From Categories with Families to Locally Cartesian Closed Categories

Pierre Clairambault

June 12, 2006

Contents

Abstract

After some necessary mathematical preliminaries about category theory, we describe some of the background in categorical logic, presenting cartesian closed categories, locally cartesian closed categories, then hyperdoctrines and P Dybjer's[4] categories with families. Then we focus on categories with families, and add some additional structure to them, until we can prove that they become actually locally cartesian closed.

1 Introduction

Formal systems and type theories are convenient and effective systems to studying programming languages and more generally, the notion of computation. Most of the times, we study these formal systems by writing down syntactic rules and syntactic equivalences to evaluate their relative power. Categorical logic provides some fresh air, since it brings with it abstract models that allow us to study known notions from a different point of view. Instead of taking care of syntactic issues (as in λ-calculus, where the question of the name of variables brought up issues, some solved by De Brujin indexes), we forget them and focus on structural properties, emphasized by the categorical setting.

The link between categorical notions and type theoretical notions has first been made by Lawvere, in the 1960ies. Cartesian closed categories appeared to be equivalent to simply typed λ-calculus, and over the time many categorical models were developed, modeling various type theoretic settings. Categories with families were introduced by P.Dybjer[4] to model a basic framework of dependent types. It provides an entirely variable-free setting of dependent type theory, proved by S.Mimram to be equivalent to the logical framework (**LF**), a dependently-typed λ-calculus.

In this paper, the approach will be quite different since we will not try to find a type theoretical equivalent to our categorical model. Instead, we're going to relate categories with families to other categorical models, particularly hyperdoctrines and locally cartesian closed categories. Indeed, their respective power is long known: hyperdoctrines are a model of predicate logic (see Seely[7]), and locally cartesian closed are exactly equivalent to a Martin-löf type theory presented for that purpose by Seely[8].

2 Mathematical preliminaries

Category theory provides a different (and more abstract) point of view to access known objects. In the case of type theory, Contexts will be only objects in a Category, and substitution operations will be denoted by abstract arrows. The following abstract definitions are necessary to define categories with families, which provide an abstract variable-free framework that has been proven by S.Mimram to be equivalent to the Logical Framework. Our hope is that such an abstract point of view emphasizes the fundamental properties of the studied objects, without taking into account details of the objects' structure, such as syntactical concerns.

The reader should be warned that the preliminaries presented here can in no way be considered as a complete introduction to category theory, but more as a reminder of some useful categorical definitions, focused on the needs of this paper. A complete introduction to categorical logic is the book by Lambek and Scott [2]. A more introductive book about category theory used for computer science is the book by Barr&Wells[3].

To express semantically precise notions of **ML** (*ie. Martin-löf's type theory*), we will need some elaborated categorical structures that will have to be understood.

2.1 Basic categorical elements

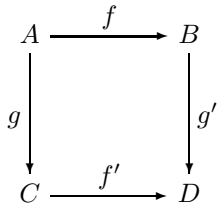
Definition. ***(Category)*** *A* ***Category C*** *is given by the following data:*

- *A collection of* objects *denoted by* $|\boldsymbol{C}|$
- *For* $A, B \in |\boldsymbol{C}|$*, a collection of* arrows *from* A *to* B*, denoted by* $Hom(A, B)$*. If* $f \in Hom(A, B)$*, then* A *is the* domain *of* f *and* B *is its* codomain*. We usually write* $f : A \to B$ *or* $A \xrightarrow{f} B$
- *For each objects* $A, B, C \in |\boldsymbol{C}|$*, for each* $f : A \to B$ *and* $g : B \to C$*, a* composite arrow $f \circ g : A \to C$*.*
- *For each obect* $A \in |\boldsymbol{C}|$*, an* identity arrow $id_A : A \to A$

such that the following rules are verified:

- *associativity of composition: for each* $A \xrightarrow{f} B \xrightarrow{g} C \xrightarrow{h} D$ *in* $\boldsymbol{C}$*,* $f \circ (g \circ h) = (f \circ g) \circ h$*.*
- *neutrality of the identity arrows: for each objects* A, B*, for each* $f \in Hom(A, B)$*,* $id_A \circ f = f \circ id_B$

While dealing with category theory, one often encounters complex algebrical equality statements involving many intricated compositions. Consequently, we often state them with *commutating diagrams*. For an example, the diagram



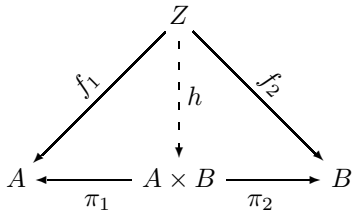
states that $g' \circ f = f' \circ g$

Example: The category **Sets** has sets as objects and arbitrary mappings as arrows.

Example: The category of *monoids* has monoids has objects, and multiplication-preserving morphisms as arrows.

Definition. ***(terminal object)*** *Let* $\boldsymbol{C}$ *be a category, let* T *be an object of* $\boldsymbol{C}$*.* T *is a* ***terminal object*** *of* $\boldsymbol{C}$ *if and only if for any object* X *of* $\boldsymbol{C}$*, there is an unique arrow* $X \to T$ *in* $\boldsymbol{C}$*.*

Definition. ***(binary products)*** *Let* $\boldsymbol{C}$ *be a category, let* A *and* B *be objects in this category. A* ***binary product*** *of* A *and* B *is an object* $A \times B$ *and two morphisms* π_1 *and* π_2 *such that for any object* Z*, for any morphisms* $f_1 : Z \to A$ *and* $f_2 : Z \to B$*, there is a unique morphism* h *such that the following diagram commutes:*



Definition. ***(functor)*** *Let* $\boldsymbol{B, C}$ *be two categories. A* ***functor*** F *from* $\boldsymbol{A}$ *to* $\boldsymbol{B}$ *is given by the following data:*

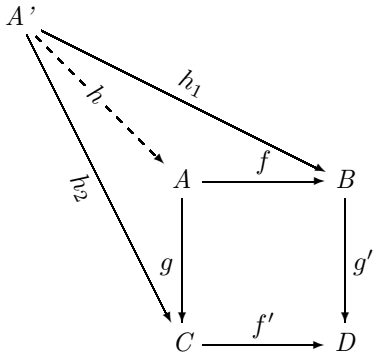
- *for each* $A \in |\boldsymbol{A}|$*, an image object* $F(A) \in |\boldsymbol{B}|$
- *for each* $A, B \in |\boldsymbol{A}|$*, for each* $A \xrightarrow{f} B \in Hom(A, B)$*, an image arrow* $F(A) \xrightarrow{F(f)} F(B) \in Hom(F(A), F(B))$*.*

such that:

- *for each* A*,* $\quad F(id_A) = id_{F(A)}$
- *for each* f, g*,* $\quad F(g \circ f) = F(g) \circ F(f)$

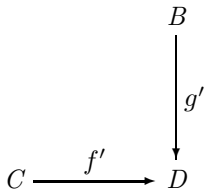
Note that we can define **Cat**, the category of categories, which has categories as objects and categorical structure preserving morphisms as arrows, *i.e., functors*. It is a general observation that many mathematical concepts arise as categorical concepts.

Definition. ***(pullback)*** *In a diagram*

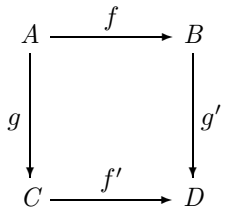


the inner square is said to be a ***pullback*** *if it commutes, and if for all* A' *with arrows* h_1 *and* h_2 *to* B *and* C *such that* $g' \circ h_1 = f' \circ h_2$*, there is an unique arrow* h *such that the two triangles commute. (*ie. $h_1 = f \circ h$ *and* $h_2 = g \circ h$*)*

Definition. ***(category with pullbacks)*** *Let* $\boldsymbol{C}$ *be a category.* $\boldsymbol{C}$ *is said to have* ***all pullbacks****, or to be a* ***category with pullbacks****, if and only if for any diagram*



in $\boldsymbol{C}$*, there is an object* A *and arrows* f, g *such that*



is a pullback diagram.

