HAL open science

# Decorated proofs for computational effects: States

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

▶ **To cite this version:**

Jean-Guillaume Dumas, Dominique Duval, Laurent Fousse, Jean-Claude Reynaud. Decorated proofs for computational effects: States. Electronic Proceedings in Theoretical Computer Science, 2012, 93, pp.45-59. 10.4204/EPTCS.93.3 . hal-00650269v2

**HAL Id: hal-00650269**

**https://hal.science/hal-00650269v2**

Submitted on 20 Dec 2011

**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.

# Decorated proofs for computational effects: States

Jean-Guillaume Dumas[*], Dominique Duval[†], Laurent Fousse[‡], Jean-Claude Reynaud[§]

December 15., 2011 (v.2)

**Abstract**

**Abstract.** The syntax of an imperative language does not mention explicitly the state, while its denotational semantics has to mention it. In this paper we show that the equational proofs about an imperative language may hide the state, in the same way as the syntax does.

## Introduction

The evolution of the state of the memory in an imperative program is a computational effect: the state is never mentioned as an argument or a result of a command, whereas in general it is used and modified during the execution of commands. Thus, the syntax of an imperative language does not mention explicitly the state, while its denotational semantics has to mention it. This means that the state is *encapsulated*: its interface, which is made of the functions for looking up and updating the values of the locations, is separated from its implementation; the state cannot be accessed in any other way than through his interface. In this paper we show that equational proofs in an imperative language may also encapsulate the state: proofs can be performed without any knowledge of the implementation of the state. We will see that a naive approach (called "apparent") cannot deal with the updating of states, while this becomes possible with a slightly more sophisticated approach (called "decorated"). This is expressed in an algebraic framework relying on category theory. To our knowledge, the first categorical treatment of computational effects, using monads, is due to Moggi [Moggi 1991]. The examples proposed by Moggi include the side-effects monad $T(A) = (A \times St)^{St}$ where $St$ is the set of states. Later on, Plotkin and Power used Lawvere theories for dealing with the operations and equations related to computational effects. The Lawvere theory for the side-effects monad involves seven equations [Plotkin & Power 2002]. In Section 1 we describe the intended denotational semantics of states. Then in Section 2 we introduce three variants of the equational logic for formalizing the computational effects due to the states: the *apparent*, *decorated* an *explicit* logics. This approach is illustrated in Section 3 by proving some of the equations from [Plotkin & Power 2002], using rules which do not mention any type of states.

## 1 Motivations

This section is made of three independent parts. Section 1.1 is devoted to the semantics of states, an example is presented in Section 1.2, and our logical framework is described in Section 1.3.

[*]LJK, Université de Grenoble, France. `Jean-Guillaume.Dumas@imag.fr`
[†]LJK, Université de Grenoble, France. `Dominique.Duval@imag.fr`
[‡]LJK, Université de Grenoble, France. `Laurent.Fousse@imag.fr`
[§]Malhivert, Claix, France. `Jean-Claude.Reynaud@imag.fr`

## 1.1 Semantics of states

This section deals with the denotational semantics of states, by providing a set-valued interpretation of the *lookup* and *update* operations. Let $St$ denote the set of *states*. Let $Loc$ denote the set of *locations* (also called *variables* or *identifiers*). For each location $i$, let $Val_i$ denote the set of possible *values* for $i$. For each location $i$ there is a *lookup* function for reading the value of location $i$ in the given state, without modifying this state: this corresponds to a function $lookup_{i,1} : St \to Val_i$ or equivalently to a function $lookup_i : St \to Val_i \times St$ such that $lookup_i(s) = \langle lookup_{i,1}(s), s\rangle$ for each state $s$. In addition, for each location $i$ there is an *update* function $update_i : Val_i \times St \to St$ for setting the value of location $i$ to the given value, without modifying the values of the other locations in the given state. This is summarized as follows, for each $i \in Loc$: a set $Val_i$, two functions $lookup_{i,1} : St \to Val_i$ and $update_i : Val_i \times St \to St$, and equations (1):

(1.1) $\forall\, a \in Val_i\,,\ \forall\, s \in St\,,\ lookup_{i,1}(update_i(a, s)) = a\ ,$

(1.2) $\forall\, a \in Val_i\,,\ \forall\, s \in St\,,\ lookup_{j,1}(update_i(a, s)) = lookup_{j,1}(s)$ for every $j \in Loc,\ j \neq i\ .$

The state can be observed thanks to the lookup functions. We may consider the tuple $\langle lookup_{i,1}\rangle_{i\in Loc} : St \to \prod_{i\in Loc} Val_i$. If this function is an isomorphism, then Equations (1) provide a definition of the update functions. In [Plotkin & Power 2002] an equational presentation of states is given, with seven equations: in Remark 1.1 these equations are expressed according to [Melliès 2010] and they are translated in our framework. We use the notations $l_i = lookup_i : St \to Val_i \times St$, $l_{i,1} = lookup_{i,1} : St \to Val_i$ and $u_i = update_i : Val_i \times St \to St$, and in addition $id_i : Val_i \to Val_i$ and $q_i : Val_i \times St \to St$ respectively denote the identity of $Val_i$ and the projection, while $perm_{i,j} : Val_j \times Val_i \times St \to Val_i \times Val_j \times St$ permutes its first and second arguments.

**Remark 1.1.** The equations in [Plotkin & Power 2002] can be expressed as the following Equations (2):

(2.1) Annihilation lookup-update. *Reading the value of a location $i$ and then updating the location $i$ with the obtained value is just like doing nothing.*
$\forall\, i \in Loc,\ \forall\, s \in St,\ \ u_i(l_i(s)) = s \in St$

(2.2) Interaction lookup-lookup. *Reading twice the same location loc is the same as reading it once.*
$\forall\, i \in Loc,\ \forall\, s \in St,\ \ l_i(q_i(l_i(s))) = l_i(s) \in Val_i \times St$

(2.3) Interaction update-update. *Storing a value $a$ and then a value $a'$ at the same location $i$ is just like storing the value $a'$ in the location.*
$\forall\, i \in Loc,\ \forall\, s \in St,\ \forall\, a, a' \in Val_i,\ \ u_i(a', u_i(a, s)) = u_i(a', s) \in St$

(2.4) Interaction update-lookup. *When one stores a value $a$ in a location $i$ and then reads the location $i$, one gets the value $a$.*
$\forall\, i \in Loc,\ \forall\, s \in St,\ \forall\, a \in Val_i,\ \ l_{i,1}(u_i(a, s)) = a \in Val_i$

(2.5) Commutation lookup-lookup. *The order of reading two different locations $i$ and $j$ does not matter.*
$\forall\, i \neq j \in Loc,\ \forall\, s \in St,\ \ (id_i \times l_j)(l_i(s)) = perm_{i,j}((id_j \times l_i)(l_j(s))) \in Val_i \times Val_j \times St$

(2.6) Commutation update-update. *The order of storing in two different locations $i$ and $j$ does not matter.*
$\forall\, i \neq j \in Loc,\ \forall\, s \in St,\ \forall\, a \in Val_i,\ \forall\, b \in Val_j,\ \ u_j(b, u_i(a, s)) = u_i(a, u_j(b, s)) \in St$

(2.7) Commutation update-lookup. *The order of storing in a location $i$ and reading in another location $j$ does not matter.*
$\forall\, i \neq j \in Loc,\ \forall\, s \in St,\ \forall\, a \in Val_i,\ \ l_j(u_i(a, s)) = (id_j \times u_i)(perm_{j,i}(a, l_j(s))) \in Val_j \times St$

**Proposition 1.2.** *Let us assume that $\langle l_{i,1}\rangle_{i\in Loc} : St \to \prod_{i\in Loc} Val_i$ is invertible. Then Equations (1) are equivalent to Equations (2).*

*Proof.* It may be observed that (2.4) is exactly (1.1). In addition, (2.7) is equivalent to (1.2) : indeed, (2.7) is equivalent to the conjunction of its projection on $Val_j$ and its projection on $St$; the first one is $l_{j,1}(u_i(a,s)) = l_{j,1}(s)$, which is (1.2), and the second one is $u_i(a,s) = u_i(a,s)$. Equations (2.2) and (2.5) follow from $q_i(l_i(s)) = s$. For the remaining equations (2.1), (2.3) and (2.6), which return states, it is easy to check that for each location $k$, by applying $l_k$ to both members and using equation (1.1) or (1.2) according to $k$, we get the same value in $Val_k$ for both hand-sides. Then equations (2.1), (2.3) and (2.6) follow from the fact that $\langle l_{i,1}\rangle_{i\in Loc} : St \to \prod_{i\in Loc} Val_i$ is invertible. □

Proposition 1.2 will be revisited in Section 3, where it will be proved that equations (1) imply equations (2) without ever mentioning explicitly the state in the proof.

