_2^{(2)} \colon A \to B_2}{[f_1 \mid f_2]_l^{(2)} \colon A \to B_1 \times B_2}$
(l-pair-eq)	$\dfrac{f_1^{(1)} \colon A \to B_1 \quad f_2^{(2)} \colon A \to B_2}{pr_1^{(0)} \circ \langle f_1, f_2 \rangle_l^{(2)} \sim f_1^{(1)} \quad pr_2^{(0)} \circ \langle f_1, f_2 \rangle_l^{(2)} \cong f_2^{(2)}}$
(l-pair-u)	$\dfrac{g^{(2)} \colon A \to B_1 \times B_2 \quad f_1^{(1)} \colon A \to B_1 \quad f_2^{(2)} \colon A \to B_2 \quad pr_1^{(0)} \circ g \sim f_1 \quad pr_2^{(0)} \circ g \cong f_2}{g^{(2)} \cong \langle f_1, f_2 \rangle_l^{(2)}}$
(effect)	$\dfrac{f, g \colon A \to B \quad f \sim g \quad \langle\, \rangle_A \circ f \cong \langle\, \rangle_A \circ g}{f \cong g}$

Figure 3: $\mathcal{L}_{st}$: additional rules for products

For each set *Loc* of *locations* (or identifiers), additional grammar and rules for the logic $\mathcal{L}_{st}$ are given in Figure 4. We extend the grammar of $\mathcal{L}_{com}$ with a type V_X, an accessor $\texttt{lookup}_X^{(1)} : \mathbb{1} \to V_X$ and a modifier $\texttt{update}_X^{(2)} : V_T \to \mathbb{1}$ for each location X, and we also extend its rules.

The rule (local-global) asserts that two functions without result coincide as soon as they coincide when observed at each location. Together with the rule (effect) it implies that two functions coincide as soon as they return the same value and coincide on each location.

For each family of objects $(V_X)_{X \in Loc}$ in $\mathbf{C}$ such that $S \cong \prod_{X \in Loc} V_X$ we build a model $\mathbf{C}_{st}$ of $\mathcal{L}_{st}$, which extends the model the model $\mathbf{C}_T$ of $\mathcal{L}_{com}$ with functions for looking up and updating the locations.

The types V_X are interpreted as the objects V_X and the accessors $\texttt{lookup}_X^{(1)} : \mathbb{1} \to V_X$ as the projections from S to V_X. Then the interpretation of each modifier $\texttt{update}_X^{(2)} : V_X \to \mathbb{1}$ is the function from $V_X \times S$ to S defined as the tuple of the functions $f_{X,Y} : V_X \times S \to V_Y$ where $f_{X,X}$ is the projection from $V_X \times S$ to V_X and $f_{X,Y}$ is made of the projection from $V_X \times S$ to S followed by

Types:	$t ::= V_X$ for each $X \in Loc$
Terms:	$f ::= \texttt{lookup}_X \mid \texttt{update}_X$ for each $X \in Loc$
(lookup)	$\dfrac{X \in Loc}{\texttt{lookup}_X^{(1)} \colon \mathbb{1} \to V_X}$
(update)	$\dfrac{X \in Loc}{\texttt{update}_X^{(2)} \colon V_X \to \mathbb{1}}$
(lookupdate)	$\dfrac{X \in Loc}{\texttt{lookup}_X \circ \texttt{update}_X \sim id_{V_X}}$ $\quad \dfrac{X, Y \in Loc \;\; X \neq Y}{\texttt{lookup}_Y \circ \texttt{update}_X \sim \texttt{lookup}_Y \circ \langle\, \rangle_{V_X}}$
(local-global)	$\dfrac{f, g \colon A \to \mathbb{1} \quad \text{for all } X \in Loc\ \texttt{lookup}_X \circ f \sim \texttt{lookup}_X \circ g}{f \cong g}$

Figure 4: $\mathcal{L}_{st}$: additional grammar and rules for states

$\texttt{lookup}_Y : S \to V_Y$ when $Y \neq X$.

The logic we get, and its model, are essentially the same as in [2]: thus, the pattern for a comonad in Section 2 can be seen as a generalization to arbitrary comonads of the approach in [2].

Since we have assumed that the category $\mathbf{C}$ is *distributive* we get new decorations for the rule patterns for coproducts: the copair of two modifiers now exists, the corresponding decorated rules are given in Figure 5.

The interpretation of the modifier $[f_1|f_2]$, when both f_1 and f_2 are modifiers, is the composition of $[f_1|f_2] \colon (A_1 \times S) + (A_2 \times S) \to B \times S$ with the inverse of the canonical morphism $(A_1 \times S) + (A_2 \times S) \to (A_1 + A_2) \times S$: this inverse exists because $\mathbf{C}$ is distributive.

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

Figure 5: $\mathcal{L}_{st}$: additional rules for coproducts, when $\mathbf{C}$ is distributive

3.2 States: conditionals and binary operations

To conclude with states, let us look at the constructions for conditionals and binary operations in the language for states.

The rules in Figure 5 provide conditionals.

There is no binary product of modifiers, but there is a left product of a constructor and a modifier and a right product of a modifier and a constructor. It follows that the *left and right sequential products* of two modifiers f_1 and f_2 can be defined, as in [4], by composing, e.g., the left product of an identity and f_1 with the right product of f_2 and an identity.

A major feature of this approach is that, for states, sequential products are defined without any new ingredient: no kind of strength, in contrast with the approach using the strong monad of states $(A \times S)^S$ [14], no "external" decoration for equations, in contrast with [4]. This property is due to the introduction of the intermediate notion of *accessors* between pure terms (or *values*) and modifiers (or *computations*).

