A presheaf model of dependent type theory

Alexis LAOUAR, M1 student in Computer Science at ENS Cachan, 2016/2017
Supervised by Thierry Coquand, Göteborgs Universitet, from 12/06/17 to 18/08/17

Contents

1 Introduction

The study of mathematical reasoning itself is of great interest both in mathematics and computer science: proofs might contain mistakes, and programs might contain bugs. A very effective way to test the correctness of those is to formalize them in a proof assistant, a tool that checks, step by step, if the reasoning in a proof is sound, or if the building blocks of a program do what they are supposed to do. Most of these assistants rely on a mathematical description of the notion of type, *type theory* - more precisely, dependent type theory, which allows, for instance, the type of the result of a program to depend on its argument.

However, the basic notion of type equality is surprisingly tricky to define. A new connection between the unexpectely similar fields of type theory and homotopy theory, the study of continuous transformations, was initiated by Voevodsky [8]. This *homotopy type theory* allows the definition of different possible notions of equality, and connects them, notably through Voevodsky's univalence axiom [7]. More precisely, two tricky types emerge: universe types, which always pose certain issues because of how close they are to paradoxes, and identity types, or identification types, as introduced by Martin-Löf in his pioneering paper [6]. These identity types appear naturally, but are somewhat hard to grasp: what is an element of an identity type? What is "an identification" of some types A and B in a universe type U?

Homotopy type theory gives a homotopic interpretation of identity types: $\mathrm{Id}\,UAB$ is the type of *paths* from A to B in U, and being "equal" (in an identity type meaning) is to be joinable by a certain path. From a type theoretic view, if $\mathrm{Id}\,UAB$ is inhabited by a term p, then p is a path from A to B, and A and B can be connected. This yields a new definition of equality, which in turn allows the definition of a notion of equivalence, a refinement of the notion of isomorphism, which is weaker than equality, in the sense that two elements that are equal are also equivalent. Formally, this means we have a map $\mathrm{Id}\,UAB \to \mathrm{Equiv}AB$.

Voevodsky's univalence axiom then states that this map itself is an equivalence: it is invertible. In other words, we have an inverse map $\mathrm{Equiv}AB \to \mathrm{Id}\,UAB$: two equivalent elements are equal. This would mean that the notions of equality and equivalence are one and the same, and therefore that we can use whatever characterisation of equality we have for two equivalent elements, and vice-versa. This also implies that two "equivalent" elements can be considered equal, and thus share the same properties - which leads, for instance, to functional extensionality (two functions that are equal on every possible value of their argument are equal) and the identification of isomorphic structures (a theorem on a group is also true on any isomorphic group (with certain restrictions)).

This univalence axiom seems to unveil equality at a very high, abstract level; but unfortunately, it is hard to *justify*. It implies strong and desirable properties, but is a strong axiom to require. Voevodsky, along with his formulation of the axiom, built a model based on simplicial sets, exposed in Kapulkin and Lumsdaine's dedicated paper [4], but the proof of univalence is non-constructive, and therefore does not explain how it can be added to type theory while preserving certain computational properties. In this paper, we use another model, in which the axiom is constructively provable, and which is more geometric, closer to computation, and therefore is better fit to a future application to proof assistants.

This model in terms of presheaf was already used by Simon Huber in his thesis [3], which served as a starting point for this work. In order to prove the univalence of the model, S. Huber introduced several homotopic notions, like path lifting (called *composition* in his work), that have a common denominator: they are property of types that are inherited through type builders. However, the focus was always set on types, when these special notions actually behave in a way surprisingly similar to types and terms. In this paper, we see these specific properties as objects on their own, that we call *type structures*, and adapt the model to make their own structure explicit. This allows us to shed light on how these inheritable properties work - for instance, what their addition to type theory as axioms implies, and how they can be added.

2 Several categorical intuitions

This whole paper involves categorical notions. In order for the paper to be understandable, the key definitions and intuitions will be described in this part. It is not intended to be complete, since it will mostly be informal; the aim of this brief summary is to give an intuitive understanding of the categorical context. The definitions and formal proofs can be found in appendices - this holds for the whole paper.

2.1 Categories

Categories were born from a desire to study not mathematical objects themselves, but rather their structure, and how they interact with each other. For instance, a group can be seen as a set of elements with a neutral element and a binary operation along with several axioms, and a morphism of groups is a function that respects the structure of its domain and codomain groups. Similarly, a ring has several components and axioms, and a ring morphism is a function respecting the ring structure.

We can widen this idea to any kind of algebraic structure, and to many more mathematical entities: an "object" is an entity, and a "morphism" transforms an entity into another with respect to their stucture, the key idea being that the morphisms *alone* are enough to describe their structure. In the case of groups, if we forget that we can build any kind of function on the underlying set of elements of groups, and limit ourselves to morphisms of groups, then their structure becomes palpable.

In light of this idea, a *category* consists of a class of *objects*, that we will often write $I, J, \dots$, and a class of *morphisms*, written $f, g, \dots$, that each have a domain object and a codomain object (similarly to functions) that can be composed when the domains and codomains fit, along with several axioms - the existence of an *identity morphism* on each object and associativity of composition (see appendix A for the formal definition).

2.2 Presheaves

One question that might come to mind is: since categories themselves have a structure, what about "category morphisms"? There such a notion, *functors*, that is fundamental in category theory. They are defined just like one would expect, a function sending objects of the first category onto objects of the second, doing the same for morphisms, in a way that respects the structure.

Although the notion of functor is central in every categorical framework, it is a particular case of functor that will be used in this paper: *presheaves*. In a nutshell, a presheaf is a set-valued contravariant functor; however, it is not the classic view of presheaves that is adapted to the presheaf model. In the following paragraph, we present an equivalent, altough different, definition of a presheaf; the classic definition, as well as the definition of a functor, can be found in appendix A.

Given a category $\mathcal{C}$, a *presheaf* Γ on $\mathcal{C}$ is defined by the following elements:

- For every object I of $\mathcal{C}$, a set $\Gamma(I)$.
- For every object I and every morphism $f : J \to I$ in $\mathcal{C}$, a *restriction* operation $\begin{cases} \Gamma(I) \to \Gamma(J) \\ \rho \mapsto \rho f \end{cases}$

The correspondance with the classic definition is simple: what we write ρf is normally written $\Gamma(f)(\rho)$.

A presheaf can be thought of with a certain notion of temporality. See the base category as "world sockets" (the objects) along with links between those world sockets (the morphisms). A presheaf fills the world sockets with actual worlds of elements (for a world socket I, $\Gamma(I)$ is an actual world of elements sitting in its place) and, for a given link between two world sockets $f : J \to I$, gives a concrete way to move "into the future" along a certain "timeline" f, the restriction operation $\rho \mapsto \rho f$.

Note that, with this notation for the restriction operation, the presheaf is implicit. This choice was made for two main reasons. First, we will almost always work with a fixed presheaf being given, so it

will most often be clear which presheaf we use to restrict an element; and when it's not, it's still not completely ambiguous, since when we write ρf, altough f is an element of the base category (independent of any presheaf), ρ is in a set $\Gamma(I)$, and looking at the definitions will make the presheaf clear. The second reason is that, as we will see later, the restriction operation is the composition in the Yoneda embedding, and this notation makes some complex calculations appear clearer (see appendix A for a simple example: the definition of a natural transformation).

We now have given the necessary definitions to present the presheaf model of type theory we have been talking about.