## 1.2 Computational effects: an example

In an informal way, we consider that a computational effect occurs when there is an apparent mismatch, i.e., some lack of soundness, between the syntax and the denotational semantics of a language. For instance in an object-oriented language, the state of an object does not appear explicitly as an argument nor as a result of any of its methods. In this section, as a toy example, we build a class `BankAccount` for managing (very simple!) bank accounts. We use the types `int` and `void`, and we assume that `int` is interpreted by the set of integers $\mathbb{Z}$ and `void` by a singleton $\{\star\}$. In the class `BankAccount`, there is a method `balance()` which returns the current balance of the account and a method `deposit(x)` for the deposit of `x` Euros on the account. The `deposit` method is a *modifier*, which means that it can use and modify the state of the current account. The `balance` method is an *inspector*, or an *accessor*, which means that it can use the state of the current account but it is not allowed to modify this state. In the object-oriented language `C++`, a method is called a *member function*; by default a member function is a modifier, when it is an accessor it is called a *constant member function* and the keyword `const` is used. So, the `C++` syntax for declaring the member functions of the class `BankAccount` looks like:

```
int balance ( ) const ;
void deposit (int) ;
```

- Forgetting the keyword `const`, this piece of `C++` syntax can be translated as a signature $Bank_{\text{app}}$, which we call the *apparent signature* (we use the word "apparent" in the sense of "seeming" i.e., "appearing as such but not necessarily so").

$$Bank_{\text{app}} : \begin{cases} \texttt{balance} : \texttt{void} \to \texttt{int} \\ \texttt{deposit} : \texttt{int} \to \texttt{void} \end{cases}$$

  In a model (or algebra) of the signature $Bank_{\text{app}}$, the operations would be interpreted as functions:

$$\begin{cases} [[\texttt{balance}]] : \{\star\} \to \mathbb{Z} \\ [[\texttt{deposit}]] : \mathbb{Z} \to \{\star\} \end{cases}$$

  which clearly is not the intended interpretation.

- In order to get the right semantics, we may use another signature $Bank_{\text{expl}}$, which we call the *explicit signature*, with a new symbol `state` for the "type of states":

$$Bank_{\text{expl}} : \begin{cases} \texttt{balance} : \texttt{state} \to \texttt{int} \\ \texttt{deposit} : \texttt{int} \times \texttt{state} \to \texttt{state} \end{cases}$$

  The intended interpretation is a model of the explicit signature $Bank_{\text{expl}}$, with $St$ denoting the set of states of a bank account:

$$\begin{cases} [[\texttt{balance}]] : St \to \mathbb{Z} \\ [[\texttt{deposit}]] : \mathbb{Z} \times St \to St \end{cases}$$

So far, in this example, we have considered two different signatures. On the one hand, the apparent signature $Bank_{\text{app}}$ is simple and quite close to the `C++` code, but the intended semantics is not a model of $Bank_{\text{app}}$. On the other hand, the semantics is a model of the explicit signature $Bank_{\text{expl}}$, but $Bank_{\text{expl}}$ is far from the `C++` syntax: actually, the very nature of the object-oriented language is lost by introducing a "type of states". Let us now define a *decorated signature* $Bank_{\text{deco}}$, which is still closer to the `C++` code than the apparent signature and which has a model corresponding to the intended semantics. The decorated signature is not exactly a signature in the classical sense, because there is a classification of its operations. This classification is provided by superscripts called *decorations*: the decorations `(1)` and `(2)` correspond respectively to the object-oriented notions of *accessor* and *modifier*.

$$Bank_{\text{deco}} : \begin{cases} \texttt{balance}^{(1)} : \texttt{void} \to \texttt{int} \\ \texttt{deposit}^{(2)} : \texttt{int} \to \texttt{void} \end{cases}$$

The decorated signature is similar to the `C++` code, with the decoration `(1)` corresponding to the keyword `const`. The apparent specification $Bank_{\text{app}}$ may be recovered from $Bank_{\text{deco}}$ by dropping the decorations. In addition, we claim that the intended semantics can be seen as a *decorated model* of this decorated signature: this will become clear in Section 2.3. In order to add to the signature constants of type `int` like `0`, `1`, `2`, ... and the usual operations on integers, a third decoration is used: the decoration `(0)` for *pure* functions, which means, for functions which neither inspect nor modify the state of the bank account. So, we add to the apparent and explicit signatures the constants `0`, `1`, ... : $\texttt{void} \to \texttt{int}$ and the operations `+`, `-`, $*$ : $\texttt{int} \times \texttt{int} \to \texttt{int}$, and we add to the decorated signature the pure constants $\texttt{0}^{(0)}$, $\texttt{1}^{(0)}$, ... : $\texttt{void} \to \texttt{int}$ and the pure operations $\texttt{+}^{(0)}$, $\texttt{-}^{(0)}$, $*^{(0)}$ : $\texttt{int} \times \texttt{int} \to \texttt{int}$. For instance the `C++` expressions

```
deposit(7); balance()   and   7 + balance()
```

can be seen as the decorated terms:

$$\texttt{balance}^{(1)} \circ \texttt{deposit}^{(2)} \circ \texttt{7}^{(0)} \quad \text{and} \quad \texttt{+}^{(0)} \circ \langle \texttt{7}^{(0)}, \texttt{balance}^{(1)} \rangle$$

which may be illustrated as:

$$\texttt{void} \xrightarrow{\texttt{7}^{(0)}} \texttt{int} \xrightarrow{\texttt{deposit}^{(2)}} \texttt{void} \xrightarrow{\texttt{balance}^{(1)}} \texttt{int}$$

$$\text{and} \quad \texttt{void} \xrightarrow{\langle \texttt{7}^{(0)}, \texttt{balance}^{(1)} \rangle} \texttt{int} \times \texttt{int} \xrightarrow{\texttt{+}^{(0)}} \texttt{int}$$

These two decorated terms have different effects: the first one does modify the state while the second one is an accessor; however, both return the same integer. Let us introduce the symbol $\sim$ for the relation "same result, maybe distinct effects". Then:

$$\texttt{balance}^{(1)} \circ \texttt{deposit}^{(2)} \circ \texttt{7}^{(0)} \quad \sim \quad \texttt{+}^{(0)} \circ \langle \texttt{7}^{(0)}, \texttt{balance}^{(1)} \rangle$$

## 1.3 Diagrammatic logics

In this paper, in order to deal with a relevant notion of morphisms between logics, we define a *logic* as a *diagrammatic logic*, in the sense of [Domínguez & Duval 2010]. For the purpose of this paper let us simply say that a logic $\mathcal{L}$ determines a category of theories $\mathbf{T}$ which is cocomplete, and that a morphism of logics is a left adjoint functor, so that it preserves the colimits. The objects of $\mathbf{T}$ are called the a *theories* of the logic $\mathcal{L}$. Quite often, $\mathbf{T}$ is a category of structured categories. The *inference rules* of the logic $\mathcal{L}$ describe the structure of its theories. When a theory $\Phi$ is generated by some presentation or *specification* $\Sigma$, a *model* of $\Sigma$ with values in a theory $\Theta$ is a morphism $M : \Phi \to \Theta$ in $\mathbf{T}$.