3.3 Hilbert-Post completeness

Now we use the decorated logic $\mathcal{L}_{st}$ for proving that the decorated theory for states is Hilbert-Post complete. This result is proved in [16, Prop.2.40] in the framework of Lawvere theories. Here we give a proof in the decorated logic for states. This proof has been checked in Coq[1].

The logic we use is the fragment $\mathcal{L}_{st,0}$ of $\mathcal{L}_{st}$ which involves neither products nor coproducts nor the empty type (but which involves the unit type). The *theory of state*, denoted $\mathcal{T}_{st}$, is the family of equations which may be derived from the axioms of $\mathcal{L}_{st,0}$ using the rules of $\mathcal{L}_{st,0}$. More generally, a *theory* $\mathcal{T}$ with respect to $\mathcal{L}_{st,0}$ is a family of equations between terms of $\mathcal{L}_{st,0}$ which is saturated with respect to the rules of $\mathcal{L}_{st,0}$. A theory $\mathcal{T}'$ is an *extension* of a theory $\mathcal{T}$ if it contains all the equations of $\mathcal{T}$. Two families of equations are called *equivalent* if each one can be derived from the other with the rules of $\mathcal{L}_{st,0}$.

As in [16, Prop.2.40], for the sake of simplicity it is assumed that there is a single location X, and we write V, `lookup` and `update` instead of V_X, $\texttt{lookup}_X$ and $\texttt{update}_X$. Then there is a single axiom $\texttt{lookup} \circ \texttt{update} \sim id_V$.

In addition, it is assumed that all types are *inhabited*, in the sense that for each type X there exists a closed pure term with type X.

Theorem 3.1. *Every equation between terms of $\mathcal{L}_{st,0}$ is equivalent to four equations between pure terms.*

Proof. The proof is obtained by merging the two parts of Proposition 3.2, which is proved in Appendix A. □

Proposition 3.2.

[1] Effect categories and COQ, http://coqeffects.forge.imag.fr

1. *Every equation between accessors is equivalent to two equations between pure terms.*

2. *Every equation between modifiers is equivalent to two equations between accessors.*

Roughly speaking, a theory (with respect to some logic) is said *syntactically complete* if no unprovable axiom can be added to the theory without introducing an inconsistency. More precisely, a theory with respect to the equational logic is *Hilbert-Post complete* if it is consistent and has no consistent proper extension [16, Definition 2.8.]. Since we use a decorated version of the equational logic, we have to define a decorated version of Hilbert-Post completeness.

Definition 3.3. With respect to the logic $\mathcal{L}_{st,0}$,

a theory $\mathcal{T}$ is *consistent* if there is an equation which is not in $\mathcal{T}$.

An extension $\mathcal{T}'$ of a theory $\mathcal{T}$ is a *pure extension* if it is generated by $\mathcal{T}$ and by equations between pure terms. It is a *proper extension* if it is not a pure extension.

A theory $\mathcal{T}$ is *Hilbert-Post complete* if it is consistent and has no consistent proper extension.

The proof of Theorem 3.4 relies on Theorem 3.1. We do not have to assume that the interpretation of the type V is a countable set. We have assumed that *Loc* is a singleton, but we conjecture that our result can be generalized to any set of locations, without any finiteness condition.

Theorem 3.4. *The theory for state is Hilbert-Post complete.*

Proof. The theory $\mathcal{T}_{st}$ is consistent: it cannot be proved that $\mathtt{update}^{(2)} \cong \langle\ \rangle_V^{(0)}$.

Let us consider an extension $\mathcal{T}$ of $\mathcal{T}_{st}$ and let $\mathcal{T}_{(0)}$ be the theory generated by $\mathcal{T}_{st}$ and by the equations between pure terms in $\mathcal{T}$. Thus, $\mathcal{T}_{(0)}$ is a pure extension of $\mathcal{T}_{st}$ and $\mathcal{T}$ is an extension of $\mathcal{T}_{(0)}$. Let us consider an arbitrary equation e in $\mathcal{T}$, according to Theorem 3.1 we get a family E of equations between pure terms which is equivalent to the given equation e.

Since e is in $\mathcal{T}$ and $\mathcal{T}$ is saturated, the equations in E are also in $\mathcal{T}$, hence they are in $\mathcal{T}_{(0)}$.

Since E is in $\mathcal{T}_{(0)}$ and $\mathcal{T}_{(0)}$ is saturated, the equation e is also in $\mathcal{T}_{(0)}$.

This proves that $\mathcal{T}_{(0)} = \mathcal{T}$, so that the theory $\mathcal{T}_{st}$ has no proper extension. □

4 Exceptions: an instance of the pattern for monads

4.1 The core language for exceptions

Let us consider a bicartesian category $\mathbf{C}$ with monomorphic coprojections and with a distinguished object E called the *object of exceptions*. We do not

assume that $\mathbf{C}$ is distributive (it would not help) nor codistributive, because usually this is not the case. The *monad of exceptions* on $\mathbf{C}$ is the monad (M, η, μ) with endofunctor $MA = A + E$, its unit η is made of the coprojections $\eta_A \colon A \to A + E$, and its multiplication μ "merges" the exceptions, in the sense that $\mu_A = [id_{A+E} | in_A] \colon (A + E) + E \to A + E$ where $in_A \colon E \to A + E$ is the coprojection. It satisfies the mono requirement because the coprojections are monomorphisms. Thus, the category $\mathbf{C}$ with the monad of exceptions provides a model of the logic $\mathcal{L}_{mon}$. The name of the decorations is 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 $MA = A + E$, it is possible to extend the logic $\mathcal{L}_{mon}$ as $\mathcal{L}_{exc}$, called the *logic for exceptions*, so that $\mathbf{C}$ with $MA = A + E$ can be extended as a model $\mathbf{C}_{exc}$ of $\mathcal{L}_{exc}$.