Definition. ***(adjointness)*** *Let* $\boldsymbol{A}$ *and* $\boldsymbol{B}$ *be two categories, and two functors* $F : \boldsymbol{A} \to \boldsymbol{B}$ *and* $G : \boldsymbol{B} \to \boldsymbol{A}$*. Then* F *is said to be a* ***left adjoint*** *to* G *(or equivalently,* G *is said to be a* ***right adjoint*** *to* F*), if and only if, for any* $A \in \boldsymbol{A}$ *and* $B \in \boldsymbol{B}$*, we have a bijection* $\Phi_{A,B} : Hom(F(A), B) \to Hom(A, G(B))$*. Moreover, this bijection has to be* natural *in* A *and* B*, meaning that for each* $A, A' \in |\boldsymbol{A}|$*, for each* $B, B' \in |\boldsymbol{B}|$*, for each* $f : A' \to A$ *and* $g : B \to B'$*, we get from any commutating diagram*

$$\begin{array}{ccc} F(A) & \xrightarrow{h} & B \\ {\scriptstyle F(f)}\uparrow & & \downarrow{\scriptstyle g} \\ F(A') & \xrightarrow[k]{} & B' \end{array}$$

another commutating diagram:

$$\begin{array}{ccc} A & \xrightarrow{\Phi_{A,B}(h)} & G(B) \\ {\scriptstyle f}\uparrow & & \downarrow{\scriptstyle G(g)} \\ A' & \xrightarrow[\Phi_{A',B'}(k)]{} & G(B') \end{array}$$

Notation: To express that $F : \mathbf{A} \to \mathbf{B}$ is left adjoint to $G : \mathbf{B} \to \mathbf{A}$, we often use the following notation:

$$\frac{F(C) \to D}{C \to G(D)}$$

where C represents any object of $\mathbf{A}$ and D any object of $\mathbf{B}$.

Exemple: let $\mathbf{A} = \mathbf{B}$ be the category of real numbers. An arrow from a to b is an inequality $a \leq b$. Then an adjointness is a couple (F, G) of functions $F : \mathbf{A} \to \mathbf{B}$ and $G : \mathbf{B} \to \mathbf{A}$ such that $\forall a \in \mathbf{A}, \forall b \in \mathbf{B}, F(a) \leq b \Leftrightarrow a \leq G(b)$.

Remark: The notion of adjointness is really important in this report, since it allows to express algebraically in the langage of categories many type theoretical elaborated notions, such as Π-types, Σ-types or equality types.

2.2 more elaborated categorical elements

Definition. ***(slice category)*** *If* $\boldsymbol{C}$ *is a category and* A *any object of* $\boldsymbol{C}$*, the* ***slice category*** $\boldsymbol{C}/A$ *is described this way:*

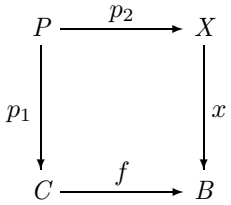
- *An object of* $\boldsymbol{C}/A$ *is an arrow* $f : C \to A$ *of* $\boldsymbol{C}$ *for some object* C*.*
- *An arrow of* $\boldsymbol{C}/A$ *from* $f : C \to A$ *to* $f' : C' \to A$ *is an arrow* $h : C \to C'$ *with the property that* $f = f' \circ h$*.*
- *The composite of* $h : f \to f'$ *and* $h' : f' \to f''$ *is* $h' \circ h$*.*

- *Given an object $f : C \to A$ of $\mathbf{C}/A$ (an arrow of $\mathbf{C}$), the identity arrow $id_f : f \to f$ is $id_C : C \to C$*

Remark: In categorical models of type theory, the concept of *fibration* often arise: a *fibration* with base category $\mathbf{C}$ is a mapping T from $|\mathbf{C}|$ to $\mathbf{Cat}$, the category of categories. For any object $A \in |\mathbf{C}|$, $T(A)$ is called the *fibre* over A. For an example linked to type theory, $\mathbf{C}$ could be a category of contexts, and $T(A)$, the fibre over A, would then be a category of types in this context. Now we can see the interest of slice categories : it allows to get a structure of fibration from any base category, which is necessary to establish correspondances between categorical models with fibrations, and simpler ones, without fibrations.

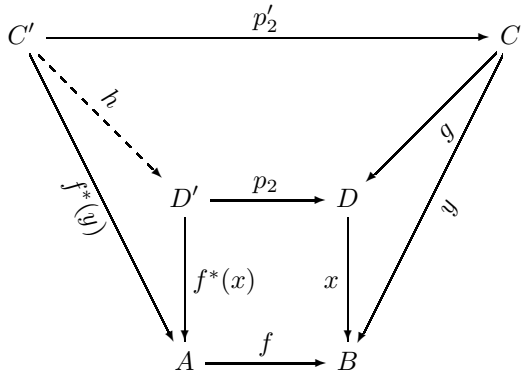
Definition. ***(pullback functor)*** *Let $\mathbf{C}$ be a category with pullbacks. Then for any arrow $f : A \to B$ of $\mathbf{C}$ we can get a functor $f^* : \mathbf{C}/B \to \mathbf{C}/A$ by the following observation:*

Image of objects: *Is $x : X \to B$ is an object of $\mathbf{C}/B$, then there is a pullback*



which allows us to state $f^*(x) = p_1 : P \to A$.

Image of arrows: *let $x : D \to B$ and $y : C \to B$ be objects of $\mathbf{C}/B$, and let $g : y \to x$ be an arrow in $\mathbf{C}/B$. Then we get the following diagram, where both the small and the bigger squares are pullbacks:*



since $C' \xrightarrow{f^*(y)} A$ and $C' \xrightarrow{p'_2} C \xrightarrow{g} D$ we get an arrow $h : C' \to D'$, and we state $f^*(g) = h$.

3 Models of Type Theory

3.1 Locally cartesian closed categories

Definition. ***(cartesian closed category)*** *Let $\mathbf{C}$ be a category. $\mathbf{C}$ is **cartesian closed** if and only if*

- *$\mathbf{C}$ has a terminal object.*
- *any two objects X and Y of $\mathbf{C}$ have a product $X \times Y$ in $\mathbf{C}$.*

- *for every object Y in $\mathbf{C}$, the functor $X \to X \times Y$ from $\mathbf{C}$ to $\mathbf{C}$ has a right adjoint.*

Remark: The above right adjoint condition is equivalent to the existence of an exponent object and an evaluation morphism. This provides a structures really near simply-typed lambda calculus. Indeed, cartesian closed categories are a model for simply typed lambda calculus.

Definition. ***(locally cartesian closed category)*** *Let $\mathbf{C}$ be a category. $\mathbf{C}$ is said to be **locally cartesian closed** if one of the equivalent properties is verified:*

- *$\mathbf{C}$ has all pullbacks and for each arrow $f : A \to B$, the pullback functor $f^* : \mathbf{C}/B \to \mathbf{C}/A$ has a right adjoint.*
- *For every object A of $\mathbf{C}$, the slice category $\mathbf{C}/A$ is cartesian closed.*

Remark: The second version of the definition is helpfull to get the idea of the power of locally cartesian closed categories. For any object A, the slice category $\mathbf{C}/A$ is cartesian closed, and so, is equivalent to a simply typed lambda calculus. This seems to add some kind of dependance over the objects of the base category $\mathbf{C}$, which could for an example be closed types. Indeed it is proved in [8] that locally cartesian closed categories are equivalent to extensional Martin-löf type theories (without inductive types).

3.2 hyperdoctrines

Then concept of *fibration*, discussed above, is of great importance when speaking about type theory, because types, possibly with free variables, are usually indexed by contexts, and terms can be as well indexed by types. Therefore, it turns out to be of interest to build models with fibrations included from the beginning, without using the concept of a slice category. One of theses models with fibrations is the hyperdoctrine.

Definition. ***(hyperdoctrines)*** *A $\mathbf{C}$-indexed category $\mathbf{P}$ (a category $\mathbf{C}$ together with a functor $T : \mathbf{C} \to \mathbf{Cat}$) is a hyperdoctrine if and only if*

- *for each object A of $\mathbf{C}$, $\mathbf{P}(A)$ is cartesian closed.*
- *for each $f : A \to B$ of $\mathbf{C}$, f^* preserve exponents.*
- *for each $f : A \to B$ of $\mathbf{C}$, f^* has a left adjoint Σ_f and a right adjoint Π_f.*
- *The Beck condition : if*

$$\begin{array}{ccc} D & \xrightarrow{h} & C \\ {\scriptstyle k}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & B \end{array}$$

is a pullback in $\mathbf{C}$, then for any object φ of $\mathbf{P}(C)$, $\Sigma_k h^* \varphi \to f^* \Sigma_g \varphi$ is an isomorphism in $\mathbf{P}(A)$.