**The monadic equational logic.** For instance, and for future use in the paper, here is the way we describe the *monadic equational logic* $\mathcal{L}_{meqn}$. In order to focus on the syntactic aspect of the theories, we use a congruence symbol "$\equiv$" rather than the equality symbol "$=$". Roughly speaking, a monadic equational theory is a sort of category where the axioms hold only up to congruence (in fact, it is a 2-category). Precisely, a *monadic equational theory* is a directed graph (its vertices are called *objects* or *types* and its edges are called *morphisms* or *terms*) with an *identity* term $id_X : X \to X$ for each type $X$ and a *composed* term $g \circ f : X \to Z$ for each pair of consecutive terms $(f : X \to Y, g : Y \to Z)$; in addition it is endowed with *equations* $f \equiv g : X \to Y$ which form a *congruence*, which means, an equivalence relation on parallel terms compatible with the composition; this compatibility can be split in two parts: *substitution* and *replacement*. In addition, the associativity and identity axioms hold up to congruence. These properties of the monadic equational theories can be described by a set of *inference rules*, as in Figure 1.

$$\text{(id)}\ \frac{X}{id_X : X \to X} \qquad \text{(comp)}\ \frac{f : X \to Y \quad g : Y \to Z}{g \circ f : X \to Z}$$

$$\text{(id-src)}\ \frac{f : X \to Y}{f \circ id_X \equiv f} \qquad \text{(id-tgt)}\ \frac{f : X \to Y}{id_Y \circ f \equiv f} \qquad \text{(assoc)}\ \frac{f : X \to Y \quad g : Y \to Z \quad h : Z \to W}{h \circ (g \circ f) \equiv (h \circ g) \circ f}$$

$$(\equiv\text{-refl})\ \frac{}{f \equiv f} \qquad (\equiv\text{-sym})\ \frac{f \equiv g}{g \equiv f} \qquad (\equiv\text{-trans})\ \frac{f \equiv g \quad g \equiv h}{f \equiv h}$$

$$(\equiv\text{-subs})\ \frac{f : X \to Y \quad g_1 \equiv g_2 : Y \to Z}{g_1 \circ f \equiv g_2 \circ f : X \to Z} \qquad (\equiv\text{-repl})\ \frac{f_1 \equiv f_2 : X \to Y \quad g : Y \to Z}{g \circ f_1 \equiv g \circ f_2 : X \to Z}$$

Figure 1: Rules of the monadic equational logic

**Adding products to the monadic equational logic.** In contrast with equational theories, the existence of products is not required in a monadic equational theory. However some specific products may exist. A product in a monadic equational theory $\mathbf{T}$ is "up to congruence", in the following sense. Let $(Y_i)_{i \in I}$ be a family of objects in $\mathbf{T}$, indexed by some set $I$. A *product* with base $(Y_i)_{i \in I}$ is a cone $(q_i : Y \to Y_i)_{i \in I}$ such that for every cone $(f_i : X \to Y_i)_{i \in I}$ on the same base there is a term $\langle f_i \rangle_{i \in I} : X \to Y$ such that $q_i \circ \langle f_i \rangle_{i \in I} \equiv f_i$ for each $i$, and in addition this term is unique up to congruence, in the sense that if $f, g : X \to Y$ are such that $q_i \circ f \equiv q_i \circ g$ for each $i$ then $f \equiv g$. When $I$ is empty, we get a *terminal* object $\mathbb{1}$, such that for every $X$ there is an arrow $\langle\,\rangle_X : X \to \mathbb{1}$ which is unique up to congruence. The corresponding inference rules are given in Figure 2. The quantification "$\forall i$", or "$\forall i \in I$", is a kind of "syntactic sugar": when occuring in the premisses of a rule it stands for a conjunction of premisses.

## 2 Three logics for states

In this section we introduce three logics for dealing with states as computational effects. This generalizes the example of the bank account in Section 1.2. We present first the explicit logic (close to the semantics), then the apparent logic (close to the syntax), and finally the decorated logic and the morphisms from the decorated logic to the apparent and the explicit ones. In the syntax of an imperative language there is no type of states (the state is "hidden") while the interpretation of this language involves a set of states $St$. More precisely, if the types $X$ and $Y$ are interpreted as the sets $[[X]]$ and $[[Y]]$, then each term $f : X \to Y$ is interpreted as a function $[[f]] : [[X]] \times St \to [[Y]] \times St$. In Moggi's paper introducing monads for effects [Moggi 1991] such a term $f : X \to Y$ is called a *computation*, and whenever the function $[[f]]$ is $[[f]]_0 \times id_{St}$ for some $[[f]]_0 : [[X]] \to [[Y]]$ then $f$ is called a *value*. We keep this distinction, using *modifier* and *pure*

When $(q_i : Y \to Y_i)_{i \in I}$ is a product:

$$\text{(tuple)}\ \frac{(f_i : X \to Y_i)_i}{\langle f_i \rangle_i : X \to Y} \qquad \text{(tuple-proj-}i\text{)}\ \frac{(f_i : X \to Y_i)_i}{q_i \circ \langle f_j \rangle_j \equiv f_i} \qquad \text{(tuple-unique)}\ \frac{f, g : X \to Y \quad \forall i\ q_i \circ f \equiv q_i \circ g}{f \equiv g}$$

When $\mathbb{1}$ is a terminal type ("empty product"):

$$\text{(final)}\ \frac{X}{\langle\,\rangle_X : X \to \mathbb{1}} \qquad \text{(final-unique)}\ \frac{f, g : X \to \mathbb{1}}{f \equiv g}$$

Figure 2: Rules for products

*term* instead of *computation* and *value*, respectively. In addition, an *accessor* (or *inspector*) is a term $f : X \to Y$ that is interpreted by a function $[[f]] = \langle [[f]]_1, q_X \rangle$, for some $[[f]]_1 : [[X]] \times St \to [[Y]]$, where $q_X : [[X]] \times St \to St$ is the projection. It follows that every pure term is an accessor and every accessor is a modifier. We will respectively use the decorations (0), (1) and (2), written as superscripts, for pure terms, accessors and modifiers. Moreover, we distinguish two kinds of equations: when $f, g : X \to Y$ are parallel terms, then a *strong* equation $f \equiv g$ is interpreted as the equality $[[f]] = [[g]] : [[X]] \times St \to [[Y]] \times St$, while a *weak* equation $f \sim g$ is interpreted as the equality $p_Y \circ [[f]] = p_Y \circ [[g]] : [[X]] \times St \to [[Y]]$, where $p_Y : [[Y]] \times St \to [[Y]]$ is the projection. Clearly, strong and weak equations coincide on accessors and on pure terms, while they differ on modifiers. As in Section 1.1, we consider some given set of locations $Loc$ and for each location $i$ a set $Val_i$ of possible values for $i$. The *set of states* is defined as $St = \prod_{i \in Loc} Val_i$, and the projections are denoted by $lookup_{i,1} : St \to Val_i$. For each location $i$, let $update_i : Val_i \times St \to St$ be defined by Equations (1) as in Section 1.1. In order to focus on the fundamental properties of states as effects, the three logics for states are based on the "poor" monadic equational logic (as described in Section 1.3).

## 2.1 The explicit logic for states

The *explicit logic for states* $\mathcal{L}_{\text{expl}}$ is a kind of "pointed" monadic equational logic: a theory $\Theta_{\text{expl}}$ for $\mathcal{L}_{\text{expl}}$ is a monadic equational theory with a distinguished object $S$, called the *type of states*, and with a product-with-$S$ functor $X \times S$. As in Section 1.2, the explicit logic provides the relevant semantics, but it is far from the syntax. The explicit theory for states $State_{\text{expl}}$ is generated by a type $V_i$ and an operation $l_{i,1} : S \to V_i$ for each location $i$, which form a product $(l_{i,1} : S \to V_i)_{i \in Loc}$. Thus, for each location $i$ there is an operation $u_i : V_i \times S \to S$, unique up to congruence, which satisfies the equations below (where $p_i : V_i \times S \to V_i$ and $q_i : V_i \times S \to S$ are the projections):

$$State_{\text{expl}} : \begin{cases} \text{operations} & l_{i,1} : S \to V_i\ ,\ u_i : V_i \times S \to S \\ \text{product} & (l_{i,1} : S \to V_i)_{i \in Loc} \\ \text{equations} & l_{i,1} \circ u_i \equiv p_i : V_i \times S \to V_i\ ,\ l_{j,1} \circ u_i \equiv l_{j,1} \circ q_i : V_i \times S \to V_j \ \text{ for each } j \neq i \end{cases}$$

Let us define the explicit theory $Set_{\text{expl}}$ as the category of sets with the equality as congruence and with the set of states $St = \prod_{j \in Loc} Val_j$ as its distinguished set. The semantics of states, as described in Section 1.1, is the model $M_{\text{expl}} : State_{\text{expl}} \to Set_{\text{expl}}$ which maps the type $V_i$ to the set $Val_i$ for each $i \in Loc$, the type $S$ to the set $St$, and the operations $l_{i,1}$ and $u_i$ to the functions $lookup_{i,1}$ and $update_i$, respectively.