First, dually to the left and right pairs for states in Figure 3, we get new decorations to the rule patterns for copairs in $\mathcal{L}_{mon}$, involving modifiers, as in Figure 6 for the left copairs (the rules for the right copairs are omitted).

The interpretation of the left copair $[f_1|f_2]_l^{(2)} : A_1 + A_2 \to B$ is the copair $[f_1|f_2] : A_1 + A_2 + E \to B + E$ of $f_1 : A_1 \to B + E$ and $f_2 : A_2 + E \to B + E$ in $\mathbf{C}$.

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 $[f_1|f_2]_l^{(2)} : A \to B$ of any constructor $f_1^{(1)} \colon A \to B$ with any modifier $f_2^{(2)} \colon \mathbb{0} \to B$, which is characterized up to strong equations by $[f_1|f_2]_l \sim f_1$ and $[f_1|f_2]_l \cong f_2$. This will be used in the construction of the `try`/`catch` expressions.

Moreover, the rule (effect) expresses the fact that, when $MA = A + E$, two modifiers coincide as soon as they coincide on ordinary values and on exceptions.

(l-copair)	$\dfrac{f_1^{(1)} \colon A_1 \to B \quad f_2^{(2)} \colon A_2 \to B}{[f_1\|f_2]_l^{(2)} \colon A_1 + A_2 \to B}$
(l-copair-eq)	$\dfrac{f_1^{(1)} \colon A_1 \to B \quad f_2^{(2)} \colon A_2 \to B}{[f_1\|f_2]_l^{(2)} \circ in_1^{(0)} \sim f_1^{(1)} \quad [f_1\|f_2]_l^{(2)} \circ in_2^{(0)} \cong f_2^{(2)}}$
(l-copair-u)	$\dfrac{g^{(2)} \colon A_1 + A_2 \to B \quad f_1^{(1)} \colon A_1 \to B \quad f_2^{(2)} \colon A_2 \to B \quad g \circ in_1 \sim f_1 \quad g \circ in_2 \cong f_2}{g^{(2)} \cong [f_1\|f_2]_l^{(2)}}$
(effect)	$\dfrac{f, g \colon A \to B \quad f \sim g \quad f \circ [\ ]_A \cong g \circ [\ ]_A}{f \cong g}$

Figure 6: $\mathcal{L}_{exc}$: additional rules for coproducts

For each set *Exn* of *exception names*, additional grammar and rules for the logic $\mathcal{L}_{exc}$ are given in Figure 7. We extend the grammar of $\mathcal{L}_{mon}$ 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 [5].

The rule (local-global) asserts that two functions without argument coincide as soon as they coincide on each exception. Together with the rule (effect) it implies that two functions coincide as soon as they coincide on their argument and on each exception.

Types:	$t ::= V_T$ for each $T \in Exn$
Terms:	$f ::= \mathtt{tag}_T \mid \mathtt{untag}_T$ for each $T \in Exn$
(tag)	$\dfrac{T \in Exn}{\mathtt{tag}_T^{(1)} : V_T \to \mathbb{0}}$
(untag)	$\dfrac{T \in Exn}{\mathtt{untag}_T^{(2)} : \mathbb{0} \to V_T}$
(untag-tag)	$\dfrac{T \in Exn}{\mathtt{untag}_T \circ \mathtt{tag}_T \sim id_{V_T}}$ $\quad \dfrac{T, R \in Exn \; T \neq R}{\mathtt{untag}_T \circ \mathtt{tag}_R \sim [\,]_{V_T} \circ \mathtt{tag}_R}$
(local-global)	$\dfrac{f, g : \mathbb{0} \to B \quad \text{for all } T \in Exn \; f \circ \mathtt{tag}_T \sim g \circ \mathtt{tag}_T}{f \cong g}$

Figure 7: $\mathcal{L}_{exc}$: additional grammar and rules for exceptions

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 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:

Since we did not assume that the category $\mathbf{C}$ is codistributive we cannot get products of modifiers in a way dual to the coproducts of modifiers for states.

However these rules have not been used for proving the Hilbert-Post completeness of the theory for state. Thus by duality from Theorem 3.4 we get "for free" a result about the core language for exceptions.

Corollary 4.1. *The core theory for exceptions is Hilbert-Post complete.*

4.2 The programmer's language for exceptions

We have obtained a logic $\mathcal{L}_{exc}$ for exceptions, with the core operations for tagging and untagging. This logic provides a direct access to catchers (the untagging functions), which is not provided by the usual mechanism of exceptions in programming languages. In fact the core operations remain *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 has no catcher: the only way to catch an exception is by using a `try/catch` expression, which itself propagates exceptions. Thus, all terms of the programmer's language are propagators. 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`. 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 [5].

The main ingredients for building the programmer's language from the core language are the coproducts $A \cong A + \mathbb{0}$ and a new conversion rule for terms. The *downcast* conversion of a catcher to a propagator could have been defined in Section 2 for the logic $\mathcal{L}_{com}$, and dually for the logic $\mathcal{L}_{mon}$; the rule is:

$$\frac{f^{(2)} \colon A \to B}{(\downarrow f)^{(1)} \colon A \to B}$$

This downcasting conversion from catchers to propagators is interpreted by mapping $f \colon MA \to MB$ to $\downarrow f = f \circ \eta_A \colon A \to MB$. It is related to weak equations: $f \sim \downarrow f$, and $f \sim g$ if and only if $\downarrow f \cong \downarrow g$. But the downcasting conversion is *unsafe*: several catchers may be downcasted to the same propagator. This powerful operation turns an effectful term to an effect-free one; since it is not required for states nor for the core language for exceptions, we did not introduce it earlier.