3 The presheaf model of type theory

3.1 Introduction

The presheaf model of type theory existed before this paper, and used a particular notion, a *category with families*. This model, when a specific category is chosen for it to be built on, is particularly well fit for homotopical notions, like path types, and was used by Simon Huber in his thesis [3]. It is a modified version of this model that will be used: one that has a whole hierarchy of universes, as well as *structured types*.

In this paper, we will not discuss *all* the notions introduced by Simon Huber that allowed him to prove the univalence axiom; the focus will be set on a specific and new notion, structured types. It is a way to describe, on a very high level, any property of types that is transferred though type builders - that is to say, a property such that if it is satisfied by two types A and B, then it is satisfied by the type $A \to B$ (with the same property for other type builders). We will come back to that later, with more detailed explanations as well as examples of structures.

3.2 Dependent type theory

Dependent type theory is an extension of classic type theory. This part discusses the creation of the first from the second: what is changed, and what is added. Classic type theory is fairly simple and well-known; in a nutshell, it formalizes the idea of elements of "type" int, or "type" int list, and allows the proof of theorems like "if f is of type $A \to B$ and a is of type A, then fa is of type B". The complete inference rules can be found in part 6 of Simon Huber's thesis [3].

In classic type theory, the "$\to$ - introduction" rule is the following :

$$\frac{\Gamma, a : A \vdash fa : B}{\Gamma \vdash f : A \to B}$$

This rule means that if, in context $\Gamma, a : A$, the term fa is of type B, then f is of type $A \to B$ in context Γ. In this rule, the type B is *independent* of a: whatever argument is given to f, the returned value will have type B. Dependent type theory allows the type of the returned value to depend on a, id est to be "B(a)". The type $A \to B$ becomes $(x : A) \to B(x)$, also written $\Pi(x : A)B(x)$. The "$\to$ - introduction rule" is adapted to become a "Π - introduction rule" as follows:

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda(x : a).t : (x : A) \to B}$$

and the elimination rules becomes :

$$\frac{\Gamma, t : (x : A) \to B \vdash u : B}{\Gamma \vdash t\, u : B(x/u)}$$

Dually, in pair types $A \times B$, B is allowed to depend on A, and dependent pair types are $(x : A) \times B(x)$, also written $\Sigma(x : A)B(x)$. The "pair introduction rule" is also modified, in a way similar to how Π-types are adapted. For complete inference rules and more explanations and intuitions, see Hoffman's dedicated paper [2] or part 6.2 of Simon Huber's thesis [3] for a more concise (but still complete) introduction.

3.3 The model

3.3.1 Category with families and universes

The presheaf model that will be considered uses an enriched version of the existing notion of category with families. The idea behind a category with families is to include the formalism of type theory in a base category itself. Therefore, the terminology will be very close, if not the same, but it is important to remember that for now, the category is *not* any kind of model of type theory. We simply define a framework in which the model will fit smoothly. For example, the objects of the base category will be called *contexts*, but this is simply a name, and they are nothing more than that. They should, however, be *thought of* as contexts of type theory, since this is how they are crafted to behave, but as long as we stick to the strict definition of a category with families and universes, the analogy goes no further.

A category with families and universes (CwF/U) consists of the following elements:

- A category $\mathcal{C}$, whose objects are called *contexts* and whose morphisms are called *substitutions*. We also require the category to contain an empty context.
- For all $\Gamma \in \mathcal{C}$, a set of *types* on Γ, written $\mathrm{Type}(\Gamma)$; and for any context Γ, any type A on context Γ, and any substitution $\sigma : \Delta \to \Gamma$, a type $A\sigma \in \mathrm{Type}(\Delta)$. This operation must satisfy $A\mathbb{1}_\Gamma = A$ (where $\mathbb{1}_\Gamma$ is the identity subtitution of Γ) and $(A\sigma)\delta = A(\sigma\delta)$ (where the substitution composition is written implicitly), for all well-defined σ and δ.
- For all $\Gamma \in \mathcal{C}$ and $A \in \mathrm{Type}(\Gamma)$, a set of *terms* of type A, written $\mathrm{Term}(A, \Gamma)$; and for any context Γ, any type A on context Γ, any term $a \in \mathrm{Term}(A, \Gamma)$, and any substitution $\sigma : \Delta \to \Gamma$, a term $a\sigma \in \mathrm{Term}(A\sigma, \Delta)$, such that $a\mathbb{1}_\Gamma = a$ and $(a\sigma)\delta = a(\sigma\delta)$, for all well-defined σ and δ.
- A sequence of Grothendieck universes $\mathcal{U}_0 \subseteq \mathcal{U}_1 \subseteq \cdots \subseteq \mathcal{U}_n \subseteq \cdots \subseteq \mathcal{U}$, id est with each set $\mathcal{U}_n$ (and $\mathcal{U}$) being closed under typical set operations:

  1. Transitivity: an element of an element of $\mathcal{U}_n$ is also an element of $\mathcal{U}_n$.
  2. Pairing: if $x, y \in \mathcal{U}_n$, then $\{x, y\} \in \mathcal{U}_n$.
  3. Power set: the set of parts of an element of $\mathcal{U}_n$ is also an element of $\mathcal{U}_n$.
  4. Union: the union of a familiy of sets *indexed on* $\mathcal{U}_n$ is also an element of $\mathcal{U}_n$.

  The idea behind universes is that their stabilities under those operations make them usable as a framework: if we are working with several elements or sets of elements defined from a universe, anything we build from them is likely to be in the universe as well.

The hierarchy of universes we have in this particular case should be thought of as different layers of types. For instance, any term built from base types and type builders will have a type "in" $\mathcal{U}_0$ (at level zero). Next, if we want to consider the type of types, and build types and terms that use this "meta-type", we will have types and terms at level one, and so forth. Unlike other links of the chain, $\mathcal{U}$ should be thought of as the reunion of all other universes, as a way to embed any element of any universe in a fixed universe.

One may wonder why not spare the effort of building this hierarchy by trying to define a type Ω such that Ω is itself of type Ω, thus going to the "meta-type" level without needing to go "a universe higher". This has been proved to be impossible through Girard's paradox (see [1] for more information), which can be compared to Russel's paradox: if there is such a type that is its own type, then all types are inhabited, which would mean, by the Curry-Howard correspondence, that everything is provable.

3.4 The presheaf model

Let us now present the model itself, which will be defined in a CwF/U. We assume given a base category $\mathcal{C}$.

Note that, in S. Huber's work [3], the category $\mathcal{C}$ is chosen to be the category of cubes, which is required to have a smooth definition of homotopic notions (like path types). However, no category has to be chosen for the model to represent dependent types in a way compatible with the addition of homotopic notions. Since we leave those notions out of this paper, we can keep considering a category $\mathcal{C}$ with no restriction whatsoever.

Let it be noted that, when working with natural transformations - which will be substitutions in our model -, we will often ommit the subscript object, since it can be determined by looking at the definitions, and more importantly makes the categorical notations even closer to the type theory syntax.

3.4.1 Contexts and substitutions

A context (in a type theory meaning) will be seen as a presheaf on $\mathcal{C}$, and a substitution as a natural transformation between presheaves. The category of contexts (in a CwF/U meaning) will therefore be the category of presheaves on $\mathcal{C}$, written $\widehat{\mathcal{C}}$. The empty context is the constant presheaf 1, for which all sets $1(I)$ are a singleton and all the restrictions constant.

From now on, we will not specify the meaning (CwF/U or type theory) of an ambiguous word, like type; given the sentence, it will always be clear which we are talking about.