## 2.2 The apparent logic for states

The *apparent logic for states* $\mathcal{L}_{\text{app}}$ is the monadic equational logic (Section 1.3). As in Section 1.2, the apparent logic is close to the syntax but it does not provide the relevant semantics. The *apparent theory for*

*states* $State_{\text{app}}$ can be obtained from the explicit theory $State_{\text{expl}}$ by identifying the type of states $S$ with the unit type $\mathbb{1}$. So, there is in $State_{\text{app}}$ a terminal type $\mathbb{1}$ and for each location $i$ a type $V_i$ for the possible values of $i$ and an operation $l_i : \mathbb{1} \to V_i$ for observing the value of $i$. A set-valued model for this part of $State_{\text{app}}$, with the constraint that for each $i$ the interpretation of $V_i$ is the given set $Val_i$, is made of an element $a_i \in Val_i$ for each $i$ (it is the image of the interpretation of $l_i$). Thus, such a model corresponds to a state, made of a value for each location; this is known as the *states-as-models* or *states-as-algebras* point of view [Gaudel et al. 1996]. In addition, it is assumed that in $State_{\text{app}}$ the operations $l_i$'s form a product $(l_i : \mathbb{1} \to V_i)_{i \in Loc}$. This assumption implies that each $l_i$ is an isomorphism, so that each $V_i$ must be interpreted as a singleton: this does not fit with the semantics of states. However, we will see in Section 2.3 that this assumption becomes meaningful when decorations are added, in a similar way as in the bank example in Section 1.2. Formally, the assumption that $(l_i : \mathbb{1} \to V_i)_{i \in Loc}$ is a product provides for each location $i$ an operation $u_i : V_i \to \mathbb{1}$, unique up to congruence, which satisfies the equations below (where $id_i : V_i \to V_i$ is the identity and $\langle\,\rangle_i = \langle\,\rangle_{V_i} : V_i \to \mathbb{1}$) :

$$State_{\text{app}} : \begin{cases} \text{operations} & l_i : \mathbb{1} \to V_i \ , \ u_i : V_i \to \mathbb{1} \\ \text{product} & (l_i : \mathbb{1} \to V_i)_{i \in Loc} \quad \text{with terminal type } \mathbb{1} \\ \text{equations} & l_i \circ u_i \equiv id_i : V_i \to V_i \ , \ l_j \circ u_i \equiv l_j \circ \langle\,\rangle_i : V_i \to V_j \quad \text{for each } j \neq i \end{cases}$$

At first view, these equations mean that after $u_i(a)$ is executed, the value of $i$ is put to $a$ and the value of $j$ (for $j \neq i$) is unchanged. However, as noted above, this intuition is not supported by the semantics in the apparent logic. We will see in Section 2.3 that these equations become sound when relevant decorations are added, so that the apparent logic can be used for checking the validity of a decorated proof, as explained in Section 2.4.

## 2.3 The decorated logic for states

Now, as in Section 1.2, we introduce a third logic for states, which is close to the syntax and which provides the relevant semantics. It is defined by adding "decorations" to the apparent logic. A theory $\Theta_{\text{deco}}$ for the *decorated logic for states* $\mathcal{L}_{\text{deco}}$ is made of:

- A monadic equational theory $\Theta^{(2)}$. The terms in $\Theta^{(2)}$ may be called the *modifiers* and the equations $f \equiv g$ may be called the *strong equations*.
- Two additional monadic equational theories $\Theta^{(0)}$ and $\Theta^{(1)}$, with the same types as $\Theta^{(2)}$, and such that $\Theta^{(0)} \subseteq \Theta^{(1)} \subseteq \Theta^{(2)}$ and the congruence on $\Theta^{(0)}$ and on $\Theta^{(1)}$ is the restriction of the congruence on $\Theta^{(2)}$. The terms in $\Theta^{(1)}$ may be called the *accessors*, and if they are in $\Theta^{(0)}$ they may be called the *pure terms*.
- A second equivalence relation $\sim$ between parallel terms in $\Theta^{(2)}$, which is only "weakly" compatible with the composition; the relation $\sim$ satisfies the substitution property but only a weak version of the replacement property, called the *pure replacement*: if $f_1 \sim f_2 : X \to Y$ and $g : Y \to Z$ then in general $g \circ f_1 \not\sim g \circ f_2$, except when $g$ is pure. The relations $f \sim g$ are called the *weak equations*. It is assumed that every strong equation is a weak equation and that every weak equation between accessors is a strong equation, so that the relations $\equiv$ and $\sim$ coincide on $\Theta^{(0)}$ and on $\Theta^{(1)}$.

We use the following notations, called *decorations*: a pure term $f$ is denoted $f^{(0)}$, an accessor $f$ is denoted $f^{(1)}$, and a modifier $f$ is denoted $f^{(2)}$; this last decoration is unnecessary since every term is a modifier, however it may be used for emphasizing. Figure 3 provides the *decorated rules*, which describe the properties of the decorated theories. For readability, the decoration properties may be grouped with other properties: for instance, "$f^{(1)} \sim g^{(1)}$" means "$f^{(1)}$ and $g^{(1)}$ and $f \sim g$".

Some specific kinds of products may be used in a decorated theory, for instance:

- A distinguished type $\mathbb{1}$ with the following *decorated terminality* property: for each type $X$ there is a pure term $\langle\,\rangle_X : X \to \mathbb{1}$ such that every modifier $g : X \to \mathbb{1}$ satisfies $g \sim \langle\,\rangle_X$. It follows from the properties of weak equations that $\mathbb{1}$ is a terminal type in $\Theta^{(0)}$ and in $\Theta^{(1)}$.

Rules of the monadic equational logic, and:

$$\text{(0-id)}\ \frac{X}{id_X^{(0)} : X \to X} \qquad \text{(0-comp)}\ \frac{f^{(0)} \quad g^{(0)}}{(g \circ f)^{(0)}} \qquad \text{(0-to-1)}\ \frac{f^{(0)}}{f^{(1)}} \qquad \text{(1-comp)}\ \frac{f^{(1)} \quad g^{(1)}}{(g \circ f)^{(1)}}$$

$$\text{(1-}\sim\text{-to-}\equiv)\ \frac{f^{(1)} \sim g^{(1)}}{f \equiv g} \qquad (\equiv\text{-to-}\sim)\ \frac{f \equiv g}{f \sim g}$$

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

$$(\sim\text{-subs})\ \frac{f : X \to Y \quad g_1 \sim g_2 : Y \to Z}{g_1 \circ f \sim g_2 \circ f : X \to Z} \qquad \text{(0-}\sim\text{-repl)}\ \frac{f_1 \sim f_2 : X \to Y \quad g^{(0)} : Y \to Z}{g \circ f_1 \sim g \circ f_2 : X \to Z}$$

Figure 3: Rules of the decorated logic for states

- An *observational product* with base $(Y_i)_{i \in I}$ is a cone of accessors $(q_i : Y \to Y_i)_{i \in I}$ such that for every cone of accessors $(f_i : X \to Y_i)_{i \in I}$ on the same base there is a modifier $\langle f_i \rangle_{i \in I} : X \to Y$ such that $q_i \circ \langle f_i \rangle_{i \in I} \sim f_i$ for each $i$, and in addition this modifier is unique up to strong equations, in the sense that if $f, g : X \to Y$ are modifiers such that $q_i \circ f \sim q_i \circ g$ for each $i$ then $f \equiv g$. An observational product allows to prove strong equations from weak ones: by looking at the results of some observations, thanks to the properties of the observational product, we get information on the state.