Definition 4.2. For each type B and each exception name T, the propagator $\mathtt{throw}^{(1)}_{B,T}$ is:

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

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

steps, involving two catchers $\texttt{catch}(T \Rightarrow g)^{(2)}$ and $\texttt{TRY}(f)\texttt{catch}(T \Rightarrow g)^{(2)}$, as follows:

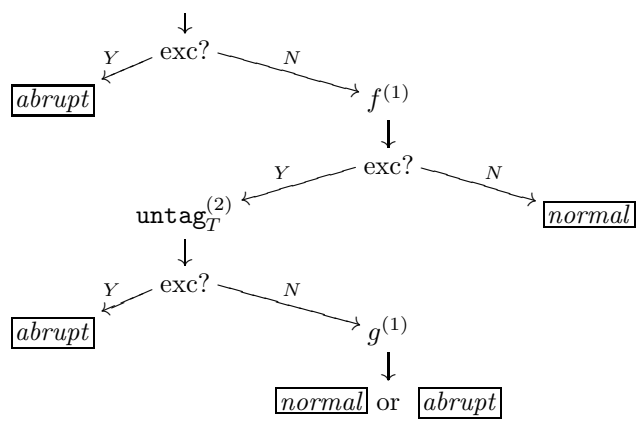
$$\begin{aligned}
&\texttt{catch}(T \Rightarrow g)^{(2)} = [\ g^{(1)} \mid [\ ]_B^{(0)}\ ]^{(1)} \circ \texttt{untag}_T^{(2)} \colon \mathbb{0} \to B \\
&\texttt{TRY}(f)\texttt{catch}(T \Rightarrow g)^{(2)} = [\ id_B \mid \texttt{catch}(T \Rightarrow g)\ ]_l^{(2)} \circ f^{(1)} \colon A \to B \\
&\texttt{try}(f)\texttt{catch}(T \Rightarrow g)^{(1)} = \downarrow(\texttt{TRY}(f)\texttt{catch}(T \Rightarrow g)) \colon A \to B
\end{aligned}$$

This means that raising an exception with name T consists in tagging the given ordinary value (in V_T) as an exception and coerce it to any given type B.

For handling an exception, the intermediate expressions $\texttt{catch}(T \Rightarrow g)$ and $\texttt{TRY}(f)\texttt{catch}(T \Rightarrow g)$ are private catchers and the expression $\texttt{try}(f)\texttt{catch}(T \Rightarrow g)$ is a public propagator: the downcast operator prevents it from catching exceptions with name T which might have been raised before the $\texttt{try}(f)\texttt{catch}(T \Rightarrow g)$ expression is considered.

The definition of $\texttt{try}(f)\texttt{catch}(T \Rightarrow g)$ corresponds to the Java mechanims for exceptions [8, 9].

The definition of $\texttt{try}(f)\texttt{catch}(T \Rightarrow g)$ corresponds to the following control flow, where `exc?` means "*is this value an exception?*", an *abrupt* termination returns an uncaught exception and a *normal* termination returns an ordinary value; this corresponds, for instance, to the Java mechanims for exceptions [8, 9].



4.3 Exceptions: case distinction and binary operations

To conclude with exceptions, let us look at the constructions for case distinction and binary operations in the programmer's language for exceptions, which means, copairs and pairs of *constructors*.

The general rules of the logic $\mathcal{L}_{mon}$ include coproducts of constructors (Figure 2), which provide case distinction for all terms in the programmer's language for exceptions.

But the general rules for a monad do not include binary products involving a constructor, hence they cannot be used for dealing with binary operations in the programmer's language for exceptions when at least an argument is not pure. Indeed, if $f_1^{(0)} \colon A \to B_1$ is pure and $f_2^{(1)} \colon A \to B_2$ does raise an exception, it is in general impossible to find $f^{(1)} \colon A \to B_1 \times B_2$ such that $pr_1 \circ f \cong f_1$ and $pr_2 \circ f \cong f_2$.

However, there are several ways to formalize the fact of first evaluating f_1 then f_2: for instance by using a strong monad [14], or a sequential product [4], or productors [21]. The sequential product approach can be used in our framework; it requires the introduction of a third kind of "equations", in addition to the strong and weak equations, which corresponds to the usual order between partial functions: details are provided in [4].

5 Conclusion

We have presented two patterns giving sound inference systems for effects arising from a monad or a comonad.

We also gave detailed examples of applications of these patterns to the state and the exceptions effects. The obtained decorated proof system for states has been implemented in Coq, so that the given proofs can be automatically verified. We plan to adapt this logic to local states (with allocation) in order to provide a decorated proof of the completeness Theorem in [20].

From this implementation, we plan to extract the generic part corresponding to the comonad pattern, dualize it and extend it to handle the programmer's language for exceptions.

Then a major issue is scalability: how can we combine effects? Within the framework of this paper, it may seem difficult to guess how several effects arising from either monads or comonads can be combined. However, as mentioned in the Introduction, this paper deals with two patterns for instanciating the more general framework of decorated logics [1]. Decorated logics are based on spans in a relevant category of logics, so that the combination of effects can be based on the well-known composition of spans.

Acknowledgment. We are grateful to Samuel Mimram for enlightning discussions.

References

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

[2] 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).

[3] 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).

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

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

[6] Jean-Guillaume Dumas, Dominique Duval, Burak Ekici, Damien Pous. Formal verification in Coq of program properties involving the global state effect. arXiv:1310.0794 (2013).

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

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

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

[10] Bart Jacobs and Jan Rutten. An introduction to (co)algebras and (co)induction. In: D. Sangiorgi and J. Rutten (eds), Advanced topics in bisimulation and coinduction, p.38-99, 2011.

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

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

[13] Paul-André Melliès. Segal Condition Meets Computational Effects. LICS 2010. p. 150-159, IEEE Computer Society (2010).