3.4.2 Types

Given a context (i.e. a presheaf) Γ, a type $A \in \mathrm{Type}(\Gamma)$ (also written $\Gamma \vdash A$) will be defined by the following elements :

- For all $I \in \mathcal{C}$ and $\rho \in \Gamma(I)$, a set $A(I, \rho)$.
- For all $I \in \mathcal{C}$ and all arrows $f : J \to I$ in $\mathcal{C}$, a map from $A(I, \rho)$ to $A(J, \rho f)$, written $a \mapsto af$, such that $a\mathbb{1} = a$ and $(af)g = a(fg)$.

Note that in the above definition, ρf is the restriction of $\rho \in \Gamma(I)$ along $f : J \to I$, i.e. the image of ρ by the morphism Γf. However, it should be thought of as a restriction operation, as explained in the introduction to presheaves; this vision is better fit to use we will make, and will later let many analogies and symmetries appear.

The substitution on types is defined as follows: if A is a type in context Γ, and if $\sigma : \Delta \to \Gamma$, then $A\sigma \in \mathrm{Type}(\Delta)$ is defined by $(A\sigma)(I, \rho) = A(I, \sigma\rho)$, for all $I \in \mathcal{C}$ and $\rho \in \Delta(I)$, and we have the induced map $(A\sigma)(I, \rho) \to (A\sigma)(J, \rho f)$.

A quick and mechanical calculus proves that this verifies the required equations for substitutions - everything was built so things work smoothly in this part.

Note that there is a quicker, yet involving more categorical prerequisites, to define types in the model: a type in a context Γ is simply a presheaf on the *category of elements* of Γ, written $\int_{\mathcal{C}} \Gamma$. For an detailed explanation on this other definition, see appendix A.

3.4.3 Terms

Given a context Γ and a type $A \in \mathrm{Type}(\Gamma)$, a term $a \in \mathrm{Term}(\Gamma, A)$ (also written $\Gamma \vdash a : A$) is given by a collection of elements: for each $I \in \mathcal{C}$ and $\rho \in \Gamma(I)$, an element $a(I, \rho) \in A(I, \rho)$. This family also has to satisfy the required equations for terms: $a\mathbb{1} = a$ and $(af)g = a(fg)$, for all well-defined f and g.

3.4.4 Universes

We assume given a hierarchy of universes $\mathcal{U}_0 \subseteq \mathcal{U}_1 \subseteq \cdots \subseteq \mathcal{U}_n \subseteq \cdots \subseteq \mathcal{U}$ like required in a CwF/U, which we translate into a hierarchy on types: given a context Γ, $\mathrm{Type}_n(\Gamma)$ is the set of types A on Γ such that $A(I, \rho) \in \mathcal{U}_n$, for all $\rho \in \Gamma(I)$. Type_n sets inherit that $\mathrm{Type}_0 \subseteq \mathrm{Type}_1 \subseteq \cdots \subseteq \mathrm{Type}_n \subseteq \ldots$

Note that it can be easily proved that Type : $\Gamma \mapsto \text{Type}(\Gamma)$, defined as before, is also a presheaf on the category of contexts, and therefore, given the definition of Type_n, they are also presheaves on the category of contexts, and even form an increasing sequence of subpresheaves of Type (see appendix C for more details on this new hierarchy).

From these universes, we also define new presheaves U_n, called *universe preasheaves* (or *encoding presheaves*), on $\mathcal{C}$, by $U_n = \text{Type}_n \circ \text{Yon}$, where Yon is the Yoneda functor. But how does this represent a universe, or an encoding?

From this definition and from Yoneda's lemma, it can be proved that Type_n is represented by U_n, i.e. that $Type_n \cong \text{Hom}(\_, U_n)$. In other words, a type in $\text{Type}_n(\Gamma)$ is "canonically equivalent" to a substitution $\Gamma \to U_n$. This leads to an isomorphism between types of $\text{Type}_n(\Gamma)$ and terms in context Γ of a certain type defined directly from U_n; we can therefore see an n-type in context Γ as a term of a type closely related to U_n, which can indeed be said to function like a universe, or an encoding of types as terms. For more details on these proofs (which are quite technical), see C.

3.4.5 Dependent type theoretical operations

The aim of this part is more to give an outline of how the model is build than its most intricate gearing. For the definition of context extension, projections, and definitions of product/sum types, abstraction and application, see the dedicated appendix B. These dependent type theory operations were left out because it would require to introduce all definitions, which is done in appendix B.

With those ideas in mind, what has been defined so far is enough to understand type structures, and how we build a model "embedded" in this one.

4 The inner model: structured types

On a quick note, the attentive reader who referred to S. Huber's thesis [3] might have noticed a difference of vocabulary; in the thesis, type structures are called *compositions*. The reason for this is that S. Huber focused on one particular type structure, the lifting of paths - also called composition -, which is indeed preserved by type builders (in other words, given a path lifting operation for $A \in \text{Type}(\Gamma)$, and one for $B \in \text{Type}(\Gamma.A)$, we can build one for $\Pi AB \in \text{Type}(\Gamma)$; see chapter 6.4.5 of [3] for more details). S. Huber gives an explicit definition at the theory level, builds it in the model, and proves the univalence axiom for this type structure. What is done here is therefore less specific, and focuses less on the integration in the model, and more on how type structures *themselves* work.

4.1 Type structures

We assume given a category with families $\mathcal{C}$. On top of the sets Type and Term, respectively given for every context and for every context and type in that context, we consider a third collection of sets, $\text{Struct}(\Gamma, A)$, given for every context Γ and every $A \in \text{Type}(\Gamma)$. There are several properties that make a set a valid Struct set (also called structures set):

- An operation similar to types and terms, the interaction with substitutions: for any given context Γ, type $A \in \text{Type}(\Gamma)$, structure $s \in \text{Struct}(\Gamma, A)$, and substitutions $\sigma : \Delta \to \Gamma$ and $\delta : \Lambda \to \Delta$, $s\sigma \in \text{Struct}(\Delta, A\sigma)$, and the operation should guarantee $s\mathbb{1} = s$ and $(s\sigma)\delta = s(\sigma\delta)$, for any σ and δ defined as before.
- A structure building operation for all type building operations: for instance, since we have the rule $\dfrac{A \in \text{Type}(\Gamma) \quad B \in \text{Type}(\Gamma.A)}{\Pi AB \in \text{Type}(\Gamma)}$, we have the rule $\dfrac{c_A \in \text{Struct}(\Gamma, A) \quad c_B \in \text{Struct}(\Gamma.A, B)}{\Pi c_A c_B \in \text{Struct}(\Gamma, \Pi c_A c_B)}$. Note that, in the case of type building operations with no premise, this implies that we assume given a type structure for every base type.

The first property guarantees that structures behave with substitutions the same way types and terms do, and the second mirrors that structures are preserved along type builders, since this is what we want

them to represent.

Type structures, due to their very general and high-level definition, can mirror very different properties. A non-trivial example was already given, path lifting (for path types), but there are more simple examples. A type structure on type $A \in \mathrm{Type}(\Gamma)$ can, for instance, simply be a term $t \in \mathrm{Term}(\Gamma, A)$, which can represent a canonical element of the type A, or more generally a chosen element that can play a certain role in a specific situation.

We have defined *sets* $\mathrm{Struct}(\Gamma, A)$, for all $A \in \mathrm{Type}(\Gamma)$. The idea of the following is not only to give a set of possible structures, but to consider one *fixed* structure for each type, that we build using the structure building operations we are given. In order to do so, a new model is built inside the existing one - hence the name *inner model* -, for which every (new) type is an (old) type, along with a type structure.

4.2 The inner model

Let us defined the inner model component by component, like we did for the classic presheaf model.

4.2.1 Contexts

Like in the first model, contexts are the presheaves on a base category $\mathcal{C}$, and substitutions are still natural transformations between presheaves.

4.2.2 Structured types

Given a context Γ, the set of *structured types* on Γ is defined by $\mathrm{SType}(\Gamma) = \{(A, s) | A \in \mathrm{Type}(\Gamma), s \in \mathrm{Struct}(\Gamma, A)\}$.

The definition of the sets "$A(I, \rho)$" and type substitution are straightforward: for all $(A, s) \in \mathrm{SType}(\Gamma)$ and $\rho \in \Gamma(I)$, $(A, s)(I, \rho) \stackrel{\mathrm{def}}{=} A(I, \rho)$, and we define the substitution of structured types as follows: for all $(A, s) \in \mathrm{SType}\Gamma$ and $\sigma : \Delta \to \Gamma$, $((A, s)\sigma) = (A\sigma, s\sigma) \in \mathrm{SType}(\Delta)$.

Concerning the type building operations, the *structured type* building operations are directly given from *type* building operations and *structure* building operations: for instance, for Π-types, if we have two structured types $\Gamma \vdash (A, s_A)$ and $\Gamma.(A, s) \vdash (B, s_B)$, we define $\Pi(A, s_A)(B, s_B) = (\Pi AB, \Pi s_A s_B)$, and similarly for Σ-types.

Note that this "choice" of structure is not made for every type independently: we not choose one structure per type, we choose one *way to build* a structure for every type builder. The structures themselves are built from the structure building rules.

4.2.3 Terms

The aim of the inner model is to attach a given struture to every type, but terms should remain unchanged. Therefore, we simply define $\mathrm{STerm}(\Gamma, (A, s)) = \mathrm{Term}(\Gamma, A)$, with all operations identical to the first model. Abstraction and application are also defined like before, and the required properties trivially remain satisfied.

4.2.4 Universes

The hierarchy of universes remains the same as well. However, if we want to create universe presheaves $\widetilde{U_n}$, simply defining $\widetilde{U_n} = \mathrm{SType}_n \circ \mathrm{Yon}$ (where SType_n is defined similarly to the first model) does not yield an isomorphism between SType_n and $\mathrm{Hom}(\_, \widetilde{U_n})$, and therefore does not create the wanted universe presheaves. In order to do so, another property must be required: a notion of continuity of Type_n (see appendix C for more details). Once this condition is either assumed or proved, the representation theorem yielding an encoding presheaf $\widetilde{U_n}$ is proved for the inner model, but this is still a work in progress.

The explanation of this construction itself proves that it is indeed a model, all the details that were left out being either trivial, or similar to the first model. This defines the inner model, where a structure is fixed for each type.

4.3 Proving the univalence axiom

Before starting to prove the univalence axiom, let us recall that it is strictly a homotopy type theory notion, and that we have not introduced path types here, neither at the level of type theory, nor in the models. Defining these types in the presheaf model was extensively done in S. Huber's thesis [3] by specifying a base category, the category of cubes, allowing for a very geometrical vision of types, and a smooth interpretation of paths. However, this lightening of the presented construction does not cancel out what is new, which is making the operations on type structures completely explicit, highlighting the analogy with types and terms.

Keeping this in mind, there are different, constructive proofs of the univalence axiom in the inner model. One that I find particularly elegant considers a type, Univ, defined as:

$$\Pi(A : U)\,\mathrm{isContr}\,(\Sigma\,(X : U)\,\mathrm{Equiv}\,A\,X)$$

One can prove that the univalence axiom is equivalent to this type being inhabited (see the homotopy type theory bible [7] for more details). In the first model, this type can be proved to be empty. However, it *is* inhabited in the inner model; and if a structured type (A, s) inhabits Univ, then by the Curry-Howard correspondance, and since the type structures attached to a type mirror its construction, then the (regular) type A constitutes a proof of the axiom of univalence in the inner model. Furthermore, since A is explicitely derived from type building operations, then the proof it represents is constructive.

5 Conclusion

The presheaf model of dependent type theory beautifully represents the close relationship between type theory and homotopy theory, and allows a natural and constructive proof of the axiom of univalence. The closer study of structured types also sheds new light on how inheritable properties on types work - very much like types themselves, and using this analogy allows one to define a new model, in a surprisingly smooth way, that keeps the same properties as the first, while carrying the structure along with the types. Not only does this refine the previous model, making it better fit for possible use with proof assistants; it also gives such future works better insight on how to deal with properties on types one would like to guarantee.

However, although this extended type theory (dependent types + path types, baptised "cubical type theory" because of their geometrical interpretation as seen in S. Huber's thesis [3]) is well-suited for automation, it is still a young theory, and some fundamental results, like its type cheking's decidability, are yet to be proved. Concerning the model itself, the encoding presheaf is not yet completely understood, and altough it yields encoding/decoding operations, the practical uses of those operations has not yet been studied. Furthermore, the inner model is all about adding properties to our type theory, but apart from path lifting or, equivalently, the Kan filling operation, not many particular properties have been looked into, which leaves ground for numerous possible investigations.

Finally, I would like to thank Thierry Coquand for his patience, and for his always complete and comprehensive explanations and discussions. I also thank Inari Listema, Rebecca Cyrén, and the many other people whose name I did not catch with whom I had numerous passionating and riveting discussion, and who overall helped me get used to life in Sweden and made my stay in Gothenburg lively.

A Categorical background

This appendix is dedicated to the formal definition of categorical notions, as well as complete proofs of theorems. It will be intentionally concise, and should only be used as a cheat sheet for the body of the paper.

A.1 Basic notions

Definition A.1. A *category* is a pair $(\mathcal{O}, \mathcal{A})$ of two classes: a class of *objects* $\mathcal{O}$, whose elements are often named $I, J, K, \ldots$; and a class of *arrows* $\mathcal{A}$ (also called *morphisms*), often named $f, g, h, \ldots$. Arrows have a domain and a codomain, that are objects of the category; if an arrow f has a domain J and a codomain I, we write $f : J \to I$. If needed, we write $\mathrm{Obj}(\mathcal{C})$ the objets of $\mathcal{C}$, and $\mathrm{Arr}(\mathcal{C})$ its arrows. We also have a composition operation $\circ$, such that :

- If $f : J \to I$ and $g : K \to J$, then $f \circ g : K \to I$.
- For every object I, there is an identity arrow $\mathbb{1}_I : I \to I$, neutral to the composition (left and right): $f \circ \mathbb{1}_J = \mathbb{1}_I \circ f = f$, for all $f : J \to I$.
- The composition is associative: $f \circ (g \circ h) = (f \circ g) \circ h$, for all well-defined f, g, h.

We will sometimes write the composition implicitely (write fg for $f \circ g$), or ommit the object as subscript of identity arrows to lighten some expressions.

Definition A.2. Given two categories $\mathcal{C}$ and $\mathcal{D}$, a *covariant functor* $F : \mathcal{C} \to \mathcal{D}$ is given by two functions:

- A function $\mathrm{Obj}(\mathcal{C}) \to \mathrm{Obj}(\mathcal{D})$, whose application on an object I is written $F(I)$.
- A function $\mathrm{Arr}(\mathcal{C}) \to \mathrm{Arr}(\mathcal{D})$, whose application on a morphism f is written Ff, that *preserves* the domain and codomain: if $f : I \to J$, then $Ff : F(I) \to F(J)$. It is also required that $F\mathbb{1} = \mathbb{1}$ and that $(Ff) \circ_{\mathcal{D}} (Fg) = F(f \circ_{\mathcal{C}} g)$.