When $\mathbb{1}$ is a decorated terminal type:

$$\text{(0-final)}\ \frac{X}{\langle\,\rangle_X^{(0)} : X \to \mathbb{1}} \qquad (\sim\text{-final-unique})\ \frac{f, g : X \to \mathbb{1}}{f \sim g}$$

When $(q_i^{(1)} : Y \to Y_i)_i$ is an observational product: $\quad \text{(obs-tuple)}\ \dfrac{(f_i^{(1)} : X \to Y_i)_i}{\langle f_i \rangle_i^{(2)} : X \to Y}$

$$\text{(obs-tuple-proj-}i)\ \frac{(f_i^{(1)} : X \to Y_i)_i}{q_i \circ \langle f_j \rangle_j \sim f_i} \qquad \text{(obs-tuple-unique)}\ \frac{f^{(2)}, g^{(2)} : X \to Y \quad \forall i\ q_i \circ f \sim q_i \circ g}{f \equiv g}$$

Figure 4: Rules for some decorated products for states

The decorated theory of states $State_{\text{deco}}$ is generated by a type $V_i$ and an accessor $l_i^{(1)} : \mathbb{1} \to V_i$ for each $i \in Loc$, which form an observational product $(l_i^{(1)} : \mathbb{1} \to V_i)_{i \in Loc}$. The modifiers $u_i$'s are defined (up to strong equations), using the property of the observational product, by the weak equations below:

$$State_{\text{deco}} : \begin{cases} \text{operations} & l_i^{(1)} : \mathbb{1} \to V_i\ ,\ u_i^{(2)} : V_i \to \mathbb{1} \\ \text{observational product} & (l_i^{(1)} : \mathbb{1} \to V_i)_{i \in Loc} \quad \text{with decorated terminal type } \mathbb{1} \\ \text{equations} & l_i \circ u_i \sim id_i : V_i \to V_i\ ,\ l_j \circ u_i \sim l_j \circ \langle\,\rangle_i : V_i \to V_j \ \text{ for each } j \neq i \end{cases}$$

The decorated theory of sets $Set_{\text{deco}}$ is built from the category of sets, as follows. There is in $Set_{\text{deco}}$ a type for each set, a modifier $f^{(2)} : X \to Y$ for each function $f : X \times St \to Y \times St$, an accessor $f^{(1)} : X \to Y$

for each function $f : X \times St \to Y$, and a pure term $f^{(0)} : X \to Y$ for each function $f : X \to Y$, with the straightforward conversions. Let $f^{(2)}, g^{(2)} : X \to Y$ corresponding to $f, g : X \times St \to Y \times St$. A strong equation $f \equiv g$ is an equality $f = g : X \times St \to Y \times St$, while a weak equation $f \sim g$ is an equality $p \circ f = p \circ g : X \times St \to Y$, where $p : Y \times St \to Y$ is the projection. For each location $i$ the projection $lookup_i : St \to Val_i$ corresponds to an accessor $lookup_i^{(1)} : \mathbb{1} \to Val_i$ in $Set_{\text{deco}}$, so that the family $(lookup_i^{(1)})_{i \in Loc}$ forms an observational product in $Set_{\text{deco}}$. We get a model $M_{\text{deco}}$ of $State_{\text{deco}}$ with values in $Set_{\text{deco}}$ by mapping the type $V_i$ to the set $Val_i$ and the accessor $l_i^{(1)}$ to the accessor $lookup_i^{(1)}$, for each $i \in Loc$. Then for each $i$ the modifier $u_i^{(2)}$ is mapped to the modifier $update_i^{(2)}$.

## 2.4 From decorated to apparent

Every decorated theory $\Theta_{\text{deco}}$ gives rise to an apparent theory $\Theta_{\text{app}}$ by dropping the decorations, which means that the apparent theory $\Theta_{\text{app}}$ is made of a type $X$ for each type $X$ in $\Theta_{\text{deco}}$, a term $f : X \to Y$ for each modifier $f : X \to Y$ in $\Theta_{\text{deco}}$ (which includes the accessors and the pure terms), and an equation $f \equiv g$ for each weak equation $f \sim g$ in $\Theta_{\text{deco}}$ (which includes the strong equations). Thus, the distinction between modifiers, accessors and pure terms disappears, as well as the distinction between weak and strong equations. Equivalently, the apparent theory $\Theta_{\text{app}}$ can be defined as the apparent theory $\Theta^{(2)}$ together with an equation $f \equiv g$ for each weak equation $f \sim g$ in $\Theta_{\text{deco}}$ which is not associated to a strong equation in $\Theta_{\text{deco}}$ (otherwise, it is yet in $\Theta^{(2)}$). Thus, a decorated terminal type in $\Theta_{\text{deco}}$ becomes a terminal type in $\Theta_{\text{app}}$ and an observational product $(q_i^{(1)} : Y \to Y_i)_i$ in $\Theta_{\text{deco}}$ becomes a product $(q_i : Y \to Y_i)_i$ in $\Theta_{\text{app}}$. In the same way, each rule of the decorated logic is mapped to a rule of the apparent logic by dropping the decorations. This property can be used for checking a decorated proof in two steps, by checking on one side the undecorated proof and on the other side the decorations. This construction of $\Theta_{\text{app}}$ from $\Theta_{\text{deco}}$, by dropping the decorations, is a morphism from $\mathcal{L}_{\text{deco}}$ to $\mathcal{L}_{\text{app}}$, denoted $F_{\text{app}}$.

## 2.5 From decorated to explicit

Every decorated theory $\Theta_{\text{deco}}$ gives rise to an explicit theory $\Theta_{\text{expl}}$ by *expanding* the decorations, which means that the explicit theory $\Theta_{\text{expl}}$ is made of:

- A type $X$ for each type $X$ in $\Theta_{\text{deco}}$; projections are denoted by $p_X : X \times S \to X$ and $q_X : X \times S \to S$.
- A term $f_2 : X \times S \to Y \times S$ for each modifier $f : X \to Y$ in $\Theta_{\text{deco}}$, such that:
  - if $f$ is an accessor then there is a term $f_1 : X \times S \to Y$ in $\Theta_{\text{expl}}$ such that $f_2 = \langle f_1, q_X \rangle$,
  - if moreover $f$ is a pure term then there is a term $f_0 : X \to Y$ in $\Theta_{\text{expl}}$ such that $f_1 = f_0 \circ p_X$, hence $f_2 = \langle f_0 \circ p_X, q_X \rangle = f_0 \times id_S$ in $\Theta_{\text{expl}}$.
- An equation $f_2 \equiv g_2 : X \times S \to Y \times S$ for each strong equation $f \equiv g : X \to Y$ in $\Theta_{\text{deco}}$.
- An equation $p_Y \circ f_2 \equiv p_Y \circ g_2 : X \times S \to Y$ for each weak equation $f \sim g : X \to Y$ in $\Theta_{\text{deco}}$.
- A product $(q_{i,1} : Y \times S \to Y_i)_i$ for each observational product $(q_i^{(1)} : Y \to Y_i)_i$ in $\Theta_{\text{deco}}$.

This construction of $\Theta_{\text{expl}}$ from $\Theta_{\text{deco}}$ is a morphism from $\mathcal{L}_{\text{deco}}$ to $\mathcal{L}_{\text{expl}}$, denoted $F_{\text{expl}}$ and called the *expansion*. The expansion morphism makes explicit the meaning of the decorations, by introducing a "type of states" $S$. Thus, each modifier $f$ gives rise to a term $f_2$ which may use and modify the state, while whenever $f$ is an accessor then $f_2$ may use the state but is not allowed to modify it, and when moreover $f$ is pure then $f_2$ may neither use nor modify the state. When $f \equiv g$ then $f_2$ and $g_2$ must return the same result and the same state; when $f \sim g$ then $f_2$ and $g_2$ must return the same result but maybe not the same state. We have seen that the semantics of states cannot be described in the apparent logic, but can be described both in the decorated logic and in the explicit logic. It should be reminded that every morphism of logics is a left adjoint functor. This is the case for the expansion morphism $F_{\text{expl}} : \mathcal{L}_{\text{deco}} \to \mathcal{L}_{\text{expl}}$: it

is a left adjoint functor $F_{\text{expl}} : \mathbf{T}_{\text{deco}} \to \mathbf{T}_{\text{expl}}$, its right adjoint is denoted $G_{\text{expl}}$. In fact, it is easy to check that $Set_{\text{deco}} = G_{\text{expl}}(Set_{\text{expl}})$, and since $State_{\text{expl}} = F_{\text{expl}}(State_{\text{deco}})$ it follows that the decorated model $M_{\text{deco}} : State_{\text{deco}} \to Set_{\text{deco}}$ and the explicit model $M_{\text{expl}} : State_{\text{expl}} \to Set_{\text{expl}}$ are related by the adjunction $F_{\text{expl}} \dashv G_{\text{expl}}$. This means that the models $M_{\text{deco}}$ and $M_{\text{expl}}$ are two different ways to formalize the semantics of states from Section 1.1. In order to conclude Section 2, the morphims of logic $F_{\text{app}}$ and $F_{\text{expl}}$ are summarized in Figure 5.

| $\Theta_{\text{app}} \xleftarrow{F_{\text{app}}}$ | | $\Theta_{\text{deco}} \xrightarrow{F_{\text{expl}}}$ | $\Theta_{\text{expl}}$ |
|---|---|---|---|
| $f : X \to Y$ | modifier | $f^{(2)} : X \to Y$ | $f_2 : X \times S \to Y \times S$ |
| $f : X \to Y$ | accessor | $f^{(1)} : X \to Y$ | $f_1 : X \times S \to Y$ |
| $f : X \to Y$ | pure term | $f^{(0)} : X \to Y$ | $f_0 : X \to Y$ |
| $f \equiv g : X \to Y$ | strong equation | $f \equiv g : X \to Y$ | $f_2 \equiv g_2 : X \times S \to Y \times S$ |
| $f \equiv g : X \to Y$ | weak equation | $f \sim g : X \to Y$ | $p_Y \circ f_2 \equiv p_Y \circ g_2 : X \times S \to Y$ |

Figure 5: A span of logics for states

# 3 Decorated proofs

The inference rules of the decorated logic $\mathcal{L}_{\text{deco}}$ are now used for proving some of the Equations (2) (in Remark 1.1). All proofs in this section are performed in the decorated logic; for readability the identity and associativity rules (id-src) , (id-tgt) and (assoc) are omitted. Some derived rules are proved in Section 3.1, then Equation (2.1) is proved in Section 3.2. In order to deal with the equations with two values as argument or as result, we use the semi-pure products introduced in [Dumas et al. 2011a]; the rules for semi-pure products are reminded in Section 3.3, then all seven Equations (2) are expressed in the decorated logic and Equation (2.6) is proved in Section 3.4. Proving the other equations would be similar. We use as axioms the fact that $l_i$ is an accessor and the weak equations in $State_{\text{deco}}$ (Section 2.3).

## 3.1 Some derived rules

Let us now derive some rules from the rules of the decorated logic (Figures 3 and 4).

| | |
|---|---|
| $(E_1^{(1)})\ \dfrac{f^{(1)} : X \to \mathbb{1} \quad g^{(1)} : X \to \mathbb{1}}{f \equiv g}$ | $(E_1^{(0)})\ \dfrac{f^{(0)} : X \to \mathbb{1} \quad g^{(0)} : X \to \mathbb{1}}{f \equiv g}$ |
| $(E_2^{(1)})\ \dfrac{f^{(1)} : X \to \mathbb{1}}{f \equiv \langle\,\rangle_X}$ | $(E_2^{(0)})\ \dfrac{f^{(0)} : X \to \mathbb{1}}{f \equiv \langle\,\rangle_X}$ |
| $(E_3^{(1)})\ \dfrac{f^{(1)} : X \to Y \quad g^{(1)} : Y \to \mathbb{1} \quad h^{(1)} : X \to \mathbb{1}}{g \circ f \equiv h}$ | $(E_3^{(0)})\ \dfrac{f^{(0)} : X \to Y \quad g^{(0)} : Y \to \mathbb{1} \quad h^{(0)} : X \to \mathbb{1}}{g \circ f \equiv h}$ |
| $(E_4^{(1)})\ \dfrac{f^{(1)} : \mathbb{1} \to X}{\langle\,\rangle_X \circ f \equiv id_{\mathbb{1}}}$ | $(E_4^{(0)})\ \dfrac{f^{(0)} : \mathbb{1} \to X}{\langle\,\rangle_X \circ f \equiv id_{\mathbb{1}}}$ |

Figure 6: Some derived rules in the decorated logic for states

*Proof.* The derived rules in the left part of Figure 6 can be proved as follows. The proof of the rules in the right part are left to the reader.

$$(1\text{-}\sim\text{-to-}\equiv)\ \dfrac{f^{(1)} \qquad g^{(1)} \qquad (\sim\text{-final-unique})\ \dfrac{f,g : X \to \mathbb{1}}{f \sim g}}{f \equiv g\ \ (E_1^{(1)})}$$

$$(E_1^{(1)})\ \dfrac{f^{(1)} : X \to \mathbb{1} \qquad \begin{array}{c} (0\text{-final})\ \dfrac{X}{\langle\,\rangle_X^{(0)} : X \to \mathbb{1}} \\ (0\text{-to-}1)\ \dfrac{\langle\,\rangle_X^{(0)} : X \to \mathbb{1}}{\langle\,\rangle_X^{(1)} : X \to \mathbb{1}} \end{array}}{f \equiv \langle\,\rangle_X\ \ (E_2^{(1)})}$$

$$(E_1^{(1)})\ \dfrac{(1\text{-comp})\ \dfrac{f^{(1)} : X \to Y \qquad g^{(1)} : Y \to \mathbb{1}}{(g \circ f)^{(1)} : X \to \mathbb{1}} \qquad h^{(1)} : X \to \mathbb{1}}{g \circ f \equiv h\ \ (E_3^{(1)})}$$

$$(E_3^{(1)})\ \dfrac{f^{(1)} : \mathbb{1} \to X \qquad \begin{array}{c} (0\text{-final})\ \dfrac{X}{\langle\,\rangle_X^{(0)} : X \to \mathbb{1}} \\ (0\text{-to-}1)\ \dfrac{\langle\,\rangle_X^{(0)} : X \to \mathbb{1}}{\langle\,\rangle_X^{(1)} : X \to \mathbb{1}} \end{array} \qquad \begin{array}{c} (0\text{-id})\ \dfrac{\mathbb{1}}{id_{\mathbb{1}}^{(0)} : \mathbb{1} \to \mathbb{1}} \\ (0\text{-to-}1)\ \dfrac{id_{\mathbb{1}}^{(0)} : \mathbb{1} \to \mathbb{1}}{id_{\mathbb{1}}^{(1)} : \mathbb{1} \to \mathbb{1}} \end{array}}{\langle\,\rangle_X \circ f \equiv id_{\mathbb{1}}\ \ (E_4^{(1)})}$$

□

## 3.2 Annihilation lookup-update

In this section we prove the decorated equation $u_i^{(2)} \circ l_i^{(1)} \equiv id_{\mathbb{1}}^{(0)}$. It is easy to check that this decorated equation gets expanded as $u_i \circ l_i \equiv id_S$, which clearly gets interpreted as Equation (2.1) in Remark 1.1. This decorated equation is now proved using the axioms of $State_{\text{deco}}$ in Section 2.3; for each location $i$:

$$(A_0)\ \ l_i^{(1)}\ , \qquad (A_1)\ \ l_i \circ u_i \sim id_i\ , \qquad (A_2)\ \ l_j \circ u_i \sim l_j \circ \langle\,\rangle_i \ \text{ for each } j \neq i\ .$$

**Proposition 3.1.** *For each location $i$, reading the value of a location $i$ and then updating the location $i$ with the obtained value is just like doing nothing.*

$$u_i^{(2)} \circ l_i^{(1)} \equiv id_{\mathbb{1}}^{(0)} : \mathbb{1} \to \mathbb{1}\ .$$

*Proof.* Let $i$ be a location. Using the unicity property of the observational product (rule (obs-tuple-unique) in Figure 4), we have to prove that $l_k \circ u_i \circ l_i \sim l_k : \mathbb{1} \to V_k$ for each location $k$.

- When $k = i$, the substitution rule for $\sim$ yields:

$$(\sim\text{-subs})\ \dfrac{(A_1)\ \ l_i \circ u_i \sim id_i}{l_i \circ u_i \circ l_i \sim l_i}$$

- When $k \neq i$, using the substitution rule for $\sim$ and the replacement rule for $\equiv$ we get:

$$(\sim\text{-trans})\ \dfrac{(\sim\text{-subs})\ \dfrac{(A_2)\ \ l_k \circ u_i \sim l_k \circ \langle\,\rangle_i}{l_k \circ u_i \circ l_i \sim l_k \circ \langle\,\rangle_i \circ l_i} \qquad (\equiv\text{-to-}\sim)\ \dfrac{(\equiv\text{-repl})\ \dfrac{(E_4^{(1)})\ \dfrac{(A_0)\ \ l_i^{(1)}}{\langle\,\rangle_i \circ l_i \equiv id_{\mathbb{1}}}}{l_k \circ \langle\,\rangle_i \circ l_i \equiv l_k}}{l_k \circ \langle\,\rangle_i \circ l_i \sim l_k}}{l_k \circ u_i \circ l_i \sim l_k}$$

□

### 3.3 Semi-pure products

Let $\Theta_{\text{deco}}$ be a theory with respect to the decorated logic for states and let $\Theta^{(0)}$ be its pure part, so that $\Theta^{(0)}$ is a monadic equational theory. The *product* of two types $X_1$ and $X_2$ in $\Theta_{\text{deco}}$ is defined as their product in $\Theta^{(0)}$ (it is a product up to strong equations, as in Section 1.1). The projections from $X_1 \times X_2$ to $X_1$ and $X_2$ are respectively denoted by $\pi_1^{(0)}$ and $\pi_2^{(0)}$ (the types $X_1$ and $X_2$ will always be clear from the context). The *product* of two pure morphisms $f_1^{(0)} : X_1 \to Y_1$ and $f_2^{(0)} : X_2 \to Y_2$ is a pure morphism $(f_1 \times f_2)^{(0)} = \langle f_1 \circ \pi_1, f_2 \circ \pi_2 \rangle : X_1 \times X_2 \to Y_1 \times Y_2$ subject to the rules in Figure 7, which are the usual rules for products up to strong equations. Moreover when $X_1$ or $X_2$ is $\mathbb{1}$ it can be proved in the usual way that the projections $\pi_1^{(0)} : X_1 \times \mathbb{1} \to X_1$ and $\pi_2^{(0)} : \mathbb{1} \times X_2 \to X_2$ are isomorphisms. The permutation $perm^{(0)}_{X_1,X_2} : X_1 \times X_2 \to X_2 \times X_1$ is defined as usual by $\pi_1 \circ perm_{X_1,X_2} \equiv \pi_2$ and $\pi_2 \circ perm_{X_1,X_2} \equiv \pi_1$.

$$\text{(0-prod)}\ \frac{f_1^{(0)} : X_1 \to Y_1 \quad f_2^{(0)} : X_2 \to Y_2}{(f_1 \times f_2)^{(0)} : X_1 \times X_2 \to Y_1 \times Y_2}$$

$$\text{(0-proj-1)}\ \frac{f_1^{(0)} : X_1 \to Y_1 \quad f_2^{(0)} : X_2 \to Y_2}{\pi_1 \circ (f_1 \times f_2) \equiv f_1 \circ \pi_1} \qquad \text{(0-proj-2)}\ \frac{f_1^{(0)} : X_1 \to Y_1 \quad f_2^{(0)} : X_2 \to Y_2}{\pi_2 \circ (f_1 \times f_2) \equiv f_2 \circ \pi_2}$$

$$\text{(0-prod-unique)}\ \frac{f^{(0)}, g^{(0)} : X \to Y_1 \times Y_2 \quad \pi_1 \circ f \equiv \pi_1 \circ g \quad \pi_2 \circ f \equiv \pi_2 \circ g}{f \equiv g}$$

Figure 7: Rules for products of pure morphisms

The rules in Figure 7, which are symmetric in $f_1$ and $f_2$, cannot be applied to modifiers: indeed, the effect of building a pair of modifiers depends on the evaluation strategy. However, following [Dumas et al. 2011a], we define the *left semi-pure product* of an identity $id_X$ and a modifier $f : X_2 \to Y_2$, as a modifier $id_X \ltimes f : X \times X_2 \to X \times Y_2$ subject to the rules in Figure 8, which form a decorated version of the rules for products. Symmetrically, the *right semi-pure product* of a modifier $f : X_1 \to Y_1$ and an identity $id_X$ is a modifier $f \rtimes id_X : X_1 \times X \to Y_1 \times X$ subject to the rules symmetric to those in Figure 8.

$$\text{(left-prod)}\ \frac{f^{(2)} : X_2 \to Y_2}{(id_X \ltimes f)^{(2)} : X \times X_2 \to X \times Y_2}$$

$$\text{(left-proj-1)}\ \frac{f^{(2)} : X_2 \to Y_2}{\pi_1 \circ (id_X \ltimes f) \sim \pi_1} \qquad \text{(left-proj-2)}\ \frac{f^{(2)} : X_2 \to Y_2}{\pi_2 \circ (id_X \ltimes f) \equiv f \circ \pi_2}$$

$$\text{(left-prod-unique)}\ \frac{f^{(2)}, g^{(2)} : X \to Y_1 \times Y_2 \quad \pi_1 \circ f \sim \pi_1 \circ g \quad \pi_2 \circ f \equiv \pi_2 \circ g}{f \equiv g}$$

Figure 8: Rules for left semi-pure products

Let us add the rules for semi-pure products to the decorated logic for states. In the decorated theory of states $State_{\text{deco}}$, let us assume that there are products $V_i \times V_j$ and $V_i \times \mathbb{1}$ and $\mathbb{1} \times V_j$ for all locations $i$ and $j$. Then it is easy to check that the expansion of the decorated Equations $(2)_d$ below gets interpreted as Equations (2) in Remark 1.1. We use the simplified notations $id_i = id_{V_i}$ and $\langle\,\rangle_i = \langle\,\rangle_{V_i}$ and $perm_{i,j} = perm_{V_i,V_j}$. Equation $(2.1)_d$ has been proved in Section 3.2 and Equation $(2.6)_d$ will be proved in Section 3.4. The other equations can be proved in a similar way.

$(2.1)_d$ Annihilation lookup-update. $\forall\, i \in Loc,\ u_i \circ l_i \equiv id_{\mathbb{1}} : \mathbb{1} \to \mathbb{1}$

$(2.2)_d$ Interaction lookup-lookup. $\forall\, i \in \mathit{Loc},\ l_i \circ \langle\,\rangle_i \circ l_i \equiv l_i : \mathbb{1} \to V_i$

$(2.3)_d$ Interaction update-update. $\forall\, i \in \mathit{Loc},\ \ u_i \circ \pi_2 \circ (u_i \rtimes id_i) \equiv u_i \circ \pi_2 : V_i \times V_i \to \mathbb{1}$

$(2.4)_d$ Interaction update-lookup. $\forall\, i \in \mathit{Loc},\ l_i \circ u_i \sim id_i : V_i \to V_i$

$(2.5)_d$ Commutation lookup-lookup. $\forall\, i \neq j \in \mathit{Loc},\ l_j \circ \langle\,\rangle_i \circ l_i \equiv \mathit{perm}_{j,i} \circ l_i \circ \langle\,\rangle_j \circ l_j : \mathbb{1} \to V_i \times V_j$

$(2.6)_d$ Commutation update-update. $\forall\, i \neq j \in \mathit{Loc},\ u_j \circ \pi_2 \circ (u_i \rtimes id_j) \equiv u_i \circ \pi_1 \circ (id_i \ltimes u_j) : V_i \times V_j \to \mathbb{1}$

$(2.7)_d$ Commutation update-lookup. $\forall\, i \neq j \in \mathit{Loc},\ l_j \circ u_i \equiv \pi_2 \circ (id_i \ltimes l_j) \circ (u_i \rtimes id_j) \circ \pi_1^{-1} : V_i \to V_j$

## 3.4 Commutation update-update

**Proposition 3.2.** *For each locations $i \neq j$ , the order of storing in two different locations $i$ and $j$ does not matter.*

$$u_j^{(2)} \circ \pi_2^{(0)} \circ (u_i \rtimes id_j)^{(2)} \equiv u_i^{(2)} \circ \pi_1^{(0)} \circ (id_i \ltimes u_j)^{(2)} : V_i \times V_j \to \mathbb{1}\ .$$

*Proof.* Let $i$ and $j$ be two distinct locations. Using the unicity property of the observational product (rule (obs-tuple-unique) in Figure 4), we have to prove that $l_k \circ u_j \circ \pi_2 \circ (u_i \rtimes id_j) \sim l_k \circ u_i \circ \pi_1 \circ (id_i \ltimes u_j)$ for each location $k$ .

- When $k \neq i, j$, let us prove independently four weak equations $(W_1)$ to $(W_4)$:

$$(\sim\text{-subs})\ \frac{(A_2)\ \ l_k \circ u_j \sim l_k \circ \langle\,\rangle_j}{l_k \circ u_j \circ \pi_2 \circ (u_i \rtimes id_j) \sim l_k \circ \langle\,\rangle_j \circ \pi_2 \circ (u_i \rtimes id_j)\ \ (W_1)}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\langle\,\rangle_j \circ \pi_2 \equiv \pi_1}\,(E_3^{(0)}) \qquad (\text{right-prod})\ \dfrac{u_i}{u_i \rtimes id_j}}{\langle\,\rangle_j \circ \pi_2 \circ (u_i \rtimes id_j) \equiv \pi_1 \circ (u_i \rtimes id_j)}\,(\equiv\text{-subs}) \qquad (\text{right-proj-1})\ \dfrac{u_i}{\pi_1 \circ (u_i \rtimes id_j) \equiv u_i \circ \pi_1}}{\langle\,\rangle_j \circ \pi_2 \circ (u_i \rtimes id_j) \equiv u_i \circ \pi_1}\,(\equiv\text{-trans})}{\dfrac{l_k \circ \langle\,\rangle_j \circ \pi_2 \circ (u_i \rtimes id_j) \equiv l_k \circ u_i \circ \pi_1}{l_k \circ \langle\,\rangle_j \circ \pi_2 \circ (u_i \rtimes id_j) \sim l_k \circ u_i \circ \pi_1\ \ (W_2)}\,(\equiv\text{-to-}\sim)}\,(\equiv\text{-repl})$$

$$(\sim\text{-subs})\ \frac{(A_2)\ \ l_k \circ u_i \sim l_k \circ \langle\,\rangle_i}{l_k \circ u_i \circ \pi_1 \sim l_k \circ \langle\,\rangle_i \circ \pi_1\ \ (W_3)} \qquad \dfrac{\dfrac{\dfrac{\vdots}{\langle\,\rangle_i \circ \pi_1 \equiv \langle\,\rangle_{V_i \times V_j}}\,(E_3^{(0)})}{l_k \circ \langle\,\rangle_i \circ \pi_1 \equiv l_k \circ \langle\,\rangle_{V_i \times V_j}}\,(\equiv\text{-subs})}{l_k \circ \langle\,\rangle_i \circ \pi_1 \sim l_k \circ \langle\,\rangle_{V_i \times V_j}\ \ (W_4)}\,(\equiv\text{-to-}\sim)$$

  Equations $(W_1)$ to $(W_4)$ together with the transitivity rule for $\sim$ give rise to the weak equation:

$$l_k \circ u_j \circ \pi_2 \circ (u_i \rtimes id_j) \sim l_k \circ \langle\,\rangle_{V_i \times V_j}.$$

  A symmetric proof shows that $l_k \circ u_i \circ \pi_1 \circ (id_i \ltimes u_j) \sim l_k \circ \langle\,\rangle_{V_i \times V_j}$. With the symmetry and transitivity rules for $\sim$, this concludes the proof when $k \neq i, j$.

- When $k = i$, on the one hand it is easy to prove that $l_i \circ u_i \circ \pi_1 \circ (id_i \ltimes u_j) \sim \pi_1$, as follows.

$$\dfrac{(\sim\text{-subs})\ \dfrac{(A_1)\ \ l_i \circ u_i \sim id_i}{l_i \circ u_i \circ \pi_1 \circ (id_i \ltimes u_j) \sim \pi_1 \circ (id_i \ltimes u_j)} \qquad (\text{left-proj-1})\ \dfrac{u_j}{\pi_1 \circ (id_i \ltimes u_j) \sim \pi_1}}{l_i \circ u_i \circ \pi_1 \circ (id_i \ltimes u_j) \sim \pi_1}\,(\sim\text{-trans})$$

  On the other hand it can also be proved that $l_i \circ u_j \circ \pi_2 \circ (u_i \rtimes id_j) \sim \pi_1$, as follows.

$$
\dfrac{\dfrac{\dfrac{(A_2)\ \ l_i \circ u_j \sim l_i \circ \langle\,\rangle_j}{l_i \circ u_j \circ \pi_2 \sim l_i \circ \langle\,\rangle_j \circ \pi_2}\,(\sim\text{-subs}) \qquad \dfrac{\dfrac{\dfrac{\vdots}{\langle\,\rangle_j \circ \pi_2 \equiv \langle\,\rangle_{1\times V_j}}\,(E_3^{(0)})}{l_i \circ \langle\,\rangle_j \circ \pi_2 \equiv l_i \circ \langle\,\rangle_{1\times V_j}}\,(\equiv\text{-repl})}{l_i \circ \langle\,\rangle_j \circ \pi_2 \sim l_i \circ \langle\,\rangle_{1\times V_j}}\,(\equiv\text{-to-}\sim)}{l_i \circ u_j \circ \pi_2 \sim l_i \circ \langle\,\rangle_{1\times V_j}}\,(\sim\text{-trans})}{l_i \circ u_j \circ \pi_2 \circ (u_i \rtimes id_j) \sim l_i \circ \langle\,\rangle_{1\times V_j} \circ (u_i \rtimes id_j)\ \ (W_1')}\,(\sim\text{-subs})
$$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\langle\,\rangle_{\mathbb{1}\times V_j} \equiv \pi_1}\,(E_1^{(0)})}{\langle\,\rangle_{1\times V_j} \circ (u_i \rtimes id_j) \equiv \pi_1 \circ (u_i \rtimes id_j)}\,(\equiv\text{-subs}) \qquad \dfrac{u_i}{\pi_1 \circ (u_i \rtimes id_j) \equiv u_i \circ \pi_1}\,(\text{right-proj-1})}{\langle\,\rangle_{1\times V_j} \circ (u_i \rtimes id_j) \equiv u_i \circ \pi_1}\,(\equiv\text{-trans})}{l_i \circ \langle\,\rangle_{1\times V_j} \circ (u_i \rtimes id_j) \equiv l_i \circ u_i \circ \pi_1}\,(\equiv\text{-repl})}{l_i \circ \langle\,\rangle_{1\times V_j} \circ (u_i \rtimes id_j) \sim l_i \circ u_i \circ \pi_1\ \ (W_2')}\,(\equiv\text{-to-}\sim)
$$

$$
\dfrac{(A_1)\ \ l_i \circ u_i \sim id_i}{l_i \circ u_i \circ \pi_1 \sim \pi_1\ \ (W_3')}\,(\sim\text{-subs})
$$

Equations $(W_1')$ to $(W_3')$ and the transitivity rule for $\sim$ give rise to $l_i \circ u_j \circ \pi_2 \circ (u_i \rtimes id_j) \sim \pi_1$. With the symmetry and transitivity rules for $\sim$, this concludes the proof when $k = i$.

- The proof when $k = j$ is symmetric to the proof when $k = i$.

□

# Conclusion

In this paper, decorated proofs are used for proving properties of states. This can be applied to other computational effects, like exceptions [Dumas et al. 2011b]. In addition, associating to each effect a span of logics as in Section 2 should result in a simple framework for combining effects.

# References


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

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

[Dumas et al. 2011b] Jean-Guillaume Dumas, Dominique Duval, Laurent Fousse, Jean-Claude Reynaud. States and exceptions considered as dual effects. arXiv:1001.1662v4 (2011).

[Gaudel et al. 1996] Marie-Claude Gaudel, Pierre Dauchy, Carole Khoury. A Formal Specification of the Steam-Boiler Control Problem by Algebraic Specifications with Implicit State. Formal Methods for Industrial Applications 1995. Springer-Verlag Lecture Notes in Computer Science 1165, p. 233-264 (1996).

[Melliès 2010] Paul-André Melliès. Segal condition meets computational effects. LICS 2010. IEEE Computer Society, p. 150-159 (2010).

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

[Plotkin & Power 2002] Gordon D. Plotkin, John Power. Notions of Computation Determine Monads. FoSSaCS 2002. Springer-Verlag Lecture Notes in Computer Science 2303, p. 342-356 (2002).