HAL open science

How to combine diagrammatic logics

Dominique Duval

▶ To cite this version:

Dominique Duval. How to combine diagrammatic logics. 2009. hal-00432330v2

HAL Id: hal-00432330
https://hal.science/hal-00432330v2

Preprint submitted on 19 Nov 2009

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.

How to combine diagrammatic logics

Dominique Duval
LJK, Université de Grenoble, France, Dominique.Duval@imag.fr

November 19., 2009

This paper is a submission to the contest: How to combine logics? at the World Congress and School on Universal Logic III, 2010. In this version 2, some typos have been corrected.

1 Introduction

We claim that combining "things", whatever these things are, is made easier if these things can be seen as the objects of a category. Then things are related by morphisms, and the categorical machinery provides various tools for combining things, typically limits and colimits, as well as other constructions.

What should be required from an object in a category to call it a logic? The analysis of various kinds of logic brings in light similar issues: which are the sentences of interest? what is the meaning of each sentence? how can a sentence be infered from another one? According to this analysis, each logic should have a syntax, a notion of model, a proof system, and the morphisms should, in some sense, preserve them. In section 2 we define the *diagrammatic logics*, which satisfy the required properties. Then in section 3 the category of diagrammatic logics follows, so that categorical constructions can be used for combining diagrammatic logics. As an example, a combination of logics using an opfibration is presented, in order to study computational side-effects due to the evolution of the state during the execution of an imperative program.

This work stems from the study of the semantics of computational effects in programming languages. It has been influenced by the theory of sketches [Ehresmann 1968, Lellahi 1989] and the sketch entailments [Makkai 1997], by categorical logic [Lawvere 1963], the theory of institutions [Goguen and Burstall 1984], the use of monads in computer science [Moggi 1989], and many other papers and people. Diagrammatic specifications appeared in [Duval 2003].

2 Diagrammatic logics

2.1 Theories and specifications

The word "theory" is widely used in logic, with various meanings. In this paper, a *theory* for a given logic is a class of sentences which is saturated: it is a class of theorems from which no new theorem can be derived inside the given logic. For instance, the theory of groups in equational logic is made of all the theorems which can be proved about groups, starting from the usual equational axioms for groups and using the congruence properties of equality. There are morphisms between theories: for instance, there is an inclusion of the theory of monoids in the theory of groups. A class of axioms generating a theory is often called a "presentation" or a "specification" of this theory (according whether this notion is used by a logician or a computer scientist), in this paper it is called a *specification*. So, in a given logic, every specification generates a theory, while every theory may be seen as a (huge) specification. In categorical terms, this means that the category $\mathbf{T}$ of theories is a *reflective subcategory* of the category $\mathbf{S}$ of specifications. Or equivalently, this means that there is an *adjunction* between $\mathbf{S}$ and $\mathbf{T}$, such that the right adjoint is full and faithful. The right adjoint $R\colon \mathbf{T} \to \mathbf{S}$ allows to consider a theory as a specification, the left adjoint $L\colon \mathbf{S} \to \mathbf{T}$ generates a theory from a specification, and the fact that R is full and faithful means that each theory is saturated. So, let us define an *"abstract diagrammatic logic"* as a functor L with a adjoint R which is full and faithful (the actual definition of a diagrammatic logic, with a syntax, is definition 2.2).

$$\begin{tikzcd} \mathbf{S} \arrow[r, bend left, "L"] \arrow[r, phantom, "\perp"] & \mathbf{T} \arrow[l, bend left, "R"] \end{tikzcd}$$

For instance, the equational logic is obtained by defining on one side the equational theories are the categories with chosen finite products and on the other side the equational specifications either as the signatures with equations or (equivalently) as the finite product sketches.

Categorical logic usually focuses on theories rather than specifications, and requires theories to be some kind of categories: theories for equational logic are categories with finite products, theories for simply-typed lambda-calculus are cartesian closed categories, and so on. Indeed, specifications do not matter as long as one focuses on abstract properties of models, but they are useful for building explicitly such models (section 2.2). More importantly, the mere notion of proof does rely on specifications: a specification is a class of axioms for a given theory, and a proof is a way to add a theorem to this class of axioms, which means that a proof is an enrichment of the given specification following the rules of the logic (section 2.5).

2.2 Models

Now, let $L\colon \mathbf{S} \to \mathbf{T}$ be some fixed diagrammatic logic, with right adjoint R. A *(strict) model* M of a specification Σ in a theory Θ is a morphism of theories $M\colon L\Sigma \to \Theta$. Equivalently,

thanks to the adjunction, a model M of Σ in Θ is a morphism of specifications $M\colon \Sigma \to R\Theta$. The abstract properties of models can be derived from their definition as morphisms of theories, while the explicit construction of models makes use of their definition as morphisms of specifications.

For example in equational logic, let Θ be the category of sets considered as an equational theory: then we recover the usual notion of model of an equational specification. For this logic, in addition, the natural transformations provide an interesting notion of morphisms between models.

2.3 Instances

There are not "enough" morphisms of specifications, in the following sense: given specifications Σ and Σ_1, usually only few morphisms of theories $\theta\colon L\Sigma \to L\Sigma_1$ are of the form $\theta = L\sigma$ for a morphism of specifications $\sigma\colon \Sigma \to \Sigma_1$. In order to get "enough" morphisms between specifications, we introduce instances as generalizations of morphisms.

Definition 2.1 An *entailment* for a diagrammatic logic $L\colon \mathbf{S} \to \mathbf{T}$ is a morphism $\tau\colon \Sigma \to \Sigma'$ in $\mathbf{S}$ such that $L\tau$ is invertible in $\mathbf{T}$. This is denoted $\Sigma \underset{\leftarrow}{\xrightarrow{\tau}} \Sigma'$. An *instance* κ of a specification Σ in a specification Σ_1 is a cospan in $\mathbf{S}$ made of a morphism σ and an entailment τ

$$\Sigma \xrightarrow{\sigma} \Sigma_1' \xleftarrow{\tau} \Sigma_1$$

Two specifications which are related by entailments are equivalent, in the sense that they generate isomorphic theories. An instance $\kappa = (\sigma, \tau)$ is also called a *fraction* with *numerator* σ and *denominator* τ, it is denoted $\kappa = \tau\backslash\sigma\colon \Sigma \to \Sigma_1$. Then $L\kappa$ is defined as $L\kappa = (L\tau)^{-1} \circ L\sigma\colon L\Sigma \to L\Sigma_1$. According to [Gabriel and Zisman 1967], since the right adjoint R is full and faithful, every morphism of theories $\theta\colon L\Sigma \to L\Sigma_1$ is of the form $\theta = L\kappa$ for an instance κ of Σ in Σ_1. So, up to equivalence, the category $\mathbf{T}$ is the category of fractions of $\mathbf{S}$ with denominators the entailments.

2.4 Syntax

The syntax for writing down the axioms and the inference rules of a given logic is provided by *limit sketches* [Ehresmann 1968, Barr and Wells 1999]. A limit sketch $\mathbf{E}$ is a presentation for a category with prescribed limits. Roughly speaking, a limit sketch $\mathbf{E}$ is a graph with *potential* composition, identities and limits, which means that the usual notations for composition, identities and limits can be used, for example there may be an arrow $g \circ f\colon X \to Z$ when $f\colon X \to Y$ and $g\colon Y \to Z$ are consecutive arrows, there may be an arrow $\mathrm{id}_X\colon X \to X$ when X is a point, there may be a point $Y_1 \times Y_2$ with arrows $\mathrm{pr}_1\colon Y_1 \times Y_2 \to Y_1$ and $\mathrm{pr}_2\colon Y_1 \times Y_2 \to Y_2$ when Y_1 and Y_2 are points, and similarly for other limits. But such arrows may as well not exist, and when they exist they do not have to satisfy the usual properties, for example the composition does not have to be associative. A limit sketch $\mathbf{E}$ generates a category $P(\mathbf{E})$ where each potential feature becomes a real one: an arrow $g \circ f$ in $\mathbf{E}$ becomes the real

composition of f and g in $P(\mathbf{E})$, an arrow id_X in $\mathbf{E}$ becomes the real identity of X in $P(\mathbf{E})$, a point $Y_1 \times Y_2$ with the arrows pr_1 and pr_2 in $\mathbf{E}$ becomes the real product of Y_1 and Y_2 with its projections in $P(\mathbf{E})$, and similarly for other limits.

A *(set-valued) realization* (or *model*) of a limit sketch $\mathbf{E}$ interprets each point of $\mathbf{E}$ as a set and each arrow as a function, in such a way that potential features become real ones. The realizations of $\mathbf{E}$ form a category $\mathit{Real}(\mathbf{E})$. The *Yoneda contravariant functor* $\mathcal{Y}_{\mathbf{E}}$ is such that for each point X of $\mathbf{E}$ the realization $\mathcal{Y}_{\mathbf{E}}(X)$ is made of the arrows in $P(\mathbf{E})$ with source X. This allows to identify the opposite of $\mathbf{E}$ to a subcategory $\mathcal{Y}_{\mathbf{E}}(\mathbf{E})$ (or simply $\mathcal{Y}(\mathbf{E})$) of $\mathit{Real}(\mathbf{E})$. This subcategory is *dense* in $\mathit{Real}(\mathbf{E})$, which means, for short, that every realization of $\mathbf{E}$ is a colimit of realizations in $\mathcal{Y}(\mathbf{E})$.

Every morphism of limit sketches $\mathbf{e}\colon \mathbf{E}_S \to \mathbf{E}_T$ defines an adjunction between the categories of realizations, where the right adjoint $R_{\mathbf{e}}\colon \mathit{Real}(\mathbf{E}_T) \to \mathit{Real}(\mathbf{E}_S)$ is the precomposition with $\mathbf{e}$, which may be called the *forgetful* functor with respect to $\mathbf{e}$. Then the left adjoint $L_{\mathbf{e}}\colon \mathit{Real}(\mathbf{E}_S) \to \mathit{Real}(\mathbf{E}_T)$ extends $\mathbf{e}$, contravariantly, it may be called the *freely generating* functor with respect to $\mathbf{e}$. A category is *locally presentable* when it is equivalent to $\mathit{Real}(\mathbf{E})$ for some limit sketch $\mathbf{E}$, and we say that a functor is *locally presentable* when it is (up to equivalence) the left adjoint functor $L_{\mathbf{e}}$ for some morphism of limit sketches $\mathbf{e}$. More precisely, we assume that a locally presentable category comes with a presentation $\mathbf{E}$, and a locally presentable functor with a presentation $\mathbf{e}$. Now we can define diagrammatic logics.

Definition 2.2 A *diagrammatic logic* is a locally presentable functor L such that its right adjoint R is full and faithful. An *inference system* for L is a morphism of limit sketches $\mathbf{e}\colon \mathbf{E}_S \to \mathbf{E}_T$ such that $L = L_{\mathbf{e}}$.

Thanks to the Yoneda contravariant functor, the morphism $\mathbf{e}$ and the functor $L_{\mathbf{e}}$ have similar properties. In particular, $\mathbf{e}$ can be chosen so as to consist of adding inverse arrows for some arrows in $\mathbf{E}_S$. In addition, since every locally presentable category has colimits, the composition of instances in $\mathbf{S}$ can be performed using pushouts.

2.5 Rules and proofs

Given a rule $\frac{\mathcal{H}_1 \ldots \mathcal{H}_n}{\mathcal{C}}$ in a given logic, the hypotheses $\mathcal{H}_1, \ldots, \mathcal{H}_n$ as well as the conclusion $\mathcal{C}$ may be seen as specifications. The hypotheses may be amalgamated, as a unique hypothesis $\mathcal{H}$ made of the colimit of the $\mathcal{H}_i$'s (it is not a sum usually: the relations between the $\mathcal{H}_i$'s are expressed in the sharing of names). Similarly, let $\mathcal{H}'$ be the colimit of $\mathcal{H}$ and $\mathcal{C}$. So, without loss of generality, let us consider a rule $\frac{\mathcal{H}}{\mathcal{C}}$ in a given logic, where the hypothesis $\mathcal{H}$ and the conclusion $\mathcal{C}$ are specifications with colimit $\mathcal{H}'$. The meaning of this rule is that each instance of $\mathcal{H}$ in a theory Θ can be uniquely extended as an instance of $\mathcal{H}'$ in Θ, which means that the morphism $\mathcal{H} \to \mathcal{H}'$ is an entailment, then clearly this yields an instance of $\mathcal{C}$ in Θ. So, basically, a rule is a fraction, i.e., it is an instance of $\mathcal{C}$ in $\mathcal{H}$.

$$\mathcal{H} \rightleftarrows \mathcal{H}' \longleftarrow \mathcal{C}$$

However, there is a distinction between an elementary rule and a derived rule, which will be called respectiveley a rule (or an inference rule) and a proof.

Definition 2.3 An *inference rule* ρ with *hypothesis* $\mathcal{H}$ and *conclusion* $\mathcal{C}$ is an instance $\rho = \tau\backslash\sigma\colon \mathcal{C} \to \mathcal{H}$ of the conclusion in the hypothesis, where σ and τ are in $\mathcal{Y}(\mathbf{E})$. Given an inference rule $\rho = \tau\backslash\sigma\colon \mathcal{C} \to \mathcal{H}$ and an instance $\kappa\colon \mathcal{H} \to \Sigma$ of the hypothesis $\mathcal{H}$ in a specification Σ, the corresponding *inference step* provides the instance $\kappa \circ \rho\colon \mathcal{C} \to \Sigma$ of the conclusion $\mathcal{C}$ in Σ. The commutative triangle in the category of fractions (on the left) is computed thanks to the commutative diagram in the category of specifications (on the right) where the square is a pushout.

$$\begin{tikzcd}
\mathcal{H} \arrow[dr, "\kappa"'] & & \mathcal{C} \arrow[ll, "\rho"'] \arrow[dl, "\kappa\circ\rho"] \\
& \Sigma &
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathcal{H} \arrow[rr, "\tau", shift left] \arrow[dr] & & \mathcal{H}' \arrow[ll, dashed, shift left] \arrow[dr] & & \mathcal{C} \arrow[ll, "\sigma"'] \arrow[dl] \\
& \Sigma_H \arrow[rr, shift left] \arrow[dr, dashed, shift left] & & \Sigma_C \arrow[ll, dashed, shift left] \arrow[dl, dashed, shift left] & \\
& & \Sigma \arrow[ul, dashed, shift left] \arrow[ur, dashed, shift left] & &
\end{tikzcd}$$

A *proof* (or *derivation*, or *derived rule*) is the description of a morphism of theories as an instance composed from inference rules.

It may be noted that the hypothesis is on the denominator side of the rule and the conclusion on the numerator side, in contrast with the usual notation $\frac{\mathcal{H}}{\mathcal{C}}$ which looks like a fraction with the hypothesis as numerator and the conclusion as denominator.

Example 2.4 Let us consider the *modus ponens* rule

$$\frac{A \qquad A \Rightarrow B}{B}$$

from the diagrammatic point of view. In a logic with this rule, a specification Σ is made of (at least) a set F called the set of *formulas*, a subset P of F called the subset of *provable formulas*, and a binary operation $\Rightarrow$ on formulas; this will be denoted simply as $\Sigma = (F, P)$ or even $\Sigma = P$. The modus ponens rule is obtained, essentially, by inverting one morphism of specifications. Let

$$\mathcal{H} = (\{A, B, A \Rightarrow B\}, \{A, A \Rightarrow B\}),\ \mathcal{H}' = (\{A, B, A \Rightarrow B\}, \{A, A \Rightarrow B, B\}),\ \mathcal{C} = (\{C\}, \{C\})$$

where the names A, B and C stand for arbitrary formulas. Let $\tau\colon \mathcal{H} \to \mathcal{H}'$ be the inclusion and $\sigma\colon \mathcal{C} \to \mathcal{H}'$ the morphism which maps C to $A \Rightarrow B$. A theory is a specification where the modus ponens rule is satisfied, which means that $L\tau$ is an isomorphism, i.e., that τ is an entailment. So, the diagrammatic version of the modus ponens rule is the fraction

$$\begin{tikzcd}
\mathcal{H} = \{A, A \Rightarrow B\} \arrow[r, "\subseteq", shift left] & \mathcal{H}' = \{A, A \Rightarrow B, B\} \arrow[l, dashed, shift left] & \mathcal{C} = \{C\} \arrow[l, "C \mapsto A \Rightarrow B"']
\end{tikzcd}$$

Example 2.5 An inference system $\mathbf{e}_{\mathrm{eq}}\colon \mathbf{E}_{\mathrm{eq},S} \to \mathbf{E}_{\mathrm{eq},T}$ for equational logic is described in [Domínguez and Duval 2009]. Focusing on unary operations, for dealing with composition and identities the limit sketch $\mathbf{E}_{\mathrm{eq},S}$ contains the following subsketch

$$\begin{tikzcd}
\mathtt{Selid} \arrow[d, shift right, "\mathtt{i0}"'] \arrow[dr, "\mathtt{selid}"] & & & \mathtt{Comp} \arrow[dl, "\mathtt{comp}"'] \arrow[d, shift left, "\mathtt{i}"] \\
\mathtt{Type} \arrow[u, dashed, shift right] & \mathtt{Term} \arrow[l, shift right, "\mathtt{dom}"'] \arrow[l, shift left, "\mathtt{codom}"] & & \mathtt{Cons} \arrow[u, dashed, shift left] \arrow[ll, shift right, "\mathtt{fst}"'] \arrow[ll, shift left, "\mathtt{snd}"]
\end{tikzcd}$$

with the suitable potential limits, so that the image by Yoneda of this part of $\mathbf{E}_{\mathrm{eq},S}$ is the following diagram of equational specifications:

$$\begin{tikzcd}
\boxed{X \circlearrowleft \mathrm{id}_X} & & & \boxed{X \xrightarrow{f} Y \xrightarrow{g} Z \;\; (g\circ f)} \\
\boxed{X} \arrow[u, shift left, "\subseteq"] \arrow[u, shift right, "|"'] & \boxed{X \underset{X\mapsto Y}{\overset{X\mapsto X}{\rightrightarrows}}} \arrow[ul, "f\mapsto \mathrm{id}_X"'] & \boxed{X \xrightarrow{f} Y} \arrow[r, "f\mapsto g\circ f"] & \boxed{X \xrightarrow{f} Y \xrightarrow{g} Z} \arrow[u, shift right, "|"'] \arrow[u, shift left, "\subseteq"] \arrow[r, "f\mapsto f", "f\mapsto g"'] & \boxed{X \xrightarrow{f} Y \xrightarrow{g} Z}
\end{tikzcd}$$

The arrows `i` and `i0` are entailments: in the limit sketch $\mathbf{E}_{\mathrm{eq},T}$ there is an inverse for each of them. This forms the rules

$$\frac{f\colon X \to Y \quad g\colon Y \to Z}{g \circ f\colon X \to Y} \quad \text{and} \quad \frac{X}{\mathrm{id}_X\colon X \to X}$$

which means that in a theory every pair of consecutive terms can be composed and every type has an identity. The morphism $\mathbf{e}$ is the inclusion $\mathbf{E}_{\mathrm{eq},S} \to \mathbf{E}_{\mathrm{eq},T}$.

3 Combining diagrammatic logics

3.1 Morphisms of logics

Definition 3.1 Let us consider two diagrammatic logics L_1 and L_2. A *morphism* of diagrammatic logics $F\colon L_1 \to L_2$ is a pair of locally presentable functors (F_S, F_T) together with a natural isomorphism $F_T \circ L_1 \cong L_2 \circ F_S$. In practice, it is defined from inference systems $\mathbf{e}_1\colon \mathbf{E}_{S,1} \to \mathbf{E}_{T,1}$ for L_1 and $\mathbf{e}_2\colon \mathbf{E}_{S,2} \to \mathbf{E}_{T,2}$ for L_2 and from a pair of morphisms of limit sketches $\mathbf{e}_S\colon \mathbf{E}_{S,1} \to \mathbf{E}_{S,2}$ and $\mathbf{e}_T\colon \mathbf{E}_{T,1} \to \mathbf{E}_{T,2}$ which form a commutative diagram

$$\begin{tikzcd}
\mathbf{E}_{S,1} \arrow[r, "\mathbf{e}_1"] \arrow[d, "\mathbf{e}_S"'] & \mathbf{E}_{T,1} \arrow[d, "\mathbf{e}_T"] \\
\mathbf{E}_{S,2} \arrow[r, "\mathbf{e}_2"] & \mathbf{E}_{T,2}
\end{tikzcd}$$

This defines the *category of diagrammatic logics*, **DiaLog**.

Clearly a morphism of diagrammatic logics preserves the syntax and the entailments, hence the rules and the proofs. It does also behave well on models, thanks to adjunction: let U_S and U_T denote the right adjoints of F_S and F_T respectively, then for each specification Σ_1 for L_1 and each theory Θ_2 for L_2, there is an isomorphism:

$$Mod_{L_1}(\Sigma_1, U_T\Theta_2) \cong Mod_{L_2}(F_S\Sigma_1, \Theta_2) \tag{1}$$

Now, combining diagrammatic logics can be performed by using categorical constructions in the category **DiaLog**. These constructions may for example involve limits and colimits.

We now focus on opfibrations, because they can be used for dealing with computational effects, which is our motivation for this work. Let us consider a typical computational effect: the evolution of the state during the execution of a program written in an imperative language.

3.2 Imperative programming

The state of the memory never appears explicitly in an imperative program, although most parts of the program are written in view of modifying it. In order to analyze this situation, we make a clear distinction between the commands as they appear in the grammar of the language and the way they should be understood: typically, in the grammar an assignment $X := e$ has two arguments (a variable and an expression) and does not return any value, while it should be understood as a function with three arguments (a variable, an expression and a state) which returns a state. Let V, E, S stand for the sets of variables, expressions and states, respectively, and U for a singleton, then on one side $(:=)\colon V \times E \to U$ and on the other side $(:=)\colon S \times V \times E \to S$. The fact of building a term $f\colon S \times X \to S \times Y$ from any term $f\colon X \to Y$ in a coherent way can be seen as a morphism of logics, from the equational logic L_{eq} to the *pointed* equational logic L^*_{eq}, made of the equational logic with a distinguished sort S.

An operation like the assignment is called a *modifier*, but there are also *pure* operations that neither use nor modify the state, like the arithmetic operations between the numerical constants. Pure operations can be seen as special kinds of modifiers, however the distinction between modifiers and pure operations is fundamental for dealing properly with the evolution of the state. For example when the monad $(S \times -)^S$ on **Set** is used for this purpose, this distinction is provided by the inclusion of **Set** (for the pure operations) in the Kleisli category of the monad (for the modifers) [Moggi 1989]. This distinction can also be provided by indexing, or *decorating*, each operation, with a keyword p if it is pure or m if it is a modifier. Then the decoration propagates to terms (pieces of programs) in the obvious way: a term is pure if it is made only of variables and pure operations, otherwise it is a modifier. Because of the decorations, this logic is not the equational logic any more, but a new logic called the *decorated* equational logic L_{dec}. For example the basic part of the sketch for equational specifications (on the left) is modified as follows (on the right), where `c` stands for the conversion of pure terms into modifiers.

$$\begin{tikzcd} \mathtt{Type} & \mathtt{Term} \arrow[l, shift right, "\mathtt{dom}"'] \arrow[l, shift left, "\mathtt{codom}"] \end{tikzcd} \qquad \begin{tikzcd} & \mathtt{Term}^m \arrow[dl, shift right, "\mathtt{dom}^m"'] \arrow[dl, shift left, "\mathtt{codom}^m"] \\ \mathtt{Type} & \\ & \mathtt{Term}^p \arrow[uu, "\mathtt{c}"'] \arrow[ul, shift right, "\mathtt{dom}^p"'] \arrow[ul, shift left, "\mathtt{codom}^p"] \end{tikzcd}$$

3.3 Zooms

In order to deal with the evolution of the state in section 3.2, we used three different logics: the equational logic L_{eq} where the state is hidden, the pointed equational logic L^*_{eq} for

showing explicitly the state, and the decorated equational logic L_{dec} as a kind of equational logic with additional information. These logics are related by morphisms, namely there is a span in the category **DiaLog**

$$\begin{tikzcd} & L_{\mathrm{dec}} \arrow[dl, "F_{\mathrm{far}}"'] \arrow[dr, "F_{\mathrm{near}}"] & \\ L_{\mathrm{eq}} & & L^*_{\mathrm{eq}} \end{tikzcd}$$

The morphism $F_{\mathrm{far}}\colon L_{\mathrm{dec}} \to L_{\mathrm{eq}}$ simply forgets the decorations: a modifier $f^m\colon X \to Y$ is mapped to $f\colon X \to Y$, as well as a pure operation $f^p\colon X \to Y$. This is the *far* view, where the distinction between pure operations and modifiers is blurred. The morphism $F_{\mathrm{near}}\colon L_{\mathrm{dec}} \to L^*_{\mathrm{eq}}$ provides the meaning of the decorations: a modifier $f^m\colon X \to Y$ becomes $f\colon S \times X \to S \times Y$ while a pure operation $f^p\colon X \to Y$ remains $f\colon X \to Y$. This is the *near* view, where the distinction between pure operations and modifiers is explicited. This span should be read from left to right: an equational specification Σ_{eq} is derived from the grammar of the language, then it is decorated as Σ_{dec} such that $F_{\mathrm{far}}\Sigma_{\mathrm{dec}} = \Sigma_{\mathrm{eq}}$, and finally the meaning of the decorations is provided by $\Sigma^*_{\mathrm{eq}} = F_{\mathrm{near}}\Sigma_{\mathrm{dec}}$. This span is called a *zoom*, because it goes from the far view to the near view.

The semantics of the programs is given by a model of Σ^*_{eq} in the pointed equational logic, with values in the theory $\mathbf{Set}_{\mathbb{S}}$ made of the equational theory of sets together with some fixed set of states $\mathbb{S}$ for interpreting the sort S. Thanks to property (1), equivalently the semantics of the programs is given by a model of Σ_{dec} in the decorated logic, with values in the decorated theory $U_{\mathrm{near}}\mathbf{Set}_{\mathbb{S}}$, where U_{near} is the right adjoint to F_{near}. But the semantics cannot be defined as a model of Σ_{eq}, which clearly does not bear enough details for providing a good semantics.

The construction of L_{dec} and F_{far} from L_{eq} is a kind of *opfibration*. Usually, given a functor $P\colon \mathbf{C} \to \mathbf{Set}$, the *category of elements* of P is the category $\mathit{Elt}(P)$ made of an object X^x for each object X in $\mathbf{C}$ and each element $x \in P(X)$, and with a morphism $f^x\colon (X, x) \to (Y, y)$ for each morphism $f\colon X \to Y$ in $\mathbf{C}$ and each element $x \in P(X)$, where $y = Pf(x)$. The functor $\varphi\colon \mathit{Elt}(P) \to \mathbf{C}$ which maps X^x to X and f^x to f is called an *opfibration*, it can be viewed as a $\mathbf{C}$-indexed family of sets. This construction, called the *Grothendieck construction*, can be generalized in several ways. For our purpose, it can be generalized to limit sketches and to logics, providing a systematic way for building new logics from well-known ones. Given a logic L_0 and a theory Θ_0 for this logic, the *logic of elements* of Θ_0 is a logic L_1 which can be seen as a variant of L_0 where every feature is indexed, or *decorated*, by some feature in Θ_0. There is a morphism of logics $\varphi\colon L_1 \to L_0$ which forgets the decorations.

For example, let Θ_0 be an equational theory generated by a unique sort D and two operations $p, m\colon D \to D$ such that $p \circ p = p$ and $p \circ m = m \circ p = m \circ m = m$. Then the logic of elements of Θ_0 looks like our decorated logic. Actually, in order to get the conversion from pure terms to modifiers in this way, we have to extend Θ_0 as a theory which is not set-valued. This does fit easily in our framework, because the realizations of a sketch need not be set-valued, but this is beyond the scope of this paper.

3.4 Sequential products

The zooming approach, as described in section 3.3, was used in [Dumas et al. 2009] for dealing with the issue of the order of evaluation of the arguments of a binary function (or more generally a n-ary function with $n \geq 2$). When there is no side effect, a term like $g(a_1, a_2)$ can be seen as the composition of the function g with the pair (a_1, a_2), which is formed from a_1 and a_2 using a cartesian product. For simplicity of the presentation, let $a_i = f_i(x_i)$ for some function $f_i\colon X_i \to Y_i$, for $i = 1, 2$, with $g\colon Y_1 \times Y_2 \to Z$. Then $(a_1, a_2) = (f_1 \times f_2)(x_1, x_2)$ and we focus on $f_1 \times f_2$. A cartesian product is a categorical product in the category of sets. The binary product on a category C is such that for all $f_1\colon X_1 \to Y_1$ and $f_2\colon X_2 \to Y_2$ there is a unique $f_1 \times f_2\colon X_1 \times X_2 \to Y_1 \times Y_2$ characterized by the following diagram, where the vertical morphisms are the projections

$$\begin{tikzcd}
X_1 \arrow[r, "f_1"] & Y_1 \\
X_1 \times X_2 \arrow[u] \arrow[r, "f_1 \times f_2"] \arrow[d] \arrow[ur, phantom, "="] \arrow[dr, phantom, "="] & Y_1 \times Y_2 \arrow[u] \arrow[d] \\
X_2 \arrow[r, "f_2"] & Y_2
\end{tikzcd}$$

When there are side effects, the effect and the result of evaluating $g(a_1, a_2)$ may depend on the order of evaluation of the arguments a_1 and a_2. This cannot be formalized by a cartesian product, which is a symmetric construction. Another construction is required, for formalizing the fact of evaluating first a_1 then a_2 (or the contrary). So, the main issue is about evaluating a_1 while keeping a_2 unchanged. For this purpose, in [Dumas et al. 2009] we go further into the decoration of equational specifications: the terms are decorated as pure or modifiers as in section 3.2, and in addition the equations themselves are decorated, as "true" equations (with symbol $=$) or *consistency* equations (with symbol $\sim$). On pure terms, both are interpreted as equalities. On modifiers, an equation $f^m = g^m\colon X \to Y$ becomes, through F_{near}, an equality $f = g\colon S \times X \to S \times Y$, whereas a consistency equation $f^m \sim g^m\colon X \to Y$ becomes an equality only on values (which is much weaker) $\text{pr} \circ f = \text{pr} \circ g\colon S \times X \to Y$, where $\text{pr}\colon S \times Y \to Y$ is the projection. Then the *right semi-pure* product $f_1 \rtimes \text{id}_{X_2}$ is characterized by the following diagram in the decorated logic, where the projections are pure

$$\begin{tikzcd}
X_1 \arrow[r, "f_1^m"] & Y_1 \\
X_1 \times X_2 \arrow[u] \arrow[r, "(f_1 \rtimes \text{id}_{X_2})^m"] \arrow[d] \arrow[ur, phantom, "="] \arrow[dr, phantom, "\sim"] & Y_1 \times X_2 \arrow[u] \arrow[d] \\
X_2 \arrow[r, "\text{id}_{X_2}^p"'] & X_2
\end{tikzcd}$$

The image of this diagram by the morphism F_{far} is such that the image of $(f_1 \rtimes \text{id}_{X_2})^m$ is $f_1 \times \text{id}_{X_2}$, which maps (s, x_1, x_2) to (s_1, y_1, x_2) where $(s_1, y_1) = f_1(s, x_1)$. Finally, the

composition of a right semi-pure product with a left semi-pure product gives rise to the required *left sequential product* $f_1 \ltimes f_2 = (\mathrm{id}_1 \ltimes f_2) \circ (f_1 \rtimes \mathrm{id}_2)$ for "first f_1 then f_2". In the pointed equational logic L^*_{eq}:

$$(f_1 \ltimes f_2)(s, x_1, x_2) = (s_2, y_1, y_2) \text{ where } (s_1, y_1) = f_1(s, x_1) \text{ and } (s_2, y_2) = f_2(s_1, x_2)$$

This is depicted below, first in the decorated logic L_{dec}, then through F_{near} in the pointed equational logic L^*_{eq}.

$$\begin{tikzcd}
X_1 \arrow[r, "f_1^m"] & Y_1 \arrow[r, "\mathrm{id}^p_{Y_1}"] & Y_1 \\
X_1 \times X_2 \arrow[u] \arrow[r, "(f_1 \rtimes \mathrm{id}_{X_2})^m"] \arrow[d] \arrow[ur, phantom, "="] & Y_1 \times X_2 \arrow[u] \arrow[r, "(\mathrm{id}_{Y_1} \ltimes f_2)^m"] \arrow[d] \arrow[ur, phantom, "\sim"] & Y_1 \times Y_2 \arrow[u] \arrow[d] \\
X_2 \arrow[r, "\mathrm{id}^p_{X_2}"] \arrow[ur, phantom, "\sim"] & X_2 \arrow[r, "f_2^m"] \arrow[ur, phantom, "="] & Y_2
\end{tikzcd}$$

$$\begin{tikzcd}
S \times X_1 \arrow[r, "f_1"] & S \times Y_1 & & Y_1 \arrow[r, "\mathrm{id}_{Y_1}"] & Y_1 \\
S \times X_1 \times X_2 \arrow[u] \arrow[r, "f_1 \times \mathrm{id}_{X_2}"] \arrow[d] \arrow[ur, phantom, "="] & S \times Y_1 \times X_2 \arrow[u] \arrow[d] \arrow[ur, phantom, "="] \arrow[r, equal] & S \times Y_1 \times X_2 \arrow[u] \arrow[d] \arrow[r, "\mathrm{id}_{Y_1} \times f_2"] \arrow[ur, phantom, "="] & S \times Y_1 \times Y_2 \arrow[u] \arrow[d] \arrow[ur, phantom, "="] & \\
X_2 \arrow[r, "\mathrm{id}_{X_2}"] & X_2 & S \times X_2 \arrow[r, "f_2"] & S \times Y_2 &
\end{tikzcd}$$

4 Conclusion

The framework of diagrammatic logics stems from issues about the semantics of computational effects. It is our hope that it may prove helpful for combining logics in other situations.

References

[Barr and Wells 1999] Barr, M. and Wells, C. (1999) Category Theory for Computing Science. *Centre de Recherches Mathématiques (CRM) Publications*, 3rd Edition.

[Domínguez and Duval 2009] Domínguez, C. and Duval, D. (2009) Diagrammatic logic applied to a parameterization process. `arXiv:0908.3737` (2009).

[Dumas et al. 2009] Dumas, J.G., Duval, D. and Reynaud, J.C. (2009) Cartesian effect categories are Freyd-categories. `arXiv:0903.3311`.

[Duval 2003] Duval, D. (2003) Diagrammatic specifications. *Mathematical Structures in Computer Science* **13** 857–890.

[Ehresmann 1968] Ehresmann, C. (1968) Esquisses et types de structures algébriques. *Bull. Instit. Polit. Iaşi* **XIV**.

[Gabriel and Zisman 1967] Gabriel, P. and Zisman, M. (1967) *Calculus of Fractions and Homotopy Theory*. Springer.

[Goguen and Burstall 1984] Goguen, J. A. and Burstall, R. M. (1984) *Introducing Institutions*. Springer Lecture Notes in Computer Science **164** 221–256.

[Lawvere 1963] Lawvere, F.W. (1963) Functorial Semantics of Algebraic Theories. *Proceedings of the National Academy of Science* **50** (5) 869-872.

[Lellahi 1989] Lellahi, S.K. (1989) Categorical abstract data type (CADT). *Diagrammes* **21**, SKL1-SKL23.

[Makkai 1997] Makkai, M. (1997) Generalized sketches as a framework for completeness theorems (I). *Journal of Pure and Applied Algebra* **115** 49–79.

[Moggi 1989] Moggi, E. (1989) Computational lambda-calculus and monads. *Logic in Computer Science (LICS '89)*, IEEE Computer Society 14–23.