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

[15] Eugenio Moggi and Sonia Fagorzi. A Monadic Multi-stage Metalanguage. FoSSaCS 2003, LNCS, Vol. 2620, p. 358-374, Springer (2003).

[16] Matija Pretnar. The Logic and Handling of Algebraic Effects. PhD. University of Edinburgh 2010.

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

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

[19] Lutz Schröder, Till Mossakowski. Generic Exception Handling and the Java Monad. AMAST 2004. LNCS, Vol. 3116, p. 443-459, Springer (2004).

[20] Sam Staton. Completeness for Algebraic Theories of Local State. FoSSaCS 2010. LNCS, Vol. 6014, p. 48-63, Springer (2010).

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

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

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

A Proof of Hilbert Post completeness

The logic used in this Appendix is the fragment $\mathcal{L}_{st,0}$ of the decorated logic for states $\mathcal{L}_{st}$ which involves neither products nor coproducts nor the empty type, but which involves the unit type.

For the sake of simplicity it is assumed that there is a single location X, and we write V, $\mathtt{lkp}$ and $\mathtt{upd}$ instead of V_X, $\mathtt{lookup}_X$ and $\mathtt{update}_X$. Then there is a single axiom $\mathtt{lkp} \circ \mathtt{upd} \sim id_V$.

In Section 3, the proof of Hilbert-Post completeness in Theorem 3.4 relies on Proposition 3.2, which is restated here as Proposition A.5. The aim of this Appendix is to prove Proposition A.5.

Lemma A.1. *The following rules can be derived:*

1. $$\frac{f^{(2)}, g^{(2)} : X \to \mathbb{1} \quad \mathtt{lkp} \circ f \sim \mathtt{lkp} \circ g}{f \cong g}$$

2. $$\frac{f^{(2)}, g^{(2)} : X \to V \quad f \sim g}{\mathtt{upd} \circ f \cong \mathtt{upd} \circ g}$$

3. $$\frac{}{\mathtt{upd} \circ \mathtt{lkp} \cong id_{\mathbb{1}}}$$

4. $$\frac{a^{(1)} : X \to V \quad u^{(0)} : V \to Y}{u^{(0)} \circ \mathtt{lkp}^{(1)} \circ \mathtt{upd}^{(2)} \circ a^{(1)} \sim u^{(0)} \circ a^{(1)}}$$

5. $$\frac{x^{(0)} : \mathbb{1} \to X}{x^{(0)} \cong x^{(0)} \circ \langle\, \rangle_V^{(0)} \circ \mathtt{lkp}^{(1)}}$$

6. $$\frac{u^{(0)}, w^{(0)} : V \to X \quad w^{(0)} \circ \mathtt{lkp}^{(1)} \cong u^{(0)} \circ \mathtt{lkp}^{(1)}}{w^{(0)} \cong u^{(0)}}$$

7. $$\frac{x^{(0)} : \mathbb{1} \to X \quad w^{(0)} : V \to X \quad w^{(0)} \circ \mathtt{lkp}^{(1)} \cong x^{(0)}}{w^{(0)} \cong x^{(0)} \circ \langle\,\rangle_V^{(0)}}$$

Proof. 1. Consequence of the observational Rule (local-global) with only one location.

2. Consequence of 1 applied to $\mathtt{upd} \circ f, \mathtt{upd} \circ g : X \to \mathbb{1}$: indeed, from the axiom $\mathtt{lkp} \circ \mathtt{upd} \sim id_V$ we get $\mathtt{lkp} \circ \mathtt{upd} \circ f \sim \mathtt{lkp} \circ \mathtt{upd} \circ g$.

3. From axiom $\mathtt{lkp} \circ \mathtt{upd} \sim id_V$ by substitution we get $\mathtt{lkp} \circ \mathtt{upd} \circ \mathtt{lkp} \sim \mathtt{lkp}$; thus, point 1 implies $\mathtt{upd} \circ \mathtt{lkp} \cong id_{\mathbb{1}}$.

4. From $\mathtt{lkp}^{(1)} \circ \mathtt{upd}^{(2)} \sim id_V$, as u is pure, by the weak replacement we have $u^{(0)} \circ \mathtt{lkp}^{(1)} \circ \mathtt{upd}^{(2)} \sim u^{(0)}$. Then, weak substitution with a yields $u^{(0)} \circ \mathtt{lkp}^{(1)} \circ \mathtt{upd}^{(2)} \circ a^{(1)} \sim u^{(0)} \circ a^{(1)}$.

5. We know that $\langle\,\rangle_V^{(0)} \circ \mathtt{lkp}^{(1)} : \mathbb{1} \to \mathbb{1} \cong id_{\mathbb{1}}$.

   It follows that $x^{(0)} \circ \langle\,\rangle_V^{(0)} \circ \mathtt{lkp}^{(1)} \cong x^{(0)}$.

6. Let $w^{(0)} \circ \mathtt{lkp}^{(1)} \cong u^{(0)} \circ \mathtt{lkp}^{(1)}$. Composing with $\mathtt{upd}$ we get $w^{(0)} \circ \mathtt{lkp}^{(1)} \circ \mathtt{upd}^{(2)} \cong u^{(0)} \circ \mathtt{lkp}^{(1)} \circ \mathtt{upd}^{(2)}$. Using the axiom $\mathtt{lkp} \circ \mathtt{upd} \sim id_V$ and the replacement rule for $\sim$, which can be used here because both w and u are pure, we get $w^{(0)} \sim u^{(0)}$. Since weak and strong equations coincide on pure terms we get $w^{(0)} \cong u^{(0)}$.

7. Let $w^{(0)} \circ \mathtt{lkp}^{(1)} \cong x^{(0)}$. By point 5 above we get $x^{(0)} \cong x^{(0)} \circ \langle\,\rangle_V^{(0)} \circ \mathtt{lkp}^{(1)}$, thus $w^{(0)} \circ \mathtt{lkp}^{(1)} \cong x^{(0)} \circ \langle\,\rangle_V^{(0)} \circ \mathtt{lkp}^{(1)}$. Then by point 6 above we get $w^{(0)} \cong x^{(0)} \circ \langle\,\rangle_V^{(0)}$.

□

Now, let us prove Proposition A.2, which says that, up to strong equations, it can be assumed that there is at most one occurrence of $\mathtt{lkp}$ in any accessor and at most one occurrence of $\mathtt{upd}$ in any modifier.

Proposition A.2. *1. For each accessor $a^{(1)} : X \to Y$, if a is not pure then there is a pure term $v^{(0)} : V \to Y$ such that*

$$a^{(1)} \cong v^{(0)} \circ \mathtt{lkp}^{(1)} \circ \langle\,\rangle_X^{(0)} \tag{1}$$

2. For each modifier $f^{(2)} : X \to Y$, if f is not an accessor then there is an accessor $a^{(1)} : X \to V$ and a pure term $u^{(0)} : V \to Y$ such that

$$f^{(2)} \cong u^{(0)} \circ \mathtt{lkp}^{(1)} \circ \mathtt{upd}^{(2)} \circ a^{(1)} \tag{2}$$

Proof. 1. If $a^{(1)} : X \to Y$ is not pure then it contains at least one occurrence of $\mathtt{lkp}^{(1)}$. Thus, it can be written in a unique way as $a^{(1)} = v^{(0)} \circ \mathtt{lkp}^{(1)} \circ a_1^{(1)}$ for some pure term $v^{(0)} : V \to Y$ and some accessor $a_1^{(1)} : X \to \mathbb{1}$. Since $a_1^{(1)} : X \to \mathbb{1}$ is such that $a_1^{(1)} \cong \langle\ \rangle_X$, the result follows.

2. If $f^{(2)} : X \to Y$ is not an accessor then it contains at least one occurrence of $\mathtt{upd}^{(2)}$. Thus, it can be written in a unique way as $f^{(2)} = b^{(1)} \circ \mathtt{upd}^{(2)} \circ f_1^{(2)}$ for some accessor $b^{(1)} : \mathbb{1} \to Y$ and some modifier $f_1^{(2)} : X \to V$. From point 1, we also have that $b^{(1)} \cong v^{(0)} \circ \mathtt{lkp}^{(1)} \circ \langle\ \rangle_{\mathbb{1}} \cong v^{(0)} \circ \mathtt{lkp}^{(1)}$ for some pure term $v^{(0)} : V \to Y$ so that $f^{(2)} \cong v^{(0)} \circ \mathtt{lkp}^{(1)} \circ \mathtt{upd}^{(2)} \circ f_1^{(2)}$.

   - If f_1 is an accessor, the result follows with $a = f_1$.
   - Otherwise, $f_1^{(2)}$ contains at least one occurrence of $\mathtt{upd}^{(2)}$. Thus, it can be written in a unique way as $f_1^{(2)} = b_1^{(1)} \circ \mathtt{upd}^{(2)} \circ f_2^{(2)}$ for some accessor $b_1^{(1)} : \mathbb{1} \to V$ and some modifier $f_2^{(2)} : X \to V$. According to point 1 applied to the accessor b_1, either b_1 is pure or $b_1^{(1)} \cong v_1^{(0)} \circ \mathtt{lkp}^{(1)}$ for some pure term $v_1^{(0)} : V \to V$
     - If $b_1^{(1)} \cong v_1^{(0)} \circ \mathtt{lkp}^{(1)}$ then $f_1 \cong v_1 \circ \mathtt{lkp} \circ \mathtt{upd} \circ f_2$. The axiom $\mathtt{lkp} \circ \mathtt{upd} \sim id_V$ and the replacement and substitution rules for $\sim$ (since v_1 is pure) yield $f_1 \sim v_1 \circ f_2$. Then it follows from point 2 in Lemma A.1 that $\mathtt{upd} \circ f_1 \cong \mathtt{upd} \circ v_1 \circ f_2$, and since $f = b \circ \mathtt{upd} \circ f_1$ we get $f \cong b \circ \mathtt{upd} \circ v_1 \circ f_2$. The result follows by induction on the number of occurrences of $\mathtt{upd}$ in f: indeed, there is one less occurrence of $\mathtt{upd}$ in $b \circ \mathtt{upd} \circ v_1 \circ f_2$ than in $f = b \circ \mathtt{upd} \circ b_1 \circ \mathtt{upd} \circ f_2$.
     - If b_1 is pure then $b_1^{(0)} \cong b_1^{(0)} \circ \langle\ \rangle_V \circ \mathtt{lkp}$ from point 5 in Lemma A.1. Thus the previous proof applies by replacing b_1 with $b_1 \circ \langle\ \rangle_V$.

□

Corollary A.3. *The previous forms can be simplified for accessors with domain $\mathbb{1}$ and for modifiers with codomain $\mathbb{1}$, as follows:*

1. *For each accessor $a^{(1)} : \mathbb{1} \to Y$ there is a pure term $v^{(0)} : V \to Y$ such that*
$$a^{(1)} \cong v^{(0)} \circ \mathtt{lkp}^{(1)}$$

2. *For each modifier $f^{(2)} : X \to \mathbb{1}$ there is an accessor $a^{(1)} : X \to V$ such that*
$$f^{(2)} \cong \mathtt{upd}^{(2)} \circ a^{(1)}$$

Proof. 1.
   - If $a : \mathbb{1} \to Y$ is pure, since $\langle\ \rangle_V \circ \mathit{lookup} \cong id_{\mathbb{1}}$ (because $\langle\ \rangle_V \circ \mathit{lookup}$ is an accessor) we get $a \cong a \circ \langle\ \rangle_V \circ \mathtt{lkp}$, thus the result is obtained with $v^{(0)} = a \circ \langle\ \rangle_V$.
   - Otherwise, we have just proved that $a \cong v^{(0)} \circ \mathtt{lkp} \circ \langle\ \rangle_X^{(0)}$ with $X = \mathbb{1}$, then $\langle\ \rangle_X \cong id_{\mathbb{1}}$ and $a^{(1)} \cong v^{(0)} \circ \mathtt{lkp}$.

2.
   - If $f : X \to \mathbb{1}$ is an accessor, since $\mathtt{upd} \circ \mathtt{lkp} \cong id_{\mathbb{1}}$ we get $f \cong \mathtt{upd} \circ \mathtt{lkp} \circ f$, thus the result is obtained with $a^{(1)} = \mathtt{lkp} \circ f$.
   - Otherwise, we have just proved that $f \cong b^{(1)} \circ \mathtt{upd} \circ a^{(1)}$ with $b^{(1)} : \mathbb{1} \to \mathbb{1}$, then $b \cong id_{\mathbb{1}}$ and $f^{(2)} \cong \mathtt{upd} \circ a^{(1)}$.

□

Corollary A.4. *For each modifier $f^{(2)} : X \to Y$, if f is not an accessor then there is an accessor $a^{(1)} : X \to V$ and a pure term $u^{(0)} : V \to Y$ such that $f \sim u^{(0)} \circ a^{(1)}$.*

Proof. From Proposition A.2 we have that $f^{(2)} \cong u^{(0)} \circ \mathtt{lkp}^{(1)} \circ \mathtt{upd}^{(2)} \circ a^{(1)}$. Using the axiom $\mathtt{lkp} \circ \mathtt{upd} \sim id_V$ and the replacement rule for $\sim$, which can be used here because $u^{(0)}$ is pure, we get $f^{(2)} \sim u^{(0)} \circ a^{(1)}$. □

We can now prove Proposition A.5 on which the Hilbert-Post completeness theorem relies. This proof has been checked with the Coq proof assistant using the system for states of [6]. The Coq library with the inference system is available there: http://coqeffects.forge.imag.fr. The single proof of the following proposition (roughly 16 pages in Coq) is directly available there: http://coqeffects.forge.imag.fr/HPcompleteCoq.v.

Proposition A.5. *Let us assume that for each type X there exists a closed pure term $h_X^{(0)} : \mathbb{1} \to X$. Then:*

1. *every equation between accessors is equivalent to one or two equations between pure terms;*
2. *every equation between modifiers is equivalent to one or two equations between accessors.*

Proof. 1. We prove that for any accessors $a_1^{(1)}, a_2^{(1)} : X \to Y$ there are three cases:

   (a) either they are both pure and $a_1 \cong a_2$ is the required equation between pure terms.

   (b) either they are both accessors and it can be derived from $a_1 \cong a_2$ that $v_1 \cong v_2$ for some pure terms $v_1^{(0)}, v_2^{(0)} : V \to Y$.

   (c) or one of them is pure and the other one is an accessor and it can be derived from $a_1 \cong a_2$ that $v_1 \cong v_2$ and $w_1 \cong w_2$ for some pure terms $v_1^{(0)}, v_2^{(0)} : V \to Y$ and $w_1^{(0)}, w_2^{(0)} : X \to Y$.

   We prove, moreover, that the converse also hold.

   (a) As already mentioned, if a_1 and a_2 are both pure and $a_1 \cong a_2$ is the required equation between pure terms.

   (b) If neither a_1 nor a_2 is pure, then according to Proposition A.2 $a_1^{(1)} \cong v_1^{(0)} \circ \mathtt{lkp} \circ \langle\ \rangle_X^{(0)}$ and $a_2^{(1)} \cong v_2^{(0)} \circ \mathtt{lkp} \circ \langle\ \rangle_X^{(0)}$ for some pure terms $v_1^{(0)}, v_2^{(0)} : V \to Y$.

- Starting from the equation $a_1^{(1)} \cong a_2^{(1)} : X \to V$ we thus get $v_1^{(0)} \circ \mathtt{lkp} \circ \langle\ \rangle_X \cong v_2^{(0)} \circ \mathtt{lkp} \circ \langle\ \rangle_X : X \to Y$. Then, using the assumption, for any function $h_X^{(0)} : \mathbb{1} \to X$, we have that $v_1^{(0)} \circ \mathtt{lkp} \circ \langle\ \rangle_X \circ h_X \circ \mathtt{upd} \cong v_2^{(0)} \circ \mathtt{lkp} \circ \langle\ \rangle_X \circ h_X \circ \mathtt{upd}$. Now $\langle\ \rangle_X \circ h_X^{(0)} \cong id_{\mathbb{1}} : \mathbb{1} \to \mathbb{1}$. This, together with the axiom $\mathtt{lkp} \circ \mathtt{upd} \sim id_V$ and the replacement rule for $\sim$ (which can be used here because both v_1 and v_2 are pure) yield $v_1^{(0)} \sim v_2^{(0)}$. As the latter are both pure terms we also have $v_1^{(0)} \cong v_2^{(0)} : V \to Y$.
- Conversely, if $v_1^{(0)} \cong v_2^{(0)} : V \to Y$ then $v_1^{(0)} \circ \mathtt{lkp}^{(1)} \circ \langle\ \rangle_X^{(0)} \cong v_2^{(0)} \circ \mathtt{lkp}^{(1)} \circ \langle\ \rangle_X : X \to Y$, which means that $a_1^{(1)} \cong a_2^{(1)} : X \to Y$.

(c) The only remaining case is w.l.o.g. if a_1 is pure and a_2 is not.

- Then $a_2^{(1)} = v_2^{(0)} \circ \mathtt{lkp}^{(1)} \circ \langle\ \rangle_X^{(0)}$ from Proposition A.2 as previously and $v_1^{(0)} = a_1^{(0)} \circ h_X^{(0)} \circ \langle\ \rangle_V : V \to Y$ satisfies $v_1^{(0)} \cong v_2^{(0)}$ for any assumed $h_X^{(0)} : \mathbb{1} \to X$. Indeed from $a_1^{(0)} \cong v_2^{(0)} \circ \mathtt{lkp}^{(1)} \circ \langle\ \rangle_X^{(0)}$ we get
$$a_1^{(0)} \circ h_X^{(0)} \cong v_2^{(0)} \circ \mathtt{lkp}^{(1)} \circ \langle\ \rangle_X^{(0)} \circ h_X^{(0)}. \tag{3}$$
But, on the one hand, $a_1^{(0)} \circ h_X^{(0)} : \mathbb{1} \to Y$ so that point 5 in Lemma A.1 gives $a_1^{(0)} \circ h_X^{(0)} \cong v_1^{(0)} \circ \mathtt{lkp}^{(1)}$ with $v_1^{(0)} = a_1^{(0)} \circ h_X^{(0)} \circ \langle\ \rangle_V^{(0)} : V \to Y$. On the other hand, $\langle\ \rangle_X^{(0)} \circ h_X^{(0)} \cong id_{\mathbb{1}}^{(0)}$ so that $v_2^{(0)} \circ \mathtt{lkp}^{(1)} \circ \langle\ \rangle_X^{(0)} \circ h_X^{(0)} \cong v_2^{(0)} \circ \mathtt{lkp}^{(1)}$. Thus Equation (3) rewrites as $v_1^{(0)} \circ \mathtt{lkp}^{(1)} \cong v_2^{(0)} \circ \mathtt{lkp}^{(1)}$ and point 7 in Lemma A.1 yields
$$a_1^{(0)} \circ h_X^{(0)} \circ \langle\ \rangle_V^{(0)} = v_1^{(0)} \cong v_2^{(0)} : V \to Y. \tag{4}$$
Thus now we also have $a_2^{(1)} \cong v_2^{(0)} \circ \mathtt{lkp}^{(1)} \circ \langle\ \rangle_X^{(0)} \cong a_1^{(0)} \circ h_X^{(0)} \circ \langle\ \rangle_V^{(0)} \circ \mathtt{lkp}^{(1)} \circ \langle\ \rangle_X^{(0)} \cong a_1^{(0)} \circ h_X^{(0)} \circ \langle\ \rangle_X^{(0)}$. From the original equation $a_1^{(0)} \cong a_2^{(1)}$ we finally get
$$a_1^{(0)} \circ h_X^{(0)} \circ \langle\ \rangle_X^{(0)} \cong a_1^{(0)} : X \to Y. \tag{5}$$
- Conversely, we start from $v_2^{(0)}$ and $a_1^{(0)}$ satisfying both Equations (4) and (5). Then, we define $a_2^{(1)} = v_2^{(0)} \circ \mathtt{lkp}^{(1)} \circ \langle\ \rangle_X^{(0)}$ which satisfies $a_2^{(1)} \cong a_1^{(0)} \circ h_X^{(0)} \circ \langle\ \rangle_V^{(0)} \circ \mathtt{lkp}^{(1)} \circ \langle\ \rangle_X^{(0)}$ thanks to Equation (4). The latter is also $a_2^{(1)} \cong a_1^{(0)} \circ h_X^{(0)} \circ \langle\ \rangle_X^{(0)}$ which is thus $a_2^{(1)} \cong a_1^{(0)}$ thanks to Equation (5).

2. The rule (effect) for states means that two modifiers coincide as soon as they return the same result and modify the state in the same way. This means that $f_1^{(2)} \cong f_2^{(2)}$ if and only if $f_1 \sim f_2$ and $\langle\ \rangle_A \circ f_1 \cong \langle\ \rangle_A \circ f_2$. Thanks to Corollary A.4 the equation $f_1 \sim f_2$ is equivalent to an equation

between accessors. It remains to prove that the equation $\langle\ \rangle_A \circ f_1 \cong \langle\ \rangle_A \circ f_2$ is also equivalent to an equation between accessors.

For $i \in \{1, 2\}$, since $\langle\ \rangle_A \circ f_i \colon A \to \mathbb{1}$, Proposition A.2 says that $\langle\ \rangle_A \circ f_i \cong \mathtt{upd} \circ a_i$ for some accessor $a_i \colon A \to V$. Thus, $\langle\ \rangle_A \circ f_1 \cong \langle\ \rangle_A \circ f_2$ if and only if $\mathtt{upd} \circ a_1 \cong \mathtt{upd} \circ a_2$. Let us check that this equation is equivalent to $a_1 \cong a_2$.

Clearly if $a_1 \cong a_2 \colon A \to V$ then $\mathtt{upd} \circ a_1 \cong \mathtt{upd} \circ a_2$. Conversely, if $\mathtt{upd} \circ a_1 \cong \mathtt{upd} \circ a_2 \colon A \to \mathbb{1}$ then $\mathtt{lkp} \circ \mathtt{upd} \circ a_1 \cong \mathtt{lkp} \circ \mathtt{upd} \circ a_2$ and since $\mathtt{lkp} \circ \mathtt{upd} \sim id_V$ we get $a_1 \sim a_2$, which is the same as $a_1 \cong a_2$ because a_1 and a_2 are accessors.

Thus, $\langle\ \rangle_A \circ f_1 \cong \langle\ \rangle_A \circ f_2$ if and only if $a_1 \cong a_2$, as required.

$\square$