Dependent type theory as the initial category with families

Simon Castellan

December 8, 2014

Internship realized under the supervision of Peter Dybjer and Thierry Coquand
Chalmers University of Technology, Gothenburg.

Abstract

In this document, we present the syntax of dependent type theory as the initial category with families, extending the similar result relating simply-typed λ-calculus and cartesian closed categories. To define the interpretation of the syntax in any model we use a method along the lines of [Str91] where annotations are added to the syntax. By viewing dependent type theory as a generalized algebraic theory we are able to obtain an annotated syntax in a clean way. The main advantage of these annotations over Streicher's is that it makes solving the coherence problem [Cur93] very easy.

Contents

Introduction

Acknowledgement I would like to first thank Peter and Thierry for welcoming me in Chalmers and for the many talks about type theory, they made me learn a lot. Thanks to Guilhem, Jean-Philippe, Bassel, Nils and many other for the very nice time in Sweden.

Finally, thanks to Alexandre Miquel for making this internship possible.

Dependent type theories

Dependent type theories have been introduced by de Bruijn and his Automath system [BdMDdm73], by Howard, and later by Scott. Martin-Löf later developed these ideas to give a new formalism for constructive mathematics. It gives technical content to the famous Brouwer-Heyting-Kolmogorov interpretation of constructive proofs in first order predicate logic, along the lines of the *propositions as types* correspondence.

It can also be seen as functional programming languages in which it is possible to prove the correctness of a function while defining it. This viewpoint has largely been used in proofs assistant such as Agda or Coq. From this point of view, a dependent type theory can be seen as a type system for a λ-calculus more powerful than simply-typed λ-calculus, that can deal with *dependent types*, *i.e* types depending on terms. The traditional example of a dependent type is the type of lists of exactly n elements.

Two features specific to dependent type theories are:

- dependent product that generalizes the function space,
- universes, that bring a subset of types at the term level.

Dependent products Logically speaking, dependent products are types reflecting universal quantification. What is the constructive meaning of a universal quantification $\forall x \in A, B(x)$? It is a way to derive a proof of $B(x)$ given a term x in A. Seen as a function we see that the return type of the functions depends on its arguments. In type theory, this is called a dependent product and is often denoted by $\Pi(x : A)B(x)$.

What a term of this type should look like ? Its canonical members are abstraction $\lambda(x : A).t$ where t is a canonical proof of $B(x)$. If B does not depend on x, then $\Pi(x : A)B$ is just the usual function space that we denote by $A \to B$.

An example of a non-trivial dependent product is, if we let $\mathrm{List}(n)$ be the type of integer lists of length exactly n, the type of the `cons` function that pushes an element at the beginning of a list. Its type would be: $\Pi(n : \mathrm{nat})(\mathrm{List}(n) \to \mathrm{nat} \to \mathrm{List}(\mathrm{succ}(n)))$.

Universes Universes are used to bring a subset of the types at the term-level so that computation can be done with them, or quantification (since we quantify over terms). A universe set is a type containing *codes* for types, along with a decoding (meta)function el such that for each term x of type set, $\mathrm{el}(x)$ is a type. With this we can represent polymorphism: for instance the identity function could be typed as $\Pi(x : \mathrm{set})(\mathrm{el}(x) \to \mathrm{el}(x))$. Throughout this report we will use lowercase identifiers for codes and uppercase for types. A universe can be seen as a subset of types, called *small types*.

Syntax of dependent type theory

In this section we give a more formal account of the syntax of dependent type theory, and especially the calculus we are going to use in this report, and we discuss a list of issues that arises when dealing with dependent types, issues that do not appear in simply-typed λ-calculus.

The simply-typed λ-calculus is defined by a relation $\Gamma \vdash t : A$ which reads "in the context Γ, the term t has the type A", where contexts and types are generated by context-free grammars. In dependent type theory it is no longer the case.

Contexts Contexts are a list of named assumptions. But in dependent type theory, since types may depend on terms, we can have contexts of the forms: $x : \text{set}, y : \text{el}(y)$ postulating a small type x and an inhabitant y of this type. This context is well-formed, but this one: $y : \text{el}(x), x : \text{set}$ is not since y depends on x but appears before its definition.

Therefore we will need to consider a predicate of well-formedness on contexts, contrary to more simple type systems such as simply-typed λ-calculus, *and* a predicate of well-formedness of types relative to a context. They are noted $\Gamma \vdash$ ("Γ is well-formed") and $\Gamma \vdash A$ ("In Γ, A is well-formed").

In this document, we will use a calculus without variable, and thus contexts will just be a list of types noted $\diamond \cdot A_1 \cdot A_2 \cdot \ldots \cdot A_n$ (the usual notation would have been $x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n$).

Definitional equality and type conversion rule As usual, terms are subject to reduction rules such as for instance the β reduction: $(\lambda x.t)u \to t[u/x]$, and we have two distinct equalities on terms : $\equiv$ (syntactic equality) and $=$ definitional equality ($x = y$ if they are convertible, *i.e.* reducible to the same term). Since types may contain terms, we have these two equalities at the level of types as well. For instance $\text{el}((\lambda x.\text{nat})0) = \text{el}(\text{nat})$ although these two types are not syntactically equal. We want two equal types to have exactly the same inhabitants, and thus we need a type conversion-rule: if $A = A'$, and $\Gamma \vdash t : A$ then $\Gamma \vdash t : A'$. This requires the definitional equality to be defined at the same time as the typing rules, by equality judgements: $\Gamma \vdash A = A'$ ("A and A' are equal types in Γ") and $\Gamma \vdash t = t' : A$ ("t and t' are two equal terms of type A in Γ")

Explicit substitutions This is not specific to dependent type theories (one can define the simply-typed calculus with explicit substitutions), but in dependent type theory it is much more interesting [1] to present the calculus with explicit substitutions.

[1] Because of the rule for application for dependent product, substitution can no longer be defined after the typing rules. With explicit variables, the application rule looks like: if t has the type $\Pi(x : A)B$ and u has type the A, then $t\ u$ has the type $B[u/x]$ (and not B as in simply typed λ-calculus)

"Explicit substitutions" means that the operation of substitution is not a meta-operation defined *a posteriori*, but is part of the syntax. Substitutions operates on both terms and types since types can contain terms and thus variables.