Remark: hyperdoctrines, defined as described above, are a model of predicate logic.

Proposition. *Let $\mathbf{C}$ be a category. If $\mathbf{C}$ has all pullbacks and a terminal object , then $\mathbf{C}$ is locally cartesian closed if and only if as a $\mathbf{C}$-indexed category, $\mathbf{C}$ is a hyperdoctrine.*

The proof can be found in [5].

3.3 Categories with families

Categories with families are a model of type theory with dependent types, which is closely related to Cartmell's *categories with attributes*. The structure of categories with families is somehow near the hyperdoctrines, with some kind of fibration, but is one step nearer the syntax of dependent types, which it aims to model.

Definition. ***(Fam)*** *Let* ***Fam*** *be the category of families of sets. It is defined as follows:*

- *its objects are families of sets* $(B(x))_{x \in A}$
- *a morphism from* $(B(x))_{x \in A}$ *to* $(B'(x'))_{x' \in A'}$ *is a pair consisting of a function* $f : A \to A'$ *and a family of functions* $g(x) : B(x) \to B'(f(x))$ *indexed by* $x \in A$*.*

Definition. ***(category with families (cwf))*** *A* ***category with families*** *consists of the four following parts:*

- *A base category* $\boldsymbol{C}$*. Its objects are called* contexts *and its morphisms are called* substitutions*.*
- *A* functor $T : \boldsymbol{C}^{op} \to \boldsymbol{Fam}$*. We write* $T(\Gamma) = (\Gamma \vdash A)_{A \in Type(\Gamma)}$*, where* Γ *is an object of* $\boldsymbol{C}$*, and call it the family of* terms *indexed by* types *in context* Γ*. Moreover, if* γ *is a morphism of* $\boldsymbol{C}$ *then the two components of* $T(\gamma)$ *interpret substitution in types and terms respectively. We write* $A[\gamma]$ *for the application of the first component to a type* A *and* $a[\gamma]$ *for the application of the second component to a term* a*.*
- *A* terminal object $\Diamond$ *of* $\boldsymbol{C}$ *called the empty context.*
- *A* context comprehension *operation which to an object* Γ *of* $\boldsymbol{C}$ *and a type* $A \in Type(\Gamma)$ *associates an object* $\Gamma; A$ *of* $\boldsymbol{C}$*, a morphism* $p : \Gamma; A \to \Gamma$ *of* $\boldsymbol{C}$ *(the* first projection*), and a term* $q \in \Gamma; A \vdash A[p]$ *(the* second projection*). The following universal property holds : for each object* Δ *in* $\boldsymbol{C}$*, morphism* $\gamma : \Delta \to \Gamma$*, and term* $a \in \Delta \vdash A[\gamma]$*, there is a unique morphism* $\theta =< \gamma, a >: \Delta \to \Gamma; A$*, such that* $p \circ \theta = \gamma$ *and* $q[\theta] = a$*.*

Remark: *(about substitutions)* The substitutions defined here do model the usual ones, summarized in [1]. Let's see a context as a list of types and a substitution as a list of terms. For an example, a term in the context $x_1 : \sigma_1, ..., x_n : \sigma_n$ is a term with $x_1, ..., x_n$ among the free variables. If $\gamma = (M_1, ..., M_n)$ is a substitution, we say that $\gamma : \Gamma \to \Delta$ if the following n judgements holds:

$$\begin{array}{lll} \Gamma & \vdash & M_1 : \sigma_1 \\ \Gamma & \vdash & M_2 : \sigma_2[M_1/x_1] \\ & & ... \\ \Gamma & \vdash & M_n : \sigma_n[M_1/x_1][M_2/x_2]...[M_{n-1}/x_{n-1}] \end{array}$$

then, if $a : \Delta \vdash A$, $a[\gamma]$ will be a where every occurence of x_i is replaced by M_i. We do get a term $a[\gamma] : \Gamma \vdash A[\gamma]$. This explains why a substitution $\gamma : \Gamma \to \Delta$ is interpreted by a function from the family over Δ to the family over Γ, and why the functor T is from $\mathbf{C}^{op}$, which is $\mathbf{C}$ where all the arrows are taken the opposite way.

Remark: the projections p and q are given for each context construction $\Gamma; A$, so they should be rather called p^{Γ}_{A} and q^{Γ}_{A}. However, these annotations will often be omitted, for an example within long syntactic proofs where they would be unconvenient. In fact, they can be recovered from the context : if we have a substitution p from $\Gamma; A$ to Γ, we know that $p = p^{\Gamma}_{A}$.

Categories with families are inspired by Cartmell's *categories with attributes*. The main difference between these two contructions is that categories with attributes are less connected to the syntax.

Such notions as types, terms or substitutions are not natively present in categories with attributes. However, they can be recovered from the categorical structure which is almost the same in categories with families. Indeed, it turns out that categories with families and categories with attributes are equivalent. Many built-in notions of categories with families are in one-to-one correspondance with other notions that fall down directly from the core categorical structure. We're going to summurize them quickly, since it's usefull to have that in mind when thinking about cwfs.

Types and first projections (or *display maps*) : *There is a one-to-one correspondance between types and first projections.* Given a first projection p_A^Γ, the type $A \in Type(\Gamma)$ can be recovered directly. Given a type $A \in Type(\Gamma)$, we get directly the first projection p_A^Γ.

Terms and sections of display maps : *There is a one-to-one correspondance between terms and sections of display maps.* A *section* of a display map $p_A^\Gamma : \Gamma; A \to \Gamma$ is a morphism $t : \Gamma \to \Gamma; A$ such that $p \circ t = id_{A[p]}$. From such a section we can get the term $a = q[t] : \Gamma \vdash A$, and from a term a we get the section $< id, a >$ of the display map p_A^Γ.

Substitutions and pullbacks : Let Γ, Δ be contexts, $\gamma : \Delta \to \Gamma$ be a substitution, and $A \in Type(\Gamma)$. Then the following diagram is a pullback:

$$\begin{array}{ccc} \Delta; A[\gamma] & \xrightarrow{< \gamma \circ p, q >} & \Gamma; A \\ \Big\downarrow{\scriptstyle p} & & \Big\downarrow{\scriptstyle p} \\ \Delta & \xrightarrow[\gamma]{} & \Gamma \end{array}$$

The proof is quite elementary and can be found at many places, in Hofmann's chapter of [1] or in [9].

We now know how categories with families relate to other categorical models of dependent types, but what is their real expression power? Has it any type theoretic equivalent? S.Mimram answered to this question last year with the following result:

Proposition. *Let* ***LF*** *denote the* Logical Framework*, which is a type theory with* Π*-types (dependent function space) and a universe (a set containing codes for types). Let* $\boldsymbol{Cwf_{LF}}$ *denote the theory of categories with families with constants added for* Π*-types and a universe. Then* ***LF*** *is equivalent to* $\boldsymbol{Cwf_{LF}}$*.*

Remark: The above proposition will remain totally informal in this paper, however the reader is advised to have a look at [9], where all this is developed and proved.

4 Additional structure of categories with families

Now that different categorical models of type theory have been presented, we're going to try to relate categories with families to locally cartesian closed categories. Since locally cartesian closed categories are a model of **ML** type theory, with in particular Π and Σ type formers, we know that we'll have to add some structure to cwfs. In particular, we need Π-types and Σ-types, and they have to arise as adjoints to any substitution functor, in order to match with the Π and Σ notions in locally cartesian closed categories.

There are two reasons why it is interesting to prove adjointness properties for Π and Σ types. First, because to prove that categories with families with our extra structure are locally cartesian closed, we will need to have adjoints to substitution functors, simply because any locally cartesian closed

category has such adjoints. The second reason is the fact that anyway it allows to characterize an almost syntactic notion, defined as a constant in cwfs, by an abstract categorical notion.

Π-types and Σ-types are indeed relevant to provide our adjoints : the Π formation rule in a cwf defined as an generalized algebraic theory (an abstract framework to define other theories) would be expressed by

$$\frac{\Gamma : Context \quad A : Type(\Gamma) \quad B : Type(\Gamma; A) \quad b : \Gamma; A \vdash B}{\lambda(b) : \Gamma \vdash \Pi(A, B)}$$

We cheat a little bit with the syntax to get the idea. P_A is the functor associated to the substitution p_A^Γ and Π_A is the functor which to any type B associates $\Pi(A, B)$. Both will be defined formally in subsections 4.1 and 4.2. We get:

$$\frac{P_A(\Gamma) \vdash B}{\Gamma \vdash \Pi_A(B)}$$

which is almost a adjointness statement.

4.1 preliminary work

Basically, contexts are just objects in the base category $\mathbf{C}$. In order to express Π-types and Σ-types as adjoints to the substitution functor, we need to define the so-called substitution functor: we need to map contexts to categories of types in this context, and substitutions to functors between these categories.

Definition. ***(categories of types)*** *Let us consider a category with families with base category* $\boldsymbol{C}$ *and functor* T*. Given a context* Γ *in the base category* $\boldsymbol{C}$ *, we define* $Type_{\boldsymbol{C}}(\Gamma)$ *the category of types in context* Γ*:*

- ***Objects:*** *The set of objects of* $Type_{\boldsymbol{C}}(\Gamma)$ *will be the index set of* $T(C)$*, semantically the set of types in context* Γ*, plus a terminal object* T*.*
- ***Arrows:*** *We need to define arrows between* $A, B \in Type(\Gamma)$*:*

$$f \in Type_{\boldsymbol{C}}(\Gamma)(A \to B) \Leftrightarrow f : \Gamma; A \vdash B[p_A]$$

We must check that the definition above does define a category. The following equational reasoning is justified by the formalisation of Categories with Families as a Generalized Algebraic Theory, exposed in [4].

- Composition: if $f : A \to B$ ($f : \Gamma; A \vdash B[p_A]$) and $g : B \to C$ ($g : \Gamma; B \vdash C[p_B]$), then:

$$g \circ f = g[< p_A, f >] : \Gamma; A \vdash C[p_A]$$

- Identity: Let A be a type in Γ. Then:

$$id_A = q_A : \Gamma; A \vdash A[p_A]$$

- Associativity of composition: Let $f : A \to B$, $g : B \to C$ and $h : C \to D$ be arrows in Γ.

$$\begin{array}{rcl} h \circ (g \circ f) & = & h \circ (g[< p_A, f >]) \\ & = & h[< p_A, g[< p_A, f >] >] \end{array}$$

$$\begin{array}{rcl} (h \circ g) \circ f & = & (h[< p_B, g >]) \circ f \\ & = & h[< p_B, g > \circ < p_A, f >] \\ & = & h[< p_A, g[< p_A, f >] >] \end{array}$$

- Neutrality of the identity arrow: Let $f : A \to B$, then

$$f \circ id_A = f[< p_A, q_A >] = f$$

$$id_B \circ f = id_B[< p_A, f >] = q_B[< p_A, f >] = f$$

Definition. ***(the substitution functor)*** *Let be a Category with Families with* $\mathbf{C}$ *as base category. let* Γ *be an object of* $\mathbf{C}$ *and* $A \in Type(\Gamma)$*. Then we define the* ***substitution functor*** P_A *as following:*

- ***Image of objects:*** *Let* $B \in Type(\Gamma)$*, then* $P_A(B) = B[p_A] \in Type(\Gamma; A)$
- ***Image of arrows:*** *Let* $B, C \in Type(\Gamma)$*, and* $f : B \to C$*, then*

$$P_A(f) = f[< p_A \circ p_{B[p]}, q_{B[p]} >] : \Gamma; A; B[p] \vdash C[p \circ p]$$

  and we do have the expected property $P_A(f) : P_A(B) \to P_A(C)$*.*

- ***Preservation of identity:***

$$P_A(id_B) = P_A(q_B) = q_B[< p_A \circ p_{B[p]}, q_{B[p]} >] = q_{B[p]} = id_{B[p]}$$

- ***Preservation of composition:*** *Let* $B, C, D \in Type(\Gamma)$ *and* $f : B \to C, g : C \to D$*, then*

$$P_A(f \circ g) = P_A(f[< p, g >]) = f[< p, g >][< p \circ p, q >] = f[< p \circ p, q >][< p, g[< p \circ p, q >] >] = P_A(f) \circ P_A(g)$$

4.2 Simple Π-types

The goal of this subsection is to define functors associated with display maps, and to show that these functors are right adjoints to display maps. First, let us present the chosen structural definition of Π-types, extracted from [4]. The rules are presented in the formal syntax of a **gat** (generalized algebraic theory).

Operator symbols:

$$\frac{\Gamma : Context \quad A : Type(\Gamma) \quad B : Type(\Gamma; A)}{\Pi(A, B) : Type(\Gamma)}$$

$$\frac{\Gamma : Context \quad A : Type(\Gamma) \quad B : Type(\Gamma; A) \quad b : \Gamma; A \vdash B}{\lambda(b) : \Gamma \vdash \Pi(A, B)}$$

$$\frac{\Gamma : Context \quad A : Type(\Gamma) \quad B : Type(\Gamma; A) \quad c : \Gamma \vdash \Pi(A, B) \quad a : \Gamma \vdash A}{app(c, a) : \Gamma \vdash B[< id, a >]}$$

Equations:

$$\left.\begin{array}{rcl} \Pi(A, B)[\gamma] & = & \Pi(A[\gamma], B[< \Gamma \circ p, q >]) \\ \lambda(b)[\gamma] & = & \lambda(b[< \gamma \circ p, q >]) \end{array}\right\} \textit{substitution laws}$$

$$\left.\begin{array}{rcl} app(c, a)[\gamma] & = & app(c[\gamma], a[\gamma]) \\ app(\lambda(b), a) & = & b[< id, a >] \end{array}\right\} \quad (\beta)$$

$$\lambda(app(c[p], q)) = c \qquad (\eta)$$

Next step is to define the Π functor, which is to be right adjoint to a substitution functor.

Definition. ***(the*** Π ***functor)*** *We define, for* $C \in Type(\Gamma)$*, a functor* $\Pi_C : Type_{\mathbf{C}}(\Gamma; C) \to Type_{\mathbf{C}}(\Gamma)$*:*

- ***Image of objects:*** *Let* $A \in Type(\Gamma; C)$*, then* $\Pi_C(A) = \Pi(C, A)$*.*

- ***Image of arrows:*** *Let* $A, B \in Type(\Gamma; C)$ *and* $f : A \to B$*, then*

$$\Pi_C(f) = \lambda(f[<< p \circ p, q >, app(q[p], q)]) : \Gamma; \Pi(C, A) \vdash \Pi(C, B)[p]$$

and we do have the expected property $\Pi_C(f) : \Pi_C(A) \to \Pi_C(B)$*.*

- ***Preservation of identity:***

$$\Pi_C(id_A) = \Pi_C(q_A) = \lambda(q[<< p \circ p, q >, app(q[p], q) >]) = \lambda(app(q[p], q)) = q = id_{\Pi_C(A)}$$

- ***Preservation of composition:***

$$\begin{array}{rcl}
\Pi_C(f \circ g) & = & \Pi_C(f[< p, g >]) \\
 & = & \lambda(f[< p, g >][<< p \circ p, q >, app(q[p], q) >]) \\
 & = & \lambda(f[<< p \circ p, q >, g[<< p \circ p, q >, app(q[p], q)] >]) \\
\Pi_C(f) \circ \Pi_C(g) & = & \lambda(f[<< p \circ p, q >, app(q[p], q) >][< p, g[< p \circ p, q, app(q[p], q) >] >] \\
 & = & \lambda(f[<< p \circ p, q >, app(g[<< p \circ p, q >, app(q[p], q) >][p], q) >]) \\
 & = & \lambda(f[<< p \circ p, q >, g[<< p \circ p, q >, app(q[p], q) >] >])
\end{array}$$

Proposition. *The functor* Π_C *is right adjoint to the substitution functor* P_C*.*

Proof: We have to prove, for Γ any context, $A, B \in Type(\Gamma)$ and $C \in Type(\Gamma; A)$ the following adjunction:

$$\frac{P_A(B) \to C}{B \to \Pi_A(C)}$$

We need to find a bijection between terms in $\Gamma; A; B[p] \vdash C[p]$ and $\Gamma; B \vdash \Pi(A, C)[p]$.
Let $f : \Gamma; A; B[p] \vdash C[p]$, we define $F_1(f) = \lambda(f[<< p \circ p, q >, q[p] >]) : \Gamma; B \vdash \Pi(A, C)[p]$.
Let $g : \Gamma; B \vdash \Pi(A, C)[p]$, we define $F_2(g) = app(f[p], q)[<< p \circ p, q >, q[p] >] : \Gamma; A; B[p] \vdash C[p]$.
Then:

$$\begin{array}{rcl}
F_1(F_2(f)) & = & F_1(app(f[p], q)[<< p \circ p, q >, q[p] >] \\
 & = & \lambda(app(f[p], q)[<< p \circ p, q >, q[p] >][<< p \circ p, q >, q[p] >]) \\
 & = & \lambda(app(f[p], q) \\
 & = & f \\
F_2(F_1(f)) & = & F_2(\lambda(f[<< p \circ p, q >, q[p] >])) \\
 & = & app(\lambda(f[<< p \circ p, q >, q[p] >])[p], q)[<< p \circ p, q >, q[p] >] \\
 & = & app(\lambda(f[<< p \circ p, q >, q[p] >])[< p \circ p, q >], q[p]) \\
 & = & app(\lambda(f[<< p \circ p \circ p, q >, q[p] >]), q[p]) \\
 & = & f[<< p \circ p \circ p, q >, q[p] >][< id, q[p] >] \\
 & = & f[<< p \circ p, q[p] >, q >] \\
 & = & f
\end{array}$$

Now, we have still to check the naturality condition, *i.e.* to check that given arrows $h_1 : \Pi_A(B) \to C, h_2 : \Pi_A(B') \to C', f : B' \to B, g : C \to C'$, $g \circ h_1 \circ \Pi_A(f) = h_2$ implies $\Pi_A(g) \circ F_1(h_1) \circ f = F_1(h_2)$ (refer to the definition of adjoints, section 2.1). This verification is easy (but laborious), following the equational definitions of the terms.

4.3 simple Σ-types

First, the formal rules to define Σ-types in categories with families:

Operator symbols:

$$\frac{\Gamma : Context \quad A : Type(\Gamma) \quad B : Type(\Gamma; A)}{\Sigma(A,B) : Type(\Gamma)}$$

$$\frac{\Gamma : Context \quad A : Type(\Gamma) \quad B : Type(\Gamma; A) \quad a : \Gamma \vdash A \quad b : \Gamma \vdash B[< id, a >]}{Pair(a,b) : \Gamma \vdash \Sigma(A,B)}$$

$$\frac{\Gamma : Context \quad A : Type(\Gamma) \quad B : Type(\Gamma; A) \quad a : \Gamma \vdash \Sigma(A,B)}{p_1(a) : \Gamma \vdash A}$$

$$\frac{\Gamma : Context \quad A : Type(\Gamma) \quad B : Type(\Gamma; A) \quad a : \Gamma \vdash \Sigma(A,B)}{p_2(a) : \Gamma \vdash B[< id, p_1(a) >]}$$

Equations:

$$\left.\begin{array}{rcl} \Sigma(A,B)[\gamma] & = & \Sigma(A[\gamma], B[< \gamma \circ p, q >]) \\ Pair(a,b)[\gamma] & = & Pair(a[\gamma], b[\gamma]) \\ p_1(c)[\gamma] & = & p_1(c[\gamma]) \\ p_2(c)[\gamma] & = & p_2(c[\gamma]) \end{array}\right\} \textit{substitution laws}$$

$$Pair(p_1(a), p_2(a)) = a \quad \textit{surjective pairing}$$

$$\left.\begin{array}{rcl} p_1(Pair(a,b)) & = & a \\ p_2(Pair(a,b)) & = & b \end{array}\right\} \textit{projection laws}$$

Definition. ***(the Σ functor)*** *We define, for $C \in Type(\Gamma)$, a functor $\Sigma_C : Type_{\mathbf{C}}(\Gamma; C) \to Type_{\mathbf{C}}(\Gamma)$:*

- ***Image of objects:*** *Let $A \in Type(\Gamma; C)$, then $\Sigma_C(A) = \Sigma(C, A)$*
- ***Image of arrows:*** *Let $A, B \in Type(\Gamma; C)$, let $f : A \to B$ ($f : \Gamma; C; A \vdash B[p]$), then*

$$\Sigma_C(f) = Pair(p_1(q), f[<< p, p_1(q) >, p_2(q) >]) : \Gamma; \Sigma(C,A) \vdash \Sigma(C,B)[p]$$

  and we do have the expected property $\Sigma_C(f) : \Sigma_C(A) \to \Sigma_C(B)$.

- ***Preservation of Identities:***

$$\Sigma_C(id_A) = \Sigma_C(q_A) = Pair(p_1(q), q[<< p, p_1(q) >, p_2(q) >]) = Pair(p_1(q), p_2(q)) = q = id_{\Sigma_C(A)}$$

- ***Preservation of composition:***

$$\begin{array}{rcl} \Sigma_C(f \circ g) & = & \Sigma_C(f[< p, g >]) \\ & = & Pair(p_1(q), f[< p, g >][<< p, p_1(q) >, p_2(q) >]) \\ & = & Pair(p_1(q), f[<< p, p_1(q) >, g[<< p, p_1(q) >, p_2(q) >] >]) \\ \Sigma_C(f) \circ \Sigma_C(g) & = & \Sigma_C(f)[< p, \Sigma_C(g) >] \\ & = & Pair(p_1(q), f[<< p, p_1(q) >, p_2(q) >])[< p, Pair(p_1(q), g[<< p, p_1(q) >, p_2(q) >]) >] \\ & = & Pair(p_1(q), f[<< p, p_1(q) >, g[<< p, p_1(q) >, p_2(q) >] >]) \end{array}$$

Proposition. *The functor Σ_C is left adjoint to the substitution functor P_C.*

Proof: We have to prove, for any context Γ, for any $A, C \in Type(\Gamma)$, $B \in Type(\Gamma; A)$, the following adjunction:

$$\frac{\Sigma_A(B) \to C}{B \to P_A(C)}$$

This is equivalent to finding a bijection between terms in $\Gamma; \Sigma(A, B) \vdash C[p]$ and $\Gamma; A; B \vdash C[p \circ p]$
Let $f : \Gamma; \Sigma(A, B) \vdash C[p]$, we define $F_1(f) = f[< p \circ p, Pair(q[p], q) >] : \Gamma; A; B \vdash C[p \circ p]$
Let $f : \Gamma; A; B \vdash C[p \circ p]$, we define $F_2(f) = f[<< p, p_1(q) >, p_2(q) >] : \Gamma; \Sigma(A, B) \vdash C[p]$

$$\begin{array}{rcl} F_1(F_2(f)) & = & f[<< p, p_1(q) >, p_2(q) >][< p \circ p, Pair(q[p], q) >] \\ & = & f[<< p \circ p, q[p] >, q >] \\ & = & f \\ F_2(F_1(f)) & = & f[< p \circ p, Pair(q[p], q) >][<< p, p_1(q) >, p_2(q) >] \\ & = & f[< p, Pair(p_1(q), p_2(q)) >] \\ & = & f[< p, q >] \\ & = & f \end{array}$$

Now, we have still to check the naturality condition, *i.e.* to check that given arrows $h_1 : \Sigma_A(B) \to C, h_2 : \Sigma_A(B') \to C', f : B' \to B, g : C \to C'$, $g \circ h_1 \circ \Sigma_A(f) = h_2$ implies $\Pi_A(g) \circ F_1(h_1) \circ f = F_1(h_2)$. As for the Π case, the proof is easy following the equational definitions, but remains laborious.

4.4 Identity types, extensionnal identity

In this subsection, we will define identity types in categories with families, first intensional and then extensional. The expression chosen of intensional identity types is mainly from [1].

Definition. ***(intensional identity types)*** *Identity types are defined by, for each type $A \in Type(\Gamma)$, the following data:*

- *a type $Id_A \in Type(\Gamma; A; A[p])$*
- *a morphism $Refl_A : \Gamma; A \to \Gamma; A; A[p]; Id_A$ such that $p \circ Refl =< id, q >: \Gamma; A \to \Gamma; A; A[p]$ the diagonal morphism.*
- *for each $B \in Type(\Gamma; A; A[p]; Id_A)$, a function $R^{Id} \in Tm(B[Refl_A]) \to Tm(B)$, where $Tm(X)$ is the set of terms in type X.*

in such a way that these data are stable under substitution with respect to morphisms $f : \Delta \to \Gamma$ and such that whenever $B \in Type(\Gamma; A; A[p]; Id_A)$ and $H \in Tm(B[Refl_A])$ then $R^{Id}(H)[Refl_A] = H$.

Remark: the type Id_A is given for each $A \in Type(\Gamma)$, and is not syntactically equal to any of the other Id_x. However, we can have a bijection of the form $Id_{A[p]} \cong Id_A[<< p \circ p \circ p >, q[p], q >]$ which may allow us to substitute later, since all will be defined up to an isomorphism. The reason of this is the following: a term of type Id_A, in context $\Gamma; A; A[p]$, is intuitively a proof that the two elements of type A in the context are equal. Thus, a term of type $\Gamma; A; A[p]; A[p \circ p] \vdash Id_{A[p]}$ is a proof that the two last occurences of A in the context are equal. Then $\Gamma; A; A[p]; A[p \circ p] \vdash Id_A[<< p \circ p \circ p >, q[p], q >]$ plays the same role, since terms of this type are proofs that the two occurences of A are equal in the context $\Gamma; A; A[p]; A[p \circ p]$ where the first occurence of A has been forgotten.

Extensional identity types: We add the following rule, which allows us to substitute two terms that are intensionally equal.

$$\frac{c : \Gamma \vdash Id_A[<< id, a >, b >]}{a = b : \Gamma \vdash A}$$

4.5 Mappings between contexts and types (or *context products*)

As the reader will have noticed, the Π and Σ types presented here are very similar to their syntactic origin. This has a flaw: they allow to quantify over single terms only. Their associated functors are from $Type_{\mathbf{C}}(\Gamma; A)$ to $Type_{\mathbf{C}}(\Gamma)$ (with Γ a given context and A a type in this context). Therefore, they can only be adjoints to display maps, which induce functors the other way, from $Type_{\mathbf{C}}(\Gamma)$ to $Type_{\mathbf{C}}(\Gamma; A)$. To reach the structure of Locally Cartesian Closed Categories, we need adjoints to functors from $Type_{\mathbf{C}}(\Delta)$ to $Type_{\mathbf{C}}(\Gamma)$, for any contexts Γ and Δ. Current Π and Σ are clearely unsufficient to play this role, indeed we will need more general Π and Σ types, able to quantify over more than single types. As we have seen in section 3.3, types are equivalent to display maps. Quantifying over single types is quantifying over display maps; the generality we want force us to quantify over *any* substitution, and not only display maps.

To solve this problem, we add constants to make the link between single terms and substitutions. Intuitively, this is equivalent to allowing unbound finite products, since we turn substitutions (lists of terms) into single terms.

Rules:

$$\frac{\Gamma : Context}{\wedge(\Gamma) : Type(\Diamond)}$$

$$\frac{a : \Diamond; \wedge(\Delta) \vdash \wedge(\Gamma)[p]}{S(a) : \Delta \to \Gamma}$$

$$\frac{\gamma : \Delta \to \Gamma}{T(\gamma) : \Diamond; \wedge(\Delta) \vdash \wedge(\Gamma)[p]}$$

Equations:

$$\begin{array}{rcl} T(id) & = & q \\ S(a \circ' b) & = & S(a) \circ S(b) \\ T(\delta \circ \gamma) & = & T(\delta) \circ' T(\gamma) \\ \wedge(\Diamond; A; B) & = & \Sigma(A, B) \textit{ in a cwf with simple } \Sigma\textit{-types} \end{array}$$

Here, $\circ$ is the usual composition of substitutions, and $\circ'$ the composition of terms, *i.e.* the composition of arrows in $Type_{\mathbf{C}}(\Diamond)$.

4.6 Generalized Π-types

The rules presented here are syntactically very complicated, therefore some annotations will be omitted when writing them down. Once more, the **gat**-like presentation is used.

Rules:

$$\frac{\delta : \Gamma' \to \Gamma \quad b : \Gamma' \vdash B}{\lambda_\delta(b) : \Gamma \vdash \Pi_\delta(B)}$$

$$\frac{\delta : \Gamma' \to \Gamma \quad \delta' : \Gamma \to \Gamma' \quad c : \Gamma \vdash \Pi_\delta(B) \quad t : \Diamond; \wedge(\Gamma') \vdash Id_{\wedge(\Gamma')[p]}[<< id, T(id) >, T(\delta \circ \delta') >]}{app(c, \delta') : \Gamma \vdash B[\delta']}$$

The rules may be difficult to read and interpret, but in principle they are simple: instead of quantifying over a single type A, we quantify over a substitution δ. We generalize the fact that types are equivalent to display maps and that terms are equivalents to sections of display maps. Here we take any substitution δ as type, and a term of that type is a substitution δ' such that $\delta \circ \delta' = id$.

Operations: These equations are probably none the better possible, however they're sufficient to prove our result.

$$\begin{array}{rcl} \Pi_\delta(C)[\delta] & = & \Pi_{id}(C) = C \\ app(\lambda_\delta(f)[\delta], id) & = & f \\ \lambda_\delta(app(f[\delta], id)) & = & f \end{array}$$

Substitution under Π: Moreover, we have the (potentially infinite) type substitution equations given by the following rule. If the following diagram is a pullback:

$$\begin{array}{ccc} \Delta' & \xrightarrow{\gamma'} & \Gamma' \\ {\scriptstyle \delta'}\downarrow & & \downarrow{\scriptstyle \delta} \\ \Delta & \xrightarrow[\gamma]{} & \Gamma \end{array}$$

then $\Pi_\delta(B)[\gamma] = \Pi_{\delta'}(B[\gamma'])$.

Remark: Note that with this definition, the substitution under Π is not unique, but only unique up to an isomorphism. An interesting point is that if we take this definition and restrain it to quantifying over display maps, we do not get exactly the initial subtitution under simple Π types, but only something equal up to an isomorphism.

Proposition. *let $\delta : \Delta \to \Gamma$ be any subtitution, then Π_δ is right adjoint to δ.*

Proof: the proof is given as an annex.

4.7 Generalized Σ-types

Rules:

$$\frac{\delta : \Gamma' \to \Gamma \quad \delta' : \Gamma \to \Gamma' \quad t : \Diamond; \wedge(\Gamma') \vdash Id_{\wedge(\Gamma')[p]}[<< id, T(id) >, T(\delta \circ \delta') >] \quad z : \Gamma \vdash B[\delta']}{\Gamma \vdash Pair(\delta', t, z) : \Sigma_\delta(B)}$$

$$\frac{c : \Gamma \vdash \Sigma_\delta(B)}{P_1(c) : \Gamma' \to \Gamma}$$

$$\frac{c : \Gamma \vdash \Sigma_\delta(B)}{P_2(c) : \Diamond; \wedge(\Gamma') \vdash Id_{\wedge(\Gamma')[p]}[<< id, T(id) >, T(\delta \circ \delta') >]}$$

$$\frac{c : \Gamma \vdash \Sigma_\delta(B)}{P_3(c) : \Gamma \vdash B[\delta']}$$

Operations: The first equation is justified by the fact that to quantify over a 'term' δ' of 'type' δ, we do need a proof that δ' *is* of type δ (*ie, that* $\delta \circ \delta' = id$), however, the obtained quantified type is not different, for any proof that $\delta \circ \delta' = id$.

$$\begin{array}{rcl} Pair(a, c, b) & = & Pair(a, d, b) \\ Pair(P_1(a), \_, P_3(a)) & = & a \\ Pair(id, \_, q)[< p \circ P_1(q), P_3(q) >] & = & q \end{array}$$

The last equation is probably not written in the simpler form. However, it is sufficient to prove our adjointness, and it is reasonable: we have $\Gamma; \Sigma_\delta(B) \xrightarrow{<p\circ P_1(q), P_3(q)>} \Gamma; B$. Building our term in the second context is equivalent to just taking it from the first context.

Substitution under Σ: the conditions and rules are the same as for the Π types.

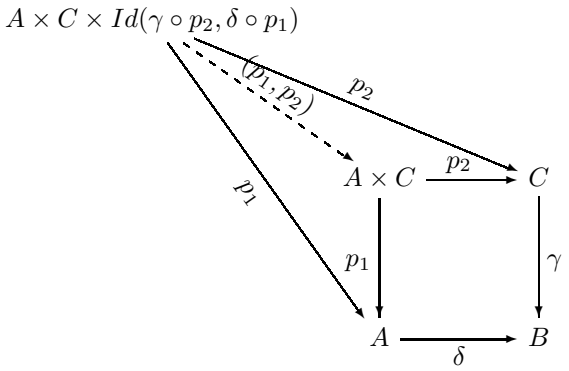
Proposition. *let $\delta : \Delta \to \Gamma$ be a substitution, then Σ_δ is left adjoint to δ.*

Proof: the proof is given in annex.

4.8 Pullbacks

Proposition. *Categories with families equipped with intensional identity types, extensional identity types, context products and generalized Π and Σ types have all pullbacks.*

Proof: only a proof sketch will be given here, a full proof will be given in annex. It is a known fact that a category with binary products and equalizers has all pullbacks: for all A, B, C objects, $\gamma : C \to B$ and $\delta : A \to B$, then this is a pullback:



Indeed, an object D and to arrows f_1, f_2 such that $f_1 : D \to A$, $f_2 : D \to C$ and $\gamma \circ f_1 = \delta \circ f_2$ induce an unique arrow $h : D \to A \times C \times Id(\gamma \circ p_2, \delta \circ p_1)$ which make the two triangles commute.

Now, note that given a context Γ, the category of types $Type_{\mathbf{C}}(\Gamma)$ has equalizers with identity types and binary products, which can be directly defined by simple Σ-types. Finally, we can expand the result to the base category $\mathbf{C}$ by using the mappings between types and contexts presented in section 4.5.

5 Equivalence results

Proposition. *A cwf with intensional and extensional identity types, context products, Π-types and Σ-types is a hyperdoctrine.*

Proof: the proof is given in annex. It's basically simple verifications, the longest part is to prove that for all $\Gamma \in |\mathbf{C}|$, $Type_{\mathbf{C}}(\Gamma)$ is cartesian closed.

Proposition. *A cwf with intensional and extensional identity types, context products, Π-types and Σ-types is locally cartesian closed.*

Proof: $\mathbf{C}$ is a category with all pullbacks and a terminal object. Moreover, as a $\mathbf{C}$-indexed category, it is a hyperdoctrine, as shown above. Therefore, the proposition of section 3.2 assures that it is locally cartesian closed.

6 Conclusion

We have finally accomplished all the path from cwfs to locally cartesian closed categories, giving sufficient additionnal structure to cwfs for it to be locally cartesian closed. Part of the work from lccc to cwfs has been done as well, but is not finished yet. The work accomplished can be continued in many ways: first, by finishing the proof from lccc to cwfs, and by trying to get back all cwfs additional structure from lccc notions, searching for an equivalence. Then, it may be possible to add a natural number object to the category and try to model inductive types.

A Π-types are right adjoints to substitution functors

Proposition. *Let $\delta : \Delta \to \Gamma$ be a substitution, then Π_δ is right adjoint to δ.*

Proof: We must prove the following adjunction:

$$\frac{B \to \Pi_\delta(C)}{B[\delta] \to C}$$

Let $f : \Gamma; B \vdash \Pi_\delta(C)[p]$, then $\Pi_\delta(C)[\delta \circ p] = \Pi_{id}(C[p])$, because the following diagram is a pullback and therefore allows such substitution ubder Π:

$$\begin{array}{ccc} \Gamma'; B[\delta] & \xrightarrow{p} & \Gamma' \\ {\scriptstyle id}\downarrow \uparrow{\scriptstyle id} & & \downarrow{\scriptstyle \delta} \\ \Gamma'; B[\delta] & \xrightarrow[\delta \circ p]{} & \Gamma \end{array}$$

Then, with

- $f[< \delta \circ p, q >] : \Gamma'; B[\delta] \vdash \Pi_\delta(C)[\delta \circ p] = \Pi_{id}(C[p])$
- $id : \Gamma' \to \Gamma'$
- $q[Refl_{\wedge(\Gamma')[p]}][< id, q >] : \Diamond; \wedge(\Gamma') \vdash Id_{\wedge(\Gamma')[p]}[< id, T(id), T(id \circ id) >]$

(note that the last type judgement is true because $q = T(id) = T(id \circ id)$)
we can build, using the *app* formation rule,

$$app(f[< \delta \circ p, q >], id) : \Gamma'; B[\delta] \vdash C[p]$$

Then we get the other way:

$$\frac{f : \Gamma'; B[\delta] \vdash C[p] \qquad < \delta \circ p, q >: \Gamma'; B[\delta] \to \Gamma; B}{\lambda_{<\delta \circ p, q>}(f) : \Gamma; B \vdash \Pi_{<\delta \circ p, q>}(C[p]) = \Pi_\delta(C)[p]}$$

Indeed, the substitution under Π is justified because the following diagram is a pullback:

$$\begin{array}{ccc} \Gamma'; B[\delta] & \xrightarrow{p} & \Gamma' \\ {\scriptstyle < \delta \circ p, q >}\downarrow & & \downarrow{\scriptstyle \delta} \\ \Gamma; B & \xrightarrow[p]{} & \Gamma \end{array}$$

Moreover, this does be a bijection, because:

$$\begin{array}{rcl} app(\lambda_{<\delta \circ p, q>}(f)[< \delta \circ p, q >], id) & = & f \\ \lambda_{<\delta \circ p, q>}(app(f[< \delta \circ p, q >], id) & = & f \end{array}$$

B Σ-types are left adjoints to substitution functors

Proposition. *Let $\delta : \Delta \to \Gamma$ be a substitution, then Σ_δ is left adjoint to δ.*

Proof: We must prove the following adjunction:

$$\frac{\Sigma_\delta(B) \to C}{B \to C[\delta]}$$

Let $f : \Gamma; \Sigma_\delta(B) \vdash C[p]$. Then, with

- $id : \Gamma'; B \to \Gamma'; B$
- $q[Refl_{\wedge(\Gamma';B)[p]}][< id, q >] : \Diamond; \wedge(\Gamma'; B) \vdash Id_{\wedge(\Gamma';B)[p]}[< id, T(id), T(id \circ id) >]$ (because $T(id) = T(id \circ id) = q$)
- $q : \Gamma'; B \vdash B[p]$

we use the Σ formation rule to build:

$$f[< \delta \circ p, Pair(id, q[Refl_{\wedge(\Gamma';B)[p]}][< id, q >], q) >] : \Gamma'; B \vdash C[\delta][p]$$

Now, let $f : \Gamma'; B \vdash C[\delta \circ p]$, then we have:

$$f[< p \circ P_1(q), P_3(q) >] : \Gamma, \Sigma_\delta(B) \vdash C[p \circ ST(< \delta \circ p, q > \circ P_1(q))] \qquad (1)$$

We can substitute $T(< \delta \circ p, q > \circ P_1(q))$ by $T(id)$. Indeed, we have:

$$q : \Gamma; \Sigma_\delta(B) \vdash \Sigma_\delta(B)[p] = \Sigma_{<\delta \circ p, q>}(B[p])$$

The substitution under Σ beeing justified by the following pullback:

$$\begin{array}{ccc} \Gamma'; \Sigma_{id}(B) & \xrightarrow{p} & \Gamma' \\ {\scriptstyle < \delta \circ p, q >}\downarrow & & \downarrow{\scriptstyle \delta} \\ \Gamma; \Sigma_\delta(B) & \xrightarrow[p]{} & \Gamma \end{array}$$

Let's prove that the above diagram is a pullback: we have $\Sigma_{id}(B) = \Sigma_\delta(B)[\delta]$ because of that other pullback:

$$\begin{array}{ccc} \Gamma' & \xrightarrow{id} & \Gamma' \\ {\scriptstyle id}\downarrow & & \downarrow{\scriptstyle \delta} \\ \Gamma' & \xrightarrow[\delta]{} & \Gamma \end{array}$$

So, the first diagram is the pullback diagram related to δ and $\Sigma_\sigma(B)$, as explained in section 3.3.

This justifies that

$$P_2(q) : \Diamond; \wedge(\Delta) \vdash Id_{\wedge(\Delta)[p]}[< id, T(id), T(< \delta \circ p, q > \circ P_1(q)) >]$$

from which we derive the following extensional equality type judgement:

$$T(< \delta \circ p, q >) = T(id) : \Diamond; \wedge(\Delta) \vdash \wedge(\Delta)[p]$$

We finally substitute it in (1), to get the wanted term:

$$f[< p \circ P_1(q), P_3(q) >] : \Gamma; \Sigma_\delta(B) \vdash C[p]$$

And finally, it does be a bijection:

$$\begin{array}{rcl} f[< p \circ P_1(q), P_3(q) >][< \delta \circ p, Pair(id_,, q) >] & = & f[< p, q >] \\ & = & f \\ f[< \delta \circ p, Pair(id, \_, q) >][< p \circ P_1(q), P_3(q) >] & = & f[< \delta \circ p \circ P_1(q), Pair(id_,, q)[< p \circ P_1(q), P_3(q) >] \\ & = & f[< p, q >] \\ & = & f \end{array}$$

(we do have $\delta \circ p \circ P_1(q) = p$, because $\delta \circ p = p \circ < p \circ \delta, q >$, and $P_1(q)$ beeing a 'term' of $< p \circ \delta, q >$, we have $< p \circ \delta, q > \circ P_1(q) = id$)

C All pullbacks

Proposition. *Categories with families equipped with intensional and extensional identity types and context products have all pullbacks.*

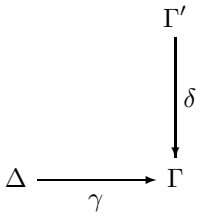
Proof: First, note that for all Γ', Γ and Δ contexts in the base category, for all $\delta : \Gamma' \to \Gamma$ and $\gamma : \Delta \to \Gamma$, the following diagram is a pullback:

$$\begin{array}{ccc} \Sigma(\wedge(\Gamma'), \Sigma(\wedge(\Delta)[p], Id_{\wedge(\Gamma)[p \circ p]}[< id, T(\delta)[p], T(\gamma)[< p \circ p, q >] >])) & \xrightarrow{P_1(q)} & \wedge(\Gamma') \\ \Big\downarrow{\scriptstyle P_1(P_2(q))} & & \Big\downarrow{\scriptstyle T(\delta)} \\ \wedge(\Delta) & \xrightarrow[T(\gamma)]{} & \wedge(\Gamma) \end{array}$$

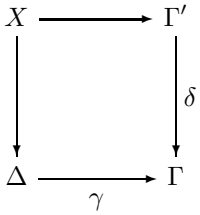
as the reader will guess, the direct syntactic proof that this is a pullback in the cwfs setting is rather boring and unreadable. Instead, let's just notice that the long Σ object described here corresponds to the binary product of $\wedge(\Delta)$ and of $\wedge(\Gamma')$, and of the equalizer of the two arrows $T(\delta) \circ P_1(q)$ and $T(\gamma) \circ P_1(P_2(q))$. That's exactly how pullbacks are built in a category with binary products and equalizers, and the proof is exactly the same as given in the book from Lambek&Scott[2], with just a far heavier syntax.

Now that we have this pullback, let's try to build one in the base category. Let's take in **C** a diagram

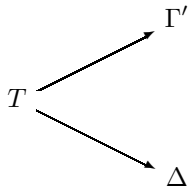
of the following form:



By using the context mappings, we get exactly the pullback described above, in $Type_{\mathbf{C}}(\Diamond)$. Now, note that for all A and B, $\Sigma(A, B)$ is isomorphic to $\wedge(\Diamond; A; B)$, so we can apply the S mapping and get a commutating diagram (S and T preserve composition)in $\mathbf{C}$:



That diagram is indeed a pullback, because if



then there is an unique $f : \wedge(T) \to \wedge(X)$ such that the two triangles commute in $Type_{\mathbf{C}}(\Diamond)$, which implies that $S(f) : T \to X$ is unique such that the two triangles commute in $\mathbf{C}$

D A cwf is a hyperdoctrine

Proposition. *A category with families, equipped with intensional and extensional identity types, context products, general* Σ *and* Π *types is a hyperdoctrine.*

Proof:

- forall $\Gamma \in |\mathbf{C}|$, $Type_{\mathbf{C}}(\Gamma)$ is cartesian closed
  - $Type_{\mathbf{C}}(\Gamma)$ has a terminal object T.
  - each pair A and B has a binary product $A \times B = \Sigma(A, B[p])$, with $\pi_1 = p_1$ and $\pi_2 = p_2$.
  - for A and B we define the exponent $[A \to B] = \Pi(A, B[p])$, and the arrow $eval = App(p_1(q), p_2(q)) : [A \to B] \times A \to B$
    let $f : C \times A \to B$, then we define $\lambda f = \lambda(f[< p \circ p, Pair(q[p], q) >]) : C \to [A \to B]$. We must prove that $eval \circ (\lambda f \times Id_A) = f$

$$
\begin{array}{rcl}
eval \circ (\lambda f \times Id_A) & = & eval[< p, Pair(\lambda f[< id, p_1(q) >], p_2(q)) >] \\
& = & App(p_1(q), p_2(q))[< p, Pair(\lambda f[< id, p_1(q) >], p_2(q)) >] \\
& = & App(\lambda f[< id, p_1(q) >], p_2(q)) \\
& = & App(\lambda(f[< p \circ p, Pair(q[p], q) >])[< id, p_1(q) >], p_2(q)) \\
& = & App(\lambda(f[< p \circ p, Pair(q[p], q) > \circ << p, p_1(q)[p] >, q >], p_2(q)) \\
& = & App(\lambda(f[< p, Pair(p_1(q)[p], q) >], p_2(q)) \\
& = & f[< p, Pair(p_1(q)[p], q) >][< id, p_2(q) >] \\
& = & f[< id, Pair(p_1(q), p_2(q)) >] \\
& = & f
\end{array}
$$

- for each $\delta : \Delta \to \Gamma$, δ^* (*ie. the substitution functor*) preserve exponents. let $a : \Gamma \vdash [A \to B] = \Pi(A, B[p])$, then

$$
\begin{array}{rcl}
\delta^*(a) = a[\delta] : \Delta \vdash \Pi(A, B[p])[\delta] & = & \Pi(A[\delta], B[\delta \circ p]) \\
& = & [A[\delta] \to B[\delta]] \\
& = & [\delta^*(A) \to \delta^*(B)]
\end{array}
$$

- let $\delta : \Delta \to \Gamma$, then δ does have a left adjoint Σ_δ and a right adjoint Π_δ.
- The Beck condition is verified, since it is exactly the condition of substitution under Σ-types.

References

[1] A.Pitts and P.Dybjer. *Semantics and Logics of Computation*. 1997.

[2] J.Lambek and P.J.Scott. *Introduction to higher order categorical logic*.

[3] M.Barr and C.Wells. *Category theory for computer science*.

[4] P.Dybjer. Internal type theory.

[5] P.Freyd. Aspects of topoÃ¯. 1972.

[6] T.S.E MAIBAUM S.Abramsky, Dov M.GABBAY. *Handbook of logic in computer science*, volume 5.

[7] R.A.G. Seely. Hyperdoctrines, natural deduction and the beck condition.

[8] R.A.G. Seely. Locally cartesian closed categories and type theory.

[9] S.Mimram. Decidability of equality in categories with families. 2004.