A *contravariant* functor F is a functor that reverses the domain and codomain of morphisms: if $f : J \to I$, then $Ff : F(I) \to F(J)$. Consequently, we require that $(Fg) \circ_{\mathcal{D}} (Ff) = F(f \circ_{\mathcal{C}} g)$ when $f : J \to I$ and $g : K \to J$.

Definition A.3. A *presheaf* is a contravariant set-valued functor, id est a contravariant functor from a category $\mathcal{C}$ into the category Set of sets, whose objects are sets and morphisms are classic set-theoretical functions.

Although this definition is compact and widely used, it is an equivalent definition of a presheaf that will be used in this work: given a category $\mathcal{C}$, a *presheaf* Γ on $\mathcal{C}$ is uniquely determined by the following elements:

- For every object I of $\mathcal{C}$, a set $\Gamma(I)$.
- For every object I and every morphism $f : J \to I$ in $\mathcal{C}$, a *restriction* operation $\begin{cases} \Gamma(I) \to \Gamma(J) \\ \rho \mapsto \rho f \end{cases}$

The idea is not only to write ρf instead of $(\Gamma f)\rho$: we see a presheaf as a class of sets $\Gamma(I)$, for all objects I, and not a way to "transform morphisms" like for any functor, but a way to restrict elements of those sets *along* morphisms.

Since a presheaf is set-valued, one can interpret a commutative diagram in terms of equality of elements, and not only equality of compositions of morphisms. For instance, if $f : J \to I$ and $g : K \to J$, the proposition $(Fg) \circ (Ff) = F(f \circ g)$ is equivalent to $\forall \rho \in F(I), F(fg)\rho = (Fg)(Ff)\rho$. This might not seem like much, but we will use and abuse this writing of properties throughout the appendices.

A.2 The category of elements of a presheaf

We assume given a category $\mathcal{C}$ and a presheaf Γ on that category. When this presheaf is the semantics of a context and we want to work with types or terms in that context, we are brought to work not with the category, but with all the elements $\rho \in \bigcup_{I \in \mathcal{C}}$, while also needing to "remember" which $\Gamma(I)$ they belong to. In order to do this while also allowing the restriction along morphisms, we define the well-named *category of elements* of Γ as follows:

Definition A.4. The *category of elements* of Γ has, as objects, pairs (I, ρ), such that I is an object of $\mathcal{C}$ and $\rho \in \Gamma(I)$, and has, for every arrow $J \xrightarrow{f} I$ of $\mathcal{C}$, an arrow $(I, \rho) \xrightarrow{\tilde{f}} (J, \rho f)$. It is written $\int_{\mathcal{C}} \Gamma$.

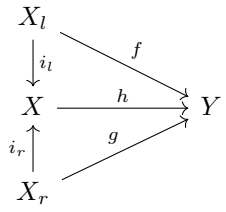
Having seen the definition of this category might raise questions about the definition of types, which somewhat resembled a presheaf definition. A type can, indeed, be defined as a presheaf on the category of elements of its context; for more details, see appendix C.

A.3 Commutative diagrams

Although commutative diagrams exist outside the context of categories, it is in the field of categories that they come the most handy, while also making beautiful symmetries appear graphically. They are an effective way to express equalities on compositions of functions (or, more generally, morphisms).

Definition A.5. A *diagram* is a finite, oriented graph, with objects of a given category as vertices, and morphisms as edges. The diagram is said to *commute* if, given two vertices v_1 and v_2, all compositions of morphisms represented by sequences of edges going from v_1 to v_2 are equal (in a categorical meaning).

For instance, the following diagram commutes:



if, and only if, $h \circ i_l = f$ and $h \circ i_r = g$.

A.4 Natural transformations

Definition A.6. Given two categories $\mathcal{C}$ and $\mathcal{D}$, and two functors (chosen contravariant, although the definition is easily changed to fit covariant functors) $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{C} \to \mathcal{D}$, a *natural transformation* $\sigma : F \to G$ is given by a family of morphisms $\sigma_I : F(I) \to G(I)$ in $\mathcal{D}$, indexed by objects I of $\mathcal{C}$, such that, for all $f : J \to I$, the following diagram commutes:

$$\begin{array}{ccc} F(I) & \xrightarrow{\sigma_I} & G(I) \\ \downarrow{\scriptstyle Ff} & & \downarrow{\scriptstyle Gf} \\ F(J) & \xrightarrow{\sigma_J} & G(J) \end{array}$$

That is to say, $(Gf) \circ \sigma_I = \sigma_J \circ (Ff)$. With the previous presheaf notation, this is equivalent to $\forall \rho \in F(I), (\sigma_I \rho) f = \sigma_J(\rho f)$: this condition of naturality on σ states that "restriction commutes with the transformation".

We will often ommit the subscript of a natural transformation, since it can be determined from the definition of the elements involved, and as foreseen in the property $\forall \rho \in F(I), (\sigma_I \rho) f = \sigma_J(\rho f)$, subscripts often bring unnecessary detail to expressions; this last identity, a property of presheafs, simply becomes $(\sigma\rho)f = \sigma(\rho f)$.

A.5 The Yoneda lemma

The Yoneda Lemma is a fundamental result in category theory: any category $\mathcal{C}$ is embedded in the category of presheaves on $\mathcal{C}$, written $\widehat{\mathcal{C}}$. This embedding is carried through the Yoneda functor:

$$\begin{aligned}\mathrm{Yon} : \mathcal{C} &\longrightarrow \widehat{\mathcal{C}} \\ I &\longmapsto \mathrm{Hom}(\_, I) \\ (f : I \to J) &\longmapsto \begin{cases} \mathrm{Hom}(\_, I) \to \mathrm{Hom}(\_, J) \\ (g : K \to I) \mapsto (fg : K \to J) \end{cases}\end{aligned}$$

Where $\mathrm{Hom}(\_, I)$ is the presheaf sending an object J to $\mathrm{Hom}(J, I)$, the set of all arrows $J \to I$, and a morphism $f : K \to J$ to the mapping $g \mapsto gf$, the composition with f on the left, sending morphisms $J \to I$ (in $\mathrm{Yon}(I)(J)$) into morphisms $K \to I$ (in $\mathrm{Yon}(I)(K)$). The image of a morphism $f : I \to J$ by Yon is sometimes written $\hat{f} : \mathrm{Yon}(I) \to \mathrm{Yon}(J)$.

Furthermore, this Yoneda functor is fully faithful, and for any given presheaf Γ and object I of $\mathcal{C}$,

$$\Gamma(I) \cong \mathrm{Hom}(\mathrm{Yon}(I), \Gamma)$$

This isomorphism being natural both in Γ and I. In the following appendices, we will use the definition of this isomorphism, so let us explicitely define it. The proofs of naturality will not be detailed here, though; the reason we go into the details of this isomorphism is that we will use those details later on, but the lemma itself is widely known. For more detailed proofs, see Saunders Mac Lane's "working" book [5].

We assume given a presheaf Γ and an object I of $\mathcal{C}$, as well as a natural transformation $\phi : \mathrm{Yon}(I) \to \Gamma$. The element $\underline{\phi} \in \Gamma(I)$ is simply defined by $\underline{\phi} = \phi_I(\mathrm{id}_I)$.

If we now have an element $\rho \in \Gamma(I)$, let us define a natural transformation $\bar{\rho} : \mathrm{Yon}(I) \to \Gamma$. In order to do so, we have to define $\bar{\rho}f$, for all $f : J \to I$. Let us require that $\bar{\rho}_I \mathrm{id}_I = \rho$; since $\bar{\rho}$ is supposed to be natural, for all $f : J \to I$, we have $\bar{\rho}f = (\bar{\rho}\mathrm{id}_I)f = \rho f$. We can therefore define $\bar{\rho}$ by requiring $\bar{\rho}_I \mathrm{id}_I$, and we uniquely determine $\bar{\rho}$ as a natural transformation. It is also immediate that the $\bar{\cdot}$ and $\underline{\cdot}$ transformations are inverse one to another.

We also have, as theorem, that $\bar{\rho}f = \rho f$, where in the left hand side of the equation, $\bar{\rho}f$ is the application of the natural transformation $\bar{\rho}$ on the element $\rho \in \Gamma(I)$, and in the right hand side, ρf is the restriction of ρ along f by the presheaf Γ.

B Dependent type notions in the presheaf model

The presheaf model of type theory was presented in part 3, but the dependent type constructions were left out for the sake of briefness. Below, we translate the dependent type building rules in the model, build the abstraction and application, and as an example of calculus in the model, prove the β-conversion.

B.1 Context entension and projections

We assume given a context Γ, id est a presheaf on $\mathcal{C}$, and a type A. We define the context $\Gamma.A$, which is also a presheaf on $\mathcal{C}$, as follows:

- For all $I \in \mathcal{C}$, $(\Gamma.A)(I) = \{(\rho, u), \rho \in \Gamma(I), u \in A(I, \rho)\}$.
- For all $f : J \to I$, $(\rho, u)f = (\rho f, uf)$.

Note that three different restrictions are used in the last expression: $(\rho, u)f$ is the restriction of (ρ, u) along f by presheaf $\Gamma.A$, ρf is by presheaf Γ, and u being in $A\rho$, we have a restriction operation (not defined as a presheaf restriction, although it can be proved to be, hence the terminology) $\rho \to \rho f$. (See part 3.4.2 to refresh your memory about types if needed)

Note that, in the definition of $(\Gamma.A)(I)$, we define "set-theoretic dependent pairs": if $(\rho, u) \in (\Gamma.A)(I)$, it means that $u \in A(I, \rho)$, so the second part of the pair depends on the first. This is how we will translate dependent types.

Let us now define the projections. First, we want to build a substitution $p : \Gamma.A \to \Gamma$, id est a natural transformation of presheafs. In order to do so, we must define $p(\rho, u)$ (ommiting the subscript), for all $I \in \mathcal{C}$ *and* all $(\rho, u) \in (\Gamma.A)(I)$. Naturally, we choose $p(\rho, u) = \rho \in \Gamma(I)$.

As for the second projection, we must define a term $\Gamma.A \vdash q : Ap$. By the definition of terms, we must define, for all I and $(\rho, u) \in (\Gamma.A)(I)$ an element $q(\rho, u) \in (Ap)(I, (\rho, u)) = A(I, \rho)$. We naturally choose $q(\rho, u) = u$.

These two examples are a good example of how the notations, that might seem far-fetched, are actually perfectly fit for working in the presheaf model. They bring type theory formalism and categorical semantics even closer, and all the definitions of the translations of types and terms come to down, for the definition of projections, to $p(\rho, u) = \rho$ and $q(\rho, u) = u$, exactly what we expect from projections.

Now, for the last defintion related to context extension, we assume given $\sigma : \Delta \to \Gamma$ and $\Delta \vdash a : A\sigma$, and we want to translate $(\sigma, a) : \Delta \to \Gamma.A$, another substitution, so a natural transformation. Once again, the definition comes naturally: for $\rho \in \Delta(I)$, we define $(\sigma, a)\rho = (\sigma\rho, a\rho)$. (Note that this notation is slightly wrong: one should write $a(I, \rho)$ instead of $a\rho$, but we ommited the object I here, only in order to highlight the symmetry of the definition)

With these definitions, the verification of the required equations is immediate and completely mechanical.

B.2 Π-types

We will now show that the defined model allows the definition of dependent types. Since Π and Σ types are dual, we will only prove that Π-types are supported; the proof for Σ-types is analogous.

B.2.1 Types and restrictions

The precise rule we want to translate in the model is the following:

$$\frac{\Gamma, x : A \vdash B}{\Gamma \vdash (x : A) \to B}$$

We therefore assume given a context Γ and a type $B \in \mathrm{Type}(\Gamma.A)$. Let us build $\Pi AB \in \mathrm{Type}(\Gamma)$.

Let ρ be an element of a given $\Gamma(I)$. We define $(\Pi AB)(I,\rho)$ to be a set $\{w = (w_f)_{f:J\to I}\}$ of families w of functions indexed by morphisms f of codomain I, such that, for any family $w \in (\Pi AB)(I,\rho)$, and for any $f : J \to I$ and $u \in A(J,\rho f)$, $w_f u \in B(J,(\rho f,u))$.

Although this definition is heavy, it becomes quite clear when one gets to process it. The functions w_f are dependent functions, whose codomain depend on their argument, like expected; and the equality we require means that the result of the application is "in the right set", the type B "applied to the argument". Note that the definition of such dependent functions is possible because we work with sets inside a given universe.

As for restrictions on Π-types, they are defined as follows.
Let $\Gamma \vdash \Pi AB, I \in \mathcal{C}, \rho \in \Gamma(I), w \in (\Pi AB)(I,\rho), f : J \to I$ and $g : K \to J$; $wf \in (\Pi AB)(J,\rho f)$ is defined by $(wf)_g u = w_{fg} u \in B(K,(\rho fg,u))$. This definition instantly yields that $w\mathbb{1} = w$ and that $(wf)g = w(fg)$.

It is delicate to explain why we need *families* of dependent functions in order to describe a single dependent term without looking at actual reasonings, proofs in the model, but keep the following intuition in mind. As was said in the part about presheaf intuitions, restricting along f corresponds to looking at the "future" along f. Now, we add another way to move in temporality: w_f is the future of w following f. Remembering this, the definition of the restriction becomes clear (the two ways to "move in time" are compatible). This intuition will, in particular, help with the understanding of the definition of λ in the following part.

B.2.2 Abstraction

Now that Π-types themselves have been defined, let us turn to defining λ. As a reminder, the λ-introduction rule we want to translate is:

$$\frac{\Gamma.A \vdash b : B}{\Gamma \vdash \lambda b : \Pi AB}$$

So let $\Gamma.A \vdash b : B$, and let us define $(\lambda b)(I,\rho) \in (\Pi AB)(I,\rho)$, for $\rho \in \Gamma(I)$.
By the definition of the semantics of ΠAB, $(\lambda b)(I,\rho)$ is a family of dependent functions; we define $((\lambda b)(I,\rho))_f u = b(J,(\rho f,u)) \in B(J,(\rho f,u))$.

By the temporal interpretation described in the last part, this definition means that taking the term λb in (I,ρ), going into the future f, and applying it to u, is like taking it into the future ρf along f, and then applying it to u. In other words, we need this temporality (id est those subscripts, we need the whole families of functions) in order to compute restrictions in a satisfying way, and the term λ forgets computes the right element $b(\dots)$ according to the temporality that was written as subscript.

Then, from the previous definitions and properties, we have, for $\rho \in \Gamma(I), f : J \to I, g : K \to J, u \in A(J,\rho f)$,

$$\begin{aligned}(((\lambda b)(I,\rho))_f u)g &= (b(J,(\rho f,u)))g \\ &= b(K,\rho fg,ug) \\ &= ((\lambda b)(I,\rho))_{fg}(ug)\end{aligned}$$

Which proves that the term λb behaves like an element of type ΠAB is supposed to behave in relation to restriction.

Furthermore, λb actually defined a term, since, considering the same quantifications as before, we have:

$$\begin{aligned}(((\lambda b)(I,\rho))f)_g u &= ((\lambda b)(I,\rho))_{fg} u \\ &= b(K,(\rho f g,u)) \\ &= (\lambda b)(J,\rho f)_g u \\ &= (\lambda b)(J,\rho f)\end{aligned}$$

Which is indeed the requirede property of terms.

B.2.3 Application

First, note that, given how we defined context extension (in a non commutative way), the contexts $\Gamma.A.B$ and $\Gamma.B.A$ are different, and contexts are less like lists than like stacks, on which we push new open variables with context extensions, and pop and bind formerly open ones with application. Therefore, given a term $\Gamma \vdash v : A$, consider the substitution $[v] : 1 \to 1.v$ by $[v] = (\mathbb{1}, v)$ (where 1 is the empty context and $\mathbb{1}$ the identity substitution); from what we just said, $[v]$ "substitutes v to the first open variable", with absolutely no ambiguity.

Let us now define the application term. Given a term $u \in \mathrm{Term}(\Gamma, \Pi AB)$ and a term $v \in \mathrm{Term}(\Gamma, A)$, we want a term $app(u,v) \in \mathrm{Term}(\Gamma, B[v])$. Remember the temporal interpretation of subscripts: in particular, if we want the "present", the subscript should simply be the identity. Therefore, we define, for $\rho \in \Gamma(I)$, $\mathrm{app}(u,v)(I,\rho) = (u(I,\rho))_{\mathrm{id}_I}(v(I,\rho)) \in B(I,(\rho, v\rho)) = B(\mathbb{1}, v)(I,\rho) = B[v]\rho$. It is simpler, and even trivial, to prove that this is a term and behaves correctly with restriction, since this has a more usual type that ΠAB.

B.2.4 β-conversion

For example, let us prove β-conversion. We assume given $\Gamma.A \vdash b : B$, $\Gamma \vdash v : A$, and $\rho \in \Gamma(I)$.

$$\begin{aligned}\mathrm{app}(\lambda b, v)(I,\rho) &= ((\lambda b)(I,\rho))_{\mathrm{id}}(v(I,\rho)) && \textit{Def. of app} \\ &= b(J,(\rho, v\rho)) && \textit{Def. of } \lambda \\ &= b(\mathbb{1}, v)(I,\rho) && \textit{Def. of substitution on product types} \\ &= b[v](I,\rho).\end{aligned}$$

This proves that $app(\lambda b, v) = b[v]$, also known as $\beta - conversion$.

C The universe presheaves

C.1 Types as presheaves

As seen in part 3.4.2, types are defined explicitely. From this definition, we can prove that other, more compact, characterizations of types - and even on Type: $\Gamma \mapsto \text{Type}(\Gamma)$ itself :

Theorem C.1. As defined in part 3.4.2, a type is a presheaf on the category of elements of its context, and this characterization is equivalent to the definition of a type without substitutions. Furthermore, Type itself is a presheaf on the category of contexts, and this characterization is equivalent to the definition of type substitution.

The proof of this theorem is systematic from the definitions.

Futhermore, a hierarchy of universes is given with the basic category. This allows us to build a hierarchy of types, that we define as presheaves on the category of contexts, as follows:

Definition C.1. For $n \in \mathbb{N}$, we define $\text{Type}_n(\Gamma) \stackrel{\text{def}}{=} \{A \in \text{Type}(\Gamma) | \forall (I, \rho) \in \int_{\mathcal{C}} \Gamma, A(I, \rho) \in \mathcal{U}_n\}$. The restriction operations are the same as for Type.

It is clear from this definition that $\text{Type}_n(\Gamma) \subseteq \text{Type}_{n+1}(\Gamma)$ for all $n \in \mathbb{N}$, and since these presheaves' restrictions are identical, a notion of subpresheaf naturally emerges:

Definition C.2. A presheaf F on $\mathcal{C}$ is a *subpresheaf* of another presheaf G on $\mathcal{C}$ if, and only if, for all objects I of $\mathcal{C}$, $F(I) \subseteq G(I)$, and their restrictions are the same.

Then Type_n forms an increasing sequence of presheaves bounded by Type.

C.2 Universe presheaves and representation of Type_n

We define the universe presheaves, U_n, as follows.

Definition C.3. $U_n = \text{Yon} \circ \text{Type}_n$.

Note that the composition above is the composition of functors, which defines both U_n's objects and its morphisms. Since Yon is covariant and Type_n is contravariant, U_n is contravariant, and set-valued, and therefore is indeed a presheaf.

In order to get the encoding that was announced in the introduction, the first step is to prove the following representation theorem: $\text{Type}_n \cong \text{Hom}(\_, U_n)$, which would mean that a n-type in a context Γ would be essentially the same as a substitution $\Gamma \to U_n$. The implications will be looked into later; for now, let us prove this theorem.

C.3 Proving the representation theorem

In order to prove that Type_n is isomorphic to $\text{Hom}(\_, U_n)$, we have to establish a natural transformation in both ways, so let us define a natural transformation $\beta : \text{Hom}(\_, U_n) \to \text{Type}_n$, and another $\alpha : \text{Type}_n \to \text{Hom}(\_, U_n)$.

C.3.1 β: transforming a substitution into a type

Let Γ be a context, and X be a subtitution (i.e., a natural transformations) $\Gamma \to U_n$. Then, $\beta(X)$ is supposed to be a type in $\text{Type}_n(\Gamma)$, so let ρ be an element of some $\Gamma(I)$.

We define $\beta(X)(I, \rho) \stackrel{\text{def}}{=} X(\rho)(I, id_I)$. Given this definition, the fact that this is indeed a type is inherited from the fact that $X(\rho)$ is a type.

C.3.2 α: transforming a type into a substitution

Let Γ be a context and A be a type in $\mathrm{Type}_n(\Gamma)$, then $\alpha(A)$ is supposed to be a natural transformation $\Gamma \to U_n$. So let I be an object of $\mathcal{C}$ and $\rho \in \Gamma(I)$, and let us define $\alpha(A)(\rho)$, which is supposed to be in $U_n(I)$, that is to say, a n-type on $\mathrm{Yon}(I)$.

With those elements defined, consider $\bar{\rho}$, the natural transformation $\mathrm{Yon}(I) \to \Gamma$ defined from ρ in the Yoneda lemma. It is a substitution of contexts, and since A is a type in context Γ, $A\bar{\rho}$ is a type in context $\mathrm{Yon}(I)$. Therefore, we can define $\alpha(A)(\rho) \overset{\text{def}}{=} A\bar{\rho}$.

But α is still not well-defined; we have to prove that $\alpha(A) : \rho \mapsto A\bar{\rho}$ is natural. This naturality condition is equivalent to the equality $(\alpha(A)(\rho))f = \alpha(A)(\rho f)$.

And indeed, we have:

$$\begin{aligned}
(\alpha(A)(\rho))f &= (A\bar{\rho})f && \textit{Def. of } \alpha \\
&= A(\bar{\rho}\hat{f}) && \textit{Since } U_n = \textit{Type}_n \circ \textit{Yon, prop. of restriction with Type}_n \\
&= A\rho\bar{f} && \textit{Naturality of } \bar{\cdot}\textit{, and def. of } U_n \\
&= \alpha(A)(\rho f).
\end{aligned}$$

The naturality of $\alpha(A)$ is established, and α is well-defined.

Now that α and β have been defined, let us prove their naturality.

C.3.3 Naturality of β

The naturality of β is equivalent to the property $\beta(X)\sigma = \beta(X\sigma)$.

Let $\rho \in \Delta(I)$. We have the following equalities:

$$\begin{aligned}
(\beta(X)\sigma)(I,\rho) &= \beta(X)(I,\sigma\rho) && \textit{Def. of type substitution} \\
&= X(\sigma\rho)(I,\mathrm{id}_I) && \textit{Def. of } \beta \\
&= (X\sigma)(\rho)(I,\mathrm{id}_I) && \textit{Naturality of } X \\
&= \beta(X\sigma)(I,\rho). && \textit{Def. of } \beta
\end{aligned}$$

And thus, the naturality of β is proved.

C.3.4 Naturality of α

Once again, the condition characterizing the naturality of α is $\alpha(A\sigma) = (\alpha(A))\sigma$.

Let ρ be an element of $\Delta(I)$. We have:

$$\begin{aligned}
\alpha(A\sigma)(\rho) &= (A\sigma)\bar{\rho} && \textit{Def. of } \alpha \\
&= A(\sigma\bar{\rho}) && \textit{Associativity of type substitution} \\
&= A(\overline{\sigma\rho}) && \textit{Naturality of } \bar{\cdot} \\
&= (\alpha(A))(\sigma\rho) && \textit{Def. of } \alpha \\
&= ((\alpha(A))\sigma)\rho && \textit{Def. of type substitution}
\end{aligned}$$

And the naturality of α is proved.

C.3.5 Reciprocity of α and β

Let us now prove that $\alpha\beta = \beta\alpha = \mathrm{id}$.

First, let us study $\alpha\beta$. Let Γ be a context, $X : \Gamma \to U_n$, $\rho \in \Gamma(I)$, and $f : J \to I$.

$$\begin{aligned}
\alpha(\beta(X))(\rho)(J,f) &= ((\beta(X))\bar{\rho})(J,f) && \textit{Def. of } \alpha \\
&= \beta(X)(J,\bar{\rho}f) && \textit{Def. of type substitution} \\
&= \beta(X)(J,\rho f) && \textit{Property of } \bar{\cdot} \\
&= X(\rho f)(J,\mathrm{id}_J) && \textit{Def. of } \beta \\
&= (X(\rho)\hat{f})(J,\mathrm{id}_J) && \textit{Naturality of } X\textit{, and } U_n = \textit{Type}_n \circ \textit{Yon} \\
&= X(\rho)(J,\hat{f}\mathrm{id}_J) && \textit{Def. of type substitution} \\
&= X(\rho)(J,f) && \textit{Property of } \hat{\cdot}
\end{aligned}$$

This proves that $\alpha\beta = \mathrm{id}$. Now, let Γ be a context, $a \in \mathrm{Type}_n(\Gamma)$, and $\rho \in \Gamma(I)$.

$$\begin{aligned}
\beta(\alpha(A))(I,\rho) &= (\alpha(A))(\rho)(I,\mathrm{id}_I) && \textit{Def. of } \beta \\
&= (A\bar{\rho})(I,\mathrm{id}_I) && \textit{Def. of } \alpha \\
&= A(I,\bar{\rho}\mathrm{id}_I) && \textit{Def. of type substitution} \\
&= A(I,\rho) && \textit{Prop. of } \bar{\cdot}
\end{aligned}$$

Thus, α and β and inverse to each other: the representation theorem is proved.

C.4 Encoding/embedding in a universe

If Γ is a context, we now define a new presheaf $\overline{U_n} \in \mathrm{Type}_{n+1}(\Gamma)$ by $\overline{U_n}(I,\rho) \overset{\text{def}}{=} U_n(I)$. It can be seen as a constant presheaf, whose value is the same on all the values of a given $\Gamma(I)$. Writing down the definition of a term of type $\overline{U_n}$ shows that such a term is equivalent to an arrow $\Gamma \to U_n$, and that we have an isomorphism $\mathrm{Hom}(\_, U_n) \cong \mathrm{Term}(\Gamma, \overline{U_n})$.

By transitivity, we get an isomorphism $\mathrm{Type}_n \cong \mathrm{Term}(\Gamma, \overline{U_n})$, for whom we have an explicit expression since we have one for the two isomorphisms we got it from. With it, we can encode any n-type on a context Γ into a term of type $\overline{U_n}$ in the *same* context Γ. The type $\overline{U_n}$ can therefore be seen as a way to encode n-types into n-terms, or as a way to embed n-types in a kind of universe by collapsing them to the level of terms.

C.5 In the inner model

Everything we did in this annex was in the first model, without type structures. Although it might seem like a herculean task, adding the type structures actually doesn't change a lot; for instance, we can define SType_n like we defined Type_n, and we can still define $\widetilde{U_n} = \mathrm{SType}_n \circ \mathrm{Yon}$. (Actually, when one has seen the proof of the representation theorem in the first model, it appears that this definition is the only one that has a chance to work).

However, there is one major difference: in general, we do not have the representation theorem $\mathrm{Type}_n \cong \mathrm{Hom}(\_, U_n)$, and without this representation of Type_n, nothing can be done any more. In order to be able to prove this, one has to require an additional condition: a condition of "continuity" on SType_n.

More precisely, we define $\mathrm{SType}'_n(\Gamma) \overset{\text{def}}{=} \{u = (u_f)_{f:\mathrm{Yon}(I)\to\Gamma}\}$, with $(u_f)g = u_{fg}$ for $f : \mathrm{Yon}(I) \to \Gamma$ and $g : \mathrm{Yon}(J) \to \mathrm{Yon}(I)$. Now, if $A \in \mathrm{SType}_n(\Gamma)$, A defines such a family by defining $A_\sigma \overset{\text{def}}{=} A\sigma$ for

$\sigma : \mathrm{Yon}(I) \to \Gamma$; this transformation yields a morphism $\mathrm{SType}_n(\Gamma) \to \mathrm{SType}'_n(\Gamma)$. Then, SType_n is said to be "continuous" if this arrow is an equivalence.

The proof that this notion of continuity leads to the representation theorem is not presented here; for more details, and more results about this continuity, please refer to the very well-written blog entry dedicated to free cocompletion by Qiaochu Yuan [9].

References

[1] Thierry Coquand. An analysis of Girard's paradox, 1986.

[2] Martin Hoffman. Syntax and semantics of dependent types, 1997.

[3] Simon Huber. Cubical interpretations of type theory, 2016.

[4] Chris Kapulkin and Peter LeFanu Lumsdaine. The Simplicial Model of Univalent Foundations (after Voevodsky), 2012.

[5] Saunders Mac Lane. Categories for the working mathematician, 1997.

[6] Per Martin-Löf. An intuitionistic theory of types, 1972.

[7] The Univalent Foundations Program. Homotopy type theory: Univalent Foundations of Mathematics, 2017.

[8] Vladimir Voevodsky. Univalent Foundations Project, 2010.

[9] Qiaochu Yuan. The free cocompletion I, 2014.