A substitution can be seen as an implementation of a context Δ within another context Γ. Such a substitution will have the type $\Gamma \to \Delta$. Informally, one can see a substitution from Γ to $\Delta = \diamond \cdot A_1 \cdot \ldots \cdot A_n$ as a list of terms $\langle t_1, \ldots, t_n \rangle$ such that t_i has the type A_i in Γ.

The substitution of f in a term t (resp. a type A) will be denoted by $t[f]$ (resp. $A[f]$). Whenever we have a substitution $f : \Gamma \to \Delta$ and a type A (or a term t) well-formed in Γ, we can substitute

[1]

the variables of A by the their implementation given by f, and obtain a type $A[f]$ (or a term $t[f]$ of type $A[f]$) well-formed in Γ.

De Bruijn indices and substitutions Our calculus will use de Bruijn indices because it saves us some trouble [2]. De Bruijn indices are a way to refer to assumption not by a name but by the index of it in the countext. If we note v_i the i^{th} indice we have:

- $\Gamma \cdot A \vdash v_0 : A$, v_0 refers to the last assumed hypothesis
- $\Gamma \cdot B \vdash v_{i+1} : A$ if $\Gamma \vdash v_i : A : v_{i+1}$ forgets one assumption and acts like v_i.

But, if we keep our intuition of what a substitution is — the implementation of a context *within* another —, then there is a projection substitution $p : \Gamma \cdot A \to \Gamma$. If we have $\Gamma \vdash t : B$ then we have $\Gamma \cdot A \vdash t[p] : B[p]$. Here $t[p]$ is t lifted, meaning that each occurrence of v_i in t is replaced by v_{i+1} in $t[p]$. So $t[p]$ forgets the last assumed assumption (since v_0 never appear in $t[p]$) and acts like t. So, now in order to represent de Bruijn indices in our syntax, we only need v_0, since we can define $v_{i+1} = v_i[p]$. We call this v_0, q. By definition of q we have $\Gamma \cdot A \vdash q : A$ for any Γ, A.

If we were to implement the projection substitution, we would have to use de Bruijn indices (because p can be seen as $\langle v_n, v_{n-1}, \dots, v_0 \rangle$), but since we are going to use categories with families that uses this approach, we prefer to choose p as being built-in.

[2] This not specific to dependent type theory, as De Bruijn indices saves us the trouble to consider terms *modulo α-conversion*.

Semantics of dependent type theory

There is several categorical framework to define what semantics of type theory is. In this report we only consider categories with families[Dyb96, CD11]. Like any other categorical formalization of type theory, the idea behind categories with families is to see contexts and substitutions as a category on which are defined types and terms. The specific part is to see terms and types as families over a context.

Families First, we give the definition of the **Fam** category, the category of families.

***Definition* 1.** — *The category **Fam** is defined by:*

- *its object are families (A, B) where A is a set and B is a A-indexed set (meaning that $\forall x \in A, B(x)$ is a set). The family (A, B) will be denoted $(B(x))_{x \in A}$.*
- *a map from a family (A, B) to (C, D) will be a couple (f^0, f^1) where $f^0 : A \to C$ and $g^0 : \bigcup_x B(x) \to \bigcup_x D(x)$ such that $D(f^0(x)) = f^1(C(x))$ for $x \in A$.*
- *composition is defined point-wise*
- *identity on (A, B) is just the identity $(\mathrm{id}_A, \mathrm{id}_{\bigcup_x B(x)})$.*

For each context Γ, we have a set of types well-formed in Γ, $Ty(\Gamma)$ and for each $A \in Ty(\Gamma)$ we have a set of terms of type A inside Γ, $Tm(\Gamma, A)$.

Moreover, if we have a substitution $f : \Gamma \to \Delta$, we have maps : $^-[f] : Ty\Delta \to Ty\Gamma$ and $^-[f] : Tm(\Delta, A) \to Tm(\Gamma, A[f])$, which leads us to see the operation $\Gamma \mapsto (Tm(\Gamma, A))_{A \in Ty(\Gamma)}$ as a contravariant functor from the category of contexts to **Fam**, the category of families.

2

Contexts formation To build contexts as a list of types, we require two things on the category of contexts:

- a terminal element $\diamond$ representing the empty context. For each context Γ, there exists a unique morphism $!_\Gamma : \Gamma \to \diamond$, the *empty* morphism (since there is nothing to implement)
- a context comprehension: if we have a context Γ and $A \in Ty(\Gamma)$ a type over Γ, we can form the context $\Gamma \cdot A$, such that
  - there is a morphism $p : \Gamma \cdot A \to \Gamma$, the *projection morphism*. Intuitively it corresponds to the fact that one can easily implement Γ inside $\Gamma \cdot A$, by just forgetting the last variable.
  - there is a term $q \in Tm(\Gamma \cdot A, A)$. We can see q as a variable referring to the last hypothesis assumed (here A).
  - for each substitution $f : \Delta \to \Gamma$ and term $t \in Tm(\Gamma, A[f])$, we can extend f by t and have a morphism $\langle f, t\rangle : \Gamma \to \Delta \cdot A$, such that $p \circ \langle f, t\rangle = f$ (projection forgets the last term) and $q[\langle f, t\rangle] = t$ (variable retrieve the last term)

A formal definition will be given in section Semantics.

Contribution of this report

This report generalizes the well-known result that simply-typed calculus can be seen as an initial cartesian closed category to type theory, *i.e.* we prove that the syntax of type theory can be seen as an initial category with families (theorem initiality). This problem requires us to interpret our syntax in any CwF, and there is two common ways of defining such an interpretation:

- by means of a partial interpretation on raw-terms (see [Str91]). The correctness of the interpretations in this context means that the interpretation is defined on terms (well-typed raw terms)
- by defining the interpretation on derivation. Because of type conversion rules, a judgement can have several derivations (this is known as the *coherence problem*). One then needs to prove that two derivations of the same judgement get the same denotation in the model. This approach is considered in [Cur93].

To define its partial interpretation in [Str91], T. Streicher needs to add some typing annotations in abstractions and applications. He then proceeds to prove that these annotations can be left out through a stripping lemma: to a term without annotations corresponds at most one well-typed term with annotations, thus justifying the traditional syntax without any annotations.

However in [Cur93], P.-L. Curien has a different kind of annotations (type conversions are made explicit in the syntax) and the proof that two derivations gets the same semantics is very technical.

The approach considered here is to view our calculus as a generalized algebraic theory (this has been done in [Dyb96]) and formalize it completely inside set theory. In this context, the coherence problem is easy to solve although the syntax is very verbose. We then proceed to show that this annotated syntax and the regular syntax are equivalent.

The report is organzied as follows:

- in the section Desugarized syntax, we define the fully explicit syntax and prove the *coherence property* of this syntax,
- in the section Semantics, we give the semantics of this syntax and prove the initiality of the term-model induced by the desugarized syntax,
- in the section Regular syntax, we introduce a syntax without any annotations, and we prove that the two syntax are equivalent, and the two corresponding term models are isomorphic.

1 Desugarized syntax

1.1 Generalised Algebraic Theories (GAT) and explicit syntax

Generalized Algebraic Theories were introduced by Cartmell[Car85] as generalisation of algebraic theories to handle dependent types. This framework can be used to describe the theory of categories or dependent type theory: in [Dyb96], Peter Dybjer gives a presentation of type theory (using category with familes combinators) as a Generalized Algebraic Theory.

Our goal is to define dependent type theory based on the idea that its core can be seen as the generalized algebraic theory of categories with families (GAT of CwFs, see Dybjer [Dyb96]). We use set theory as a metalanguage and define precisely the raw syntax and the derivable judgement using this idea. The GAT of CwFs yields a very "explicit" raw syntax, where types and terms are annotated with context and type information not available in the usual raw syntax of lambda calculus, whether given as usual or as a calculus of explicit substitutions.

Although our presentation does not use explicitly the GAT framework, we give an overview of what is a GAT. A GAT is given by:

- Sorts operators. Sorts operators comes with an arity and an introduction rule. They build up *sorts* that represent sets of terms (and we note $x \in A$ to mean that x belongs to the sort A [3]). The argument passed to sort operators are *terms*. For instance, in our case, we have $Context$ a sort operator of arity 0, and a sort $Ty(^-)$ of arity 1 and introduction rule:

$$\frac{\Gamma \in Context}{Ty(\Gamma) \text{ is a sort}}$$

- Terms operators. Terms in GATs look very much like terms in ordinary first order theories, build up with *term operators*. Each term operator comes with its introduction rule. For example, the introduction rule for $\cdot$ (of arity 2):

$$\frac{\Gamma \in Context \quad A \in Ty(\Gamma)}{\Gamma \cdot A \in Context}$$

- Term equality. A theory should provide a definition (a judgement) for term equality.

The typing rules of a GAT must satisfy these two restrictions:

- every variable appearing in the premisses of a rule must be declared,
- every variable appearing in the premisses of a rule must appear in the term being introduced.

[3]

For instance the rule for Π-introduction (remember: we use de Bruijn indices):

$$\frac{B \in Ty(\Gamma \cdot A)}{\Pi(A, B) \in Ty(\Gamma).}$$

B is declared by the statement $\Gamma \cdot A \vdash B$ but neither Γ nor A are. Besides the term $(\Pi(A, B))$ being constructed has no reference to Γ. To make this rule completely explicit we need to write:

$$\frac{\Gamma \in Context \quad A \in Ty(\Gamma) \quad B \in Ty(\Gamma \cdot A)}{\Pi(\Gamma, A, B) \in Ty(\Gamma)}.$$

[sort] The notation "$x \in A$" is local to this section, later on we use the traditional judgement notation $\Gamma \vdash A$ (and others), because we need a uniformity that does not appear in regular judgement notation, and this notation makes it easy to distinguish the sort (A) from the term (t) which is less obvious with $\Gamma \vdash t : A$.

By definition, generalized algebraic theories are based on dependent types (since sorts depend on terms) but type theory (as CwFs) can be seen as GATs [Dyb96], so the correspondence goes both way.

1.2 Definition of our calculus

In this section we give all the rules defining our calculus. To define our calculus, we need to explain:

- What are the well-formed contexts, notation: $\Gamma \vdash$
- What are the well-formed types in a context Γ, notation: $\Gamma \vdash A$
- What are the well-formed terms in a context Γ of type A, notation: $\Gamma \vdash t : A$
- What are the well-formed substitutions from a context Γ to a context Δ, notation: $\Gamma \vdash f : \Delta$

We also need to define what it means for two entities of the same kind to be convertible:

- Equality of two contexts, notation: $\Gamma = \Gamma' \vdash$
- Equality of two types in a context Γ, notation: $\Gamma \vdash A = A'$
- Equality of two terms of type A in Γ, notation: $\Gamma \vdash t = t' : A$
- Equality of two substitutions from Γ to Δ, notation $\Gamma \vdash f = f' : \Delta$

Later on, the expression *typing judgement* will refer to the first four kind of judgements, and *equality judgements* to the four last. A *typing derivation* (resp. *equality derivation*) will be a derivation (according to the rules enumerated in the section Rules) whose conclusion is a typing judgement (resp. equality judgement). Since we are working with dependent types, we need to define these eight judgements simultaneously.

The calculus we are about to define is known as Martin-Löf's *Logical Framework* (LF) and consists in a universe set along with Π-types.

Application in presence of explicit substitutions. In presence of explicit substitutions, application can be handled differently than usual, by defining a unary operator, ap (along the lines of traditional denotation semantics for λ-calculus). The intuition behind ap is that it is the inverse of the λ operator. Given a term of type $\Pi(A, B)$ in Γ it returns a term of type B in $\Gamma \cdot A$, *i.e.* with one more free variable. One can switch from one representation to the other by the following equation:

$$t\ u = \mathrm{ap}(t)[\langle p, u\rangle], \quad \mathrm{ap}(t) = t[p]\ q$$

In the first equation $\langle p, u\rangle$ is the substitution that one would usually denote by $[u/0]$.

Pre-syntax We define first the syntax of raw terms (*pre-syntax*), and then define typing rules for these raw terms. "Terms" will then denote *well-typed raw terms*.

$\Gamma, \Delta ::=$	**contexts**
$\mid \diamond$	Empty context
$\mid \Gamma \cdot A$	Context comprehension
$A, B ::=$	**types**
$\mid \mathrm{set}(\Gamma)$	Universe of small types
$\mid \Pi(\Gamma, A, B)$	Dependent product
$\mid \mathrm{el}(\Gamma, x)$	Elements of a small type
$\mid A[f]^{\Gamma}_{\Delta}$	Explicit substitutions
$t, u ::=$	**terms**
$\mid q(\Gamma, A)$	Reference to last assumption
$\mid \lambda(\Gamma, A, B, t)$	λ-abstraction
$\mid \mathrm{ap}(\Gamma, A, B, t)$	Unary application
$\mid (t : A)[f]^{\Gamma}_{\Delta}$	Explicit substitutions
$f, g ::=$	**substitutions**
$\mid \mathrm{id}(\Gamma)$	Identity substitution
$\mid p(\Gamma, A)$	Projection substitution
$\mid \mathrm{comp}(\Gamma, \Delta, \Theta, f, g)$	Composition of substitution(reversed)
$\mid !_{\Gamma}$	Empty substitution
$\mid \langle f^{\Gamma}_{\Delta}, (t : A)\rangle$	Extension

1.3 Rules

There are a few common rules that will appear in the definition of the judgements. Each equality judgement will come with rules for reflexivity, symmetry, transitivity, even though it is not strictly necessary, (it could be derived) as well as congruence rules. Each judgement (except for context and context equality) will come with type conversion rules that allow to replace any components of the judgements with another entity that is convertible to the first one. It comes from the more general rule in GATs that says that:

$$\frac{t \in S \quad S = S'}{t \in S'}$$

In our case, equality on sorts is just congruence thus giving us usual type conversion rules. Despite our desire to define completely the system, all those rules will be omitted in this document for lack of space.

Moreover, because some rules are long, not all premisses are going to be included. Premises declaring variables such as contexts may be omitted if it is obvious what should be the corresponding premisse. For instance, in equality rule, whenever we write $\Gamma \vdash A = B$, we assume also the premisses $\Gamma \vdash A$ and $\Gamma \vdash B$. Likewise, we may not always write terms in a fully explicit way and may drop some arguments to keep it readable.

If $\Gamma \vdash f : \Delta$, we note $f^+ = \langle \mathrm{comp}(p, f), q \rangle$. This alias is going to be used in rules, and we will have $\Gamma \cdot A \vdash f^+ : \Delta \cdot A$.

Rules for context We give the rules for context.

$$\frac{}{\diamond \vdash} \qquad \frac{\Gamma \vdash \qquad \Gamma \vdash A}{\Gamma \cdot A \vdash}$$

The only rule for context equality (apart from reflexivity/symmetry/transitivity) is a congruence rule:

$$\frac{\Gamma = \Gamma' \quad \Gamma \vdash A = A'}{\Gamma \cdot A = \Gamma' \cdot A'}$$

This kind of rules will be omitted for other judgements.

Types

$$\frac{\Delta \vdash A \quad \Gamma \vdash f : \Delta}{\Gamma \vdash A[f]^\Gamma_\Delta} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \mathrm{set}(\Gamma)} \qquad \frac{\Gamma \vdash \quad \Gamma \vdash t : \mathrm{set}(\Gamma)}{\Gamma \vdash \mathrm{el}(\Gamma, t)} \qquad \frac{\Gamma \vdash \quad \Gamma \vdash A \quad \Gamma \cdot A \vdash B}{\Gamma \vdash \Pi(\Gamma, A, B)}$$

$$\frac{\Gamma \vdash A \quad \Gamma = \Gamma' \vdash}{\Gamma' \vdash A}$$

Type equality

$$\frac{\Gamma = \Gamma' \vdash \quad \Gamma' \vdash A = A'}{\Gamma \vdash A = A'}\ TyConv \qquad \frac{\Gamma \vdash \quad \Gamma \vdash A}{\Gamma \vdash A[id]^\Gamma_\Gamma = A}\ IdSub \qquad \frac{\Gamma \vdash \quad \Delta \vdash \quad \Gamma \vdash f : \Delta}{\Gamma \vdash \mathrm{set}(\Delta)[f]^\Gamma_\Delta = \mathrm{set}(\Gamma)}\ SubSet$$

$$\frac{\Delta \vdash \quad \Gamma \vdash \quad \Delta \vdash t : \mathrm{set}(\Gamma) \quad \Gamma \vdash f : \Delta}{\Gamma \vdash \mathrm{el}(\Delta, t)[f]^\Gamma_\Delta) = \mathrm{el}(\Gamma, (t : \mathrm{set}(\Delta))[f]^\Gamma_\Delta)}\ SubEl$$

$$\frac{\Delta \vdash A \quad \Delta \cdot A \vdash B \quad \Gamma \vdash f : \Delta}{\Gamma \vdash \Pi(\Delta, A, B)[f] = \Pi(\Gamma, A[f]^\Gamma_\Delta, B[f^+]^\Gamma_{\Delta \cdot A}}\ Sub\Pi$$

$$\frac{\Theta \vdash A \quad \Delta \vdash f : \Theta \quad \Gamma \vdash g : \Delta}{\Gamma \vdash A[f]^\Delta_\Theta[g]^\Gamma_\Delta = A[\mathrm{comp}(\Gamma, \Delta, \Theta, g, f)]}\ SubComp$$

Terms

$$\frac{\Gamma \vdash \quad \Gamma \vdash A}{\Gamma \cdot A \vdash q(\Gamma, A) : A} \qquad \frac{\Gamma \vdash \quad \Delta \vdash \quad \Delta \vdash A \quad \Delta \vdash t : A \quad \Gamma \vdash f : \Delta}{\Gamma \vdash (t : A)[f]^{\Gamma}_{\Delta} : A[f]^{\Gamma}_{\Delta}}$$

$$\frac{\Gamma \vdash \quad \Gamma \vdash A \quad \Gamma \cdot A \vdash B \quad \Gamma \cdot A \vdash t : B}{\Gamma \vdash \lambda(\Gamma, A, B, t) : \Pi(\Gamma, A, B)} \qquad \frac{\Gamma \vdash \quad \Gamma \vdash A \quad \Gamma \cdot A \vdash B \quad \Gamma \vdash t : \Pi(\Gamma, A, B)}{\Gamma \cdot A \vdash \mathrm{ap}(\Gamma, A, B, t) : B}$$

$$\frac{\Gamma \vdash t : A \quad \Gamma = \Gamma' \vdash \quad \Gamma \vdash A = A'}{\Gamma' \vdash t : A'}$$

Term equality

$$\frac{\Gamma = \Gamma' \vdash \quad \Gamma \vdash A = A' \quad \Gamma' \vdash t = t' : A'}{\Gamma \vdash t = t' : A} \; TyConv \qquad \frac{\Gamma \vdash \quad \Gamma \vdash A \quad \Gamma \vdash t : A}{\Gamma \vdash (t : A)[id]^{\Gamma}_{\Gamma} = t : A} \; IdSub$$

$$\frac{\Gamma \vdash f : \Delta \quad \Delta \vdash A \quad \Gamma \vdash t : A[f]^{\Gamma}_{\Delta}}{\Gamma \vdash q(\Gamma, A)[\langle f^{\Gamma}_{\Delta}, (t : A)\rangle]^{\Gamma}_{\Delta \cdot A} = (t : A)[f]^{\Gamma}_{\Delta}} \; qAndExt$$

$$\frac{\Gamma \vdash f : \Delta \quad \Delta \cdot A \vdash t : B}{\Gamma \vdash \lambda(\Delta, A, B, t)[f] = \lambda(\Gamma, A[f], B[f^+], t[f^+]) : \Pi(\Delta, A, B)[f]} \; \lambda AndSub$$

$$\frac{\Gamma \vdash f : \Delta \quad \Delta \vdash t : \Pi(\Delta, A, B)}{\Gamma \vdash \mathrm{ap}(\Delta, A, B, t[f]^{\Gamma}_{\Delta}) = \mathrm{ap}(\Gamma, A[f], B[f^+], t)[f^+] : \Pi(\Gamma, A, B)} \; apAndSub$$

$$\frac{\Theta \vdash A \quad \Delta \vdash f : \Theta \quad \Gamma \vdash g : \Delta \quad \Theta \vdash t : A}{\Gamma \vdash (t : A)[f]^{\Delta}_{\Theta}[g]^{\Gamma}_{\Delta} = (t : A)[\mathrm{comp}(\Gamma, \Delta, \Theta, g, f)] : A[f]^{\Delta}_{\Theta}[g]^{\Gamma}_{\Delta}} \; SubComp$$

$$\frac{\Gamma \cdot A \vdash t : \Pi(\Gamma, A, B)}{\Gamma \cdot A \vdash \mathrm{ap}(\Gamma, A, B, \lambda(\Gamma, A, B, t)) = t : B} \; \beta \qquad \frac{\Gamma \cdot A \vdash t : B}{\Gamma \vdash \lambda(\Gamma, A, B, \mathrm{ap}(\Gamma, A, B, t)) = t : \Pi(\Gamma, A, B)} \; \eta$$

The fact that ap and λ are inverse of each other is expressed by the two rules: β and η.

Morphisms

$$\frac{\Gamma \vdash}{\Gamma \vdash \mathrm{id}\,\Gamma : \Gamma} \qquad \frac{\Gamma \vdash}{\Gamma \vdash !_{\Gamma} : \diamond} \qquad \frac{\Gamma \vdash \quad \Gamma \vdash A}{\Gamma \cdot A \vdash p(\Gamma, A) : \Gamma} \qquad \frac{\Gamma \vdash \quad \Delta \vdash \quad \Gamma \vdash f : \Delta \quad \Delta \vdash A \quad \Gamma \vdash t : A[f]}{\Gamma \vdash \langle f^{\Gamma}_{\Delta}, (t : A)\rangle : \Delta \cdot A}$$

$$\frac{\Gamma \vdash g : \Delta \quad \Delta \vdash f : \Theta}{\Gamma \vdash \mathrm{comp}(\Gamma, \Delta, \Theta, g, f) : \Theta} \qquad \frac{\Gamma = \Gamma' \vdash \quad \Delta = \Delta' \vdash \quad \Gamma \vdash f : \Delta}{\Gamma' \vdash f : \Delta'}$$

Morphism equality

$$\frac{\Gamma = \Gamma' \vdash \quad \Delta = \Delta' \vdash \quad \Gamma' \vdash f : \Delta'}{\Gamma \vdash f : \Delta}\ TyConv \qquad \frac{\Gamma \vdash g : \Delta}{\Gamma \vdash g = \mathrm{comp}(\Gamma, \Gamma, \Delta, \mathrm{id}\,\Gamma, g)}\ IdRight$$

$$\frac{\Gamma \vdash g : \Delta}{\Gamma \vdash g = \mathrm{comp}(\Gamma, \Delta, \Delta, g, \mathrm{id}\,\Delta)}\ IdLeft$$

$$\frac{\Gamma \vdash f : \Delta \quad \Delta \vdash g : \Theta \quad \Theta \vdash h :}{\Gamma \vdash \mathrm{comp}(\Gamma, \Delta, , \mathrm{comp}(\Delta, \Theta, , f, g), h) = \mathrm{comp}(\Gamma, \Theta, K, f, \mathrm{comp}(\Gamma, \Delta, \Theta, g, h))}\ CompAssoc$$

$$\frac{\Gamma \vdash g : \diamond}{\Gamma \vdash !_\Gamma = g : \diamond}\ Unicity\varepsilon \qquad \frac{\Gamma \vdash f : \Delta \quad \Delta \vdash A \quad \Gamma \vdash t : A[f]^\Gamma_\Delta}{\Gamma \vdash \mathrm{comp}(\Gamma, \Delta \cdot A, \Delta, \langle f^\Gamma_\Delta, (t : A)\rangle, p(\Delta, A)) = f : \Delta)}\ ProjExt$$

$$\frac{\Gamma \vdash A}{\Gamma \cdot A \vdash \langle p(\Gamma, A), q(\Gamma, A)\rangle = \mathrm{id}(\Gamma \cdot A)}\ ExtId$$

$$\frac{\Gamma \vdash g : \Delta \quad \Delta \vdash f : \Theta \quad \Theta \vdash A \quad \Delta \vdash t : A[f]^\Delta_\Theta}{\Gamma \vdash \mathrm{comp}(\Gamma, \Delta, \Theta \cdot A, g, \langle f, t\rangle_{\Delta,\Theta,A}) = \langle \mathrm{comp}(\Gamma, \Delta, \Theta, g, f), (t : A)[g]^\Gamma_\Delta\rangle}\ ExtComp$$

1.4 Coherence property

Now that we have defined the typing rules of our calculus, we need to first prove a property about it. For instance when we have a term t of type A in a context Γ, it makes only sense if Γ is well-formed context and if A is a formed-type in Γ. That is what states the following lemma:

***Lemma* 1.** — *The following holds:*

- *If* $\Gamma \vdash A$, *then* $\Gamma \vdash$
- *If* $\Gamma \vdash A = A'$ *then* $\Gamma \vdash A$ *and* $\Gamma \vdash A'$
- *If* $\Gamma \vdash t : A$ *then* $\Gamma \vdash A$
- *If* $\Gamma \vdash t = t' : A$ *then* $\Gamma \vdash t : A$ *and* $\Gamma \vdash t : A'$
- *If* $\Gamma \vdash f : \Delta$ *then* $\Gamma \vdash$ *and* $\Delta \vdash$
- *If* $\Gamma \vdash f = f' : \Delta$ *then* $\Gamma \vdash$ *and* $\Delta \vdash$.

Note that Streicher in [Str91] needs to add some annotations in the term to be able to carry out its correctness proof (typing annotations in app and λ-abstraction) and *after* defining the interpretation, proves that they are not needed.

Our approach here is to add much more annotations in the term so that solving the *coherence problem* becomes easy, and then we prove that the annotations are not needed for the calculus to be unambiguous.

Compressing derivations From a typing derivation δ, we can obtain a smaller derivation by *compressing it* in two steps:

- first, compact two type-conversion rules into one by transitivity. For example, the reduction rule for well-formedness of type is:

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma'' \vdash A} \quad \Gamma' = \Gamma''}{\Gamma' \vdash A} \quad \Gamma = \Gamma'}{\Gamma \vdash A} \rightarrow \dfrac{\dfrac{\vdots}{\Gamma'' \vdash A} \quad \Gamma = \Gamma''}{\Gamma \vdash A},$$

and we have similar reduction rules for typing judgement and substitution well-formedness judgement.

- remove instances of type-conversion rules (again this is just the rule for type well-formedness):

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash A} \quad \Gamma = \Gamma}{\Gamma \vdash A} \rightarrow \dfrac{\vdots}{\Gamma \vdash A}$$

Each typing derivation δ has a normal form with respect to these rules (since both of them make the depth decrease) that we note δ^z.

The next lemma is a very characteristic property of our syntax: intuitively this means that we have just enough annotations in terms, to know *exactly* what will be the first rule (apart from type conversion) used in any derivation for this term.

***Lemma* 2.** — *Let J be a typing judgement. If δ and δ' are two derivations of J, then $\delta^z = (\delta')^z$.*

Proof. The proof is made by induction on the depths of δ and δ'. If $\delta^z \neq \delta$ and $\delta'^z \neq \delta'$, then we can use the induction hypothesis on (δ^z, δ'^z) and get the desired result since $^{-z}$ is idempotent. We now assume that $\delta^z = \delta$ and $\delta'^z = \delta'$. We only detail one case, when J is the judgement $\Delta \vdash \lambda(\Gamma, A, B, t) : \Pi(\Theta, C, D)$.

Since $\delta = \delta^z$, δ can only have two forms:

- The last rule is the λ rule

$$\delta = \dfrac{\Delta \vdash \quad \Delta \vdash A \quad \Delta \cdot A \vdash B \quad \Delta \cdot A \vdash t : B}{\Delta \vdash \lambda(\Delta, A, B, t) : \Pi(\Delta, A, B, t)},$$

  and this implies that $\Gamma \equiv \Delta \equiv \Theta$, $A \equiv C$, $B \equiv D$ (remember $\equiv$ denotes syntactical equality). Thus δ^z has to have the same form (since "useless" type conversion rules do not appear in $\delta'^z = \delta'$) and we can conclude by induction.

- The last rule is a type conversion rule, then δ is of the form

$$\delta = \dfrac{\dfrac{\Delta \vdash \qquad \Delta \vdash A \qquad \Delta \cdot A \vdash B \qquad \Delta \cdot A \vdash t : B}{\Delta \vdash \lambda(\Delta, A, B, t) : \Pi(\Delta, A, B, t)} \qquad \Gamma = \Delta \qquad \Gamma \vdash \Pi(\Delta, A, B) = \Pi(\Theta, C, D)}{\Gamma \vdash \lambda(\Delta, A, B, t) : \Pi(\Theta, C, D)}$$

Then, again δ' has to have the same form, and we can apply the hypothesis induction on the judgements $\Delta \vdash$, $\Delta \vdash A$, $\Delta \cdot A \vdash B$, $\Delta \cdot A \vdash t : B$ and the corresponding subderivations of δ and δ'.

□

Interpretation One informal property of a model of type theory is that the definitional equality of the syntax should be interpreted as the equality in the model.

This leads to the definition of interpretation:

Definition 2. — *[Interpretation] Let X be a set and $\varphi : \mathscr{D} \to X$ be map from $\mathscr{D}$, the set of typing derivation inductively generated by the rules of section Rules.*

φ is an interpretation whenever it is invariant by type conversion rules, meaning that if δ is a derivation of $\Gamma \vdash A$, then

$$\varphi(\delta) = \varphi\left(\dfrac{\delta : \Gamma \vdash A \qquad \Gamma = \Gamma' \vdash}{\Gamma' \vdash A}\right)$$

for any derivation of $\Gamma = \Gamma' \vdash$, and so on for the other kinds of judgements ($\Gamma \vdash t : A$ and $\Gamma \vdash f : \Delta$)

Note that for any interpretation $\varphi(\delta) = \varphi(\delta^z)$ since δ and δ^z only differs by applications of type-conversion rules. This leads to the *coherence property* of our calculus:

Lemma 3. — *[Coherence property] Let $\varphi : \mathscr{D} \to X$ be an interpretation. φ induces a unique map $: \mathscr{J} \to X$ from the set $\mathscr{J}$ of derivable typing judgements to X, such that if δ is a derivation of the typing judgement J, $\varphi(\delta) = (J)$.*

Proof. Formally, $\mathscr{J}$ is $\mathscr{D}$ quotiented by the relation "δ and δ' are equivalent iff they share the same conclusion". To have the result, we need to prove that φ is compatible with this relation. If we have two equivalent derivations δ and δ', then by the lemma unicity we have $\delta^z = \delta'^z$ and since φ is an interpretation we have the desired result.

□

2 Semantics

In this section, we define the notion of categories with families well-suited to our calculus. We then prove that the syntax of section Desugarized syntax quotiented by definitional equality is an initial object in this category.

2.1 The category of categories with families

We first define a notion of categories with familes with extra-structure to speak of the type former we have in our calculus.

***Definition* 3.** — *[Category with families] A category with family (CwF) is a couple* $(\mathbb{C}, F)$ *where* $\mathbb{C}$ *is a category, the category of contexts and substitution of the CwF and* $F : \mathbb{C}^{op} \to \textbf{\textit{Fam}}$ *is a contravariant functor.*

Before giving the axioms, we set up a few notations:

- $Ty(\Gamma)$ *is a short-hand for* $F(\Gamma)_1$*, i.e. the index set of the family* $F(\Gamma)$ *(where* $\Gamma \in \mathbb{C}$*)*
- $Tm(\Gamma, A)$ *is a short-hand for* $F(\Gamma)_2(A)$ *i.e. the family* $F(\Gamma)$ *at index* A *(*$\Gamma \in \mathbb{C}$ *and* $A \in Ty(\Gamma)$*)*
- *If* $f : \Gamma \to \Delta$ *is a morphism in* $\mathbb{C}$ *then* $F(f) : F(\Delta) \to F(\Gamma)$ *is a map of families and for* $A \in Ty(\Delta)$ *and* $t \in Tm(\Delta, A)$*, we note* $A\{f\} \in Ty(\Gamma)$ *for* $F(f)_1(A)$ *the image of the index* A *by* $F(f)$*, and* $t\{f\} \in Tm(\Gamma, A\{f\})$ *for* $F(f)_2(t)$

$\mathbb{C}$ *and* F *must satisfy:*

- *there is a terminal element* $1_{\mathbb{C}}$ *in* $\mathbb{C}$*, the empty context,*
- *for each context* $\Delta \in \mathbb{C}$ *and type* $A \in Ty(\Delta)$ *over* Δ*, there is a context* $\Delta \cdot A \in \mathbb{C}$*, along with a morphism* $\boldsymbol{p}_{\Delta,A} : \Delta \cdot A \to \Delta$ *and a term* $\boldsymbol{q}_{\Delta,A}$*, satisfying the following universal property. For each substitution* $f : \Gamma \to \Delta$ *and each term* $t \in Tm(\Gamma, A\{f\})$ *there is a unique* $\langle f, t\rangle : \Gamma \to \Delta \cdot A$ *such that* $p \circ \langle f, t\rangle = f$ *and* $\boldsymbol{q}\{\langle f, t\rangle\} = t$*.*

This is the basic notion of CwF that takes into account the combinatorics of dependent type theory. We should add constructions to reflect type formers. We use $x : A \to B(x)$ for the dependent product on the meta-theory, and as in the syntax, for any $f : \Gamma \to \Delta$ the notation f^+ will mean the map $\langle f \circ p, q\rangle : \Gamma \cdot A\{f\} \to \Delta \cdot A$ for any type $A \in Ty\Delta$.

***Definition* 4.** — *[Categories with families for the Logical Framework] A category with families for the Logical Framework (CwFLF) is a CwF* $(\mathscr{C}, F)$*. along with:*

- *a function* $\textbf{\textit{set}} : (\Gamma : |\mathscr{C}|) \to Ty\Gamma$ *reflecting the universe*
- *a function* $\textbf{\textit{el}} : (\Gamma : |\mathscr{C}|) \to Tm(\Gamma, \textbf{\textit{set}}(\Gamma)) \to Ty\Gamma$
- *a function* $\Pi : (\Gamma : |\mathscr{C}|) \to (A : Ty\Gamma) \to Ty(\Gamma \cdot A) \to Ty\Gamma$
- *a function* $\lambda : (\Gamma : |\mathscr{C}|) \to (A : Ty\Gamma) \to (B : Ty(\Gamma \cdot A)) \to Tm(\Gamma \cdot A, B) \to Tm\Gamma(\Pi(\Gamma, A, B))$
- *a function* $\textbf{\textit{ap}} : (\Gamma : |\mathscr{C}|) \to (A : Ty\Gamma) \to (B : Ty(\Gamma \cdot A)) \to Tm(\Gamma, B) \to Tm\Gamma(\Pi(\Gamma, A, B))$

subject to the following hypothesis:

- $\textbf{\textit{set}}(\Delta)\{f\} = \textbf{\textit{set}}(\Gamma)$ *for* $f : \Gamma \to \Delta$ *in* $|\mathscr{C}|$
- $\textbf{\textit{el}}(\Delta, x)\{f\} = \textbf{\textit{el}}(\Gamma, x\{f\})$ *for* $f : \Gamma \to \Delta$ *in* $|\mathscr{C}|$
- $\Pi(\Delta, A, B)\{f\} = \Pi(\Gamma, A\{f\}, B\{f^+\})$ *for* $f : \Gamma \to \Delta$ *in* $|\mathscr{C}|$

- $\lambda(\Delta, A, B, t)\{f\} = \lambda(\Gamma, A\{f\}, B\{f^+\}, t\{f^+\})$ *for* $f : \Gamma \to \Delta$ *in* $|\mathscr{C}|$
- $\boldsymbol{ap}(\Gamma, A\{f\}, B\{f^+\}, t\{f\}) = \boldsymbol{ap}(\Delta, A, B, t)\{f^+\}$ *for* $f : \Gamma \to \Delta$ *in* $|\mathscr{C}|$
- $\boldsymbol{ap}(\Gamma, A, B, \lambda(\Gamma, A, B, t)) = t$ *for any* $t \in Tm(\Gamma \cdot A, B)$
- $\lambda(\Gamma, A, B, \boldsymbol{ap}(\Gamma, A, B, t)) = t$ *for any* $t \in Tm(\Gamma, \Pi(\Gamma, A, B))$

As before, we may skip some arguments that can be inferred from the context.
We can know define the categorical structure on CwFLFs [Dyb96] :

***Definition* 5.** — *A CwF-morphism from a CwFLF* $(\mathscr{C}, F)$ *to a CwFLF* $(\mathscr{D}, G)$ *is given by a functor* $\Phi : \mathscr{C} \to \mathscr{D}$ *and a natural transformation* $\eta : F \Rightarrow G \circ \Phi$ *such that "structure is preserved". We note* $\eta_\Gamma : Ty_{\mathscr{C}}(\Gamma) \to Ty_{\mathscr{D}}(\Phi(\Gamma))$ *and* $\eta_{\Gamma,A} : Tm_{\mathscr{C}}(\Gamma, A) \to Tm_{\mathscr{D}}(\Phi(\Gamma), \eta_\Gamma(A))$ *for the components of* η. Φ *and* η *must satisfy:*

- $\Phi(1_{\mathscr{C}}) = 1_{\mathscr{D}}$ *and* $\Phi(\Gamma \cdot A) = \Phi(\Gamma) \cdot \eta_\Gamma(A)$
- $\Phi(p : \Gamma \cdot A \to \Gamma) = (p : \Phi(\Gamma \cdot A) \to \Phi(\Gamma))$
- $\eta_{\Gamma \cdot A, A}(q) = q \in Tm(\Gamma \cdot A, A)$
- $\eta_\Gamma(\boldsymbol{set}\Gamma) = \boldsymbol{set}\Phi(\Gamma)$ *and so on for the other type constructions*

One can compose these arrows point-wise, and the identity functor along with the identity natural transformation gives CwFLFs and arrows between CwFLFs a structure of category.

2.2 The term model

In this section, we build the model of terms as a CwFLFs.

***Definition* 1.** — *The model of terms* $\mathbb{T}$ *is given by:*

- *Objects of* $\mathbb{T}$ *:* $\{\Gamma | \Gamma \vdash\} / =^c$ *where* $\Gamma =^c \Gamma'$ *if* $\Gamma = \Gamma' \vdash$ *is derivable.*
- *Maps from* $[\Gamma]$ *to* $[\Delta]$ *:* $\{f | \Gamma \vdash f : \Delta\} / =^\Gamma_\Delta$ *where* $f =^\Gamma_\Delta g$ *iff* $\Gamma \vdash f = g : \Delta$ *is derivable. Note that this makes sense and only depends on the equivalence class of* Γ *in* $|\mathbb{T}|$ *because of type conversion rules.*
- *Types over* $[\Gamma]$ *:* $\{A | \Gamma \vdash A\} / =^\Gamma$ *where* $A =^\Gamma B$ *if* $\Gamma \vdash A = B$
- *Terms over* $[\Gamma]$ *of type* $[A]$ *:* $\{t | \Gamma \vdash t : A\} / =^\Gamma_A$ *where* $t =^\Gamma_A t'$ *if* $\Gamma \vdash t = t' : A$.

The CwFLFs constructions on $\mathbb{T}$ *are defined without any surprise, for instance* $\lambda([\Gamma], [A], [B], [t]) = [\lambda(\Gamma, A, B, t)]$ *and so on. Type conversion rules make it a good definition.*

It is easy easy to check that $\mathbb{T}$ is a CwFLF.

2.3 Interpretation in any CwFLF

The goal of this section is to build a partial interpretation from the pre-syntax to any CwFLF. Let $(\mathscr{C}, F)$ be a CwFLF. We build an interpretation of derivations by induction: a derivation δ will be mapped to $[\![\delta]\!]$.

Because at the definition time we don't know yet that $[\![\cdot]\!]$ is an interpretation, we cannot have a strong invariant on where the interpreted objects will live. To solve this, we define $[\![\cdot]\!]$ partially and then show that it is always defined. In the following, we will write expressions that might be ill-defined. If so, the interpretation of the derivation is considered to be undefined.

The rules for $[\![\cdot]\!]$ is given on figure 1.

The following lemma proves the correctness of the interpretation:

***Lemma* 4.** — *The map $[\![\cdot]\!]$ is an interpretation. We note $[\![\cdot]\!]$ as well the induced map. We have*

- *For any derivation $\delta : \Gamma \vdash$, $[\![\delta]\!]$ is defined and is an object of $\mathscr{C}$*
- *For any derivation $\delta : \Gamma \vdash f : \Delta$, $[\![\delta]\!]$ is defined and is a map from $[\![\Gamma \vdash]\!]$ to $[\![\Delta \vdash]\!]$*
- *For any derivation $\delta : \Gamma \vdash A$, $[\![\delta]\!]$ is defined and lives in $Ty([\![\Gamma \vdash]\!])$*
- *For any derivation $\delta : \Gamma \vdash t : A$, $[\![\delta]\!]$ is defined and lives in $Tm([\![\Gamma \vdash]\!], [\![\Gamma \vdash A]\!])$*

Proof. By induction.

□

2.4 Unicity of the interpretation

In this section, we prove that the term model is initial in the category of categories with families, and that the unique morphism going from the term model to any CwFLF is induced by the interpretation.

Let $(\mathscr{C}, F)$. The morphism from $\mathbb{T}$ to $\mathscr{C}$ is given on contexts by $[\vdash \Gamma] \mapsto [\![\vdash \Gamma]\!]$. It is well-defined precisely because $[\![^-]\!]$ is an interpretation. The definition on morphisms/types/terms is done likewise. The following lemma proves that it is the unique map from $\mathbb{T}$ to $\mathscr{C}$:

***Lemma* 5.** — *[Uniqueness lemma] Let (φ, η) and (ψ, ι) be two maps of CwFLFs from $\mathbb{T}$ to a CwFLF $(\mathscr{C}, F)$. Then*

- *If $\vdash \Gamma$ is provable, then $\varphi([\vdash \Gamma]) = \psi([\vdash \Gamma])$*
- *If $\Gamma \vdash f : \Delta$ is provable then $\varphi([\vdash \Gamma]) = \psi([\vdash \Gamma])$, $\varphi([\vdash \Delta]) = \psi([\vdash \Delta])$ and $\varphi([\Gamma \vdash f : \Delta]) = \psi([\Gamma \vdash f : \Delta])$*
- *If $\Gamma \vdash A$ is provable, then $\varphi([\vdash \Gamma]) = \psi([\vdash \Gamma])$, ${\eta_{[\Gamma]}}^1([\Gamma \vdash A]) = {\iota_{[\Gamma]}}^1([\Gamma \vdash A])$*
- *If $\Gamma \vdash t : A$ is provable, then $\varphi([\vdash \Gamma]) = \psi([\vdash \Gamma])$, ${\eta_{[\Gamma]}}^1([\Gamma \vdash A]) = {\iota_{[\Gamma]}}^1([\Gamma \vdash A])$ and ${\eta_{[\Gamma]}}^2([\Gamma \vdash A], [\Gamma \vdash t : A]) = {\iota_{[\Gamma]}}^2([\Gamma \vdash A], [\Gamma \vdash t : A])$*

Proof. In the proof we will be using φ and ψ where we should have used η and ι to make it easier to follow. The proof is by induction on derivations. We detail only the lambda-abstraction case (explicit derivation)

$$[\![\varepsilon \vdash]\!] = 1_{\mathscr{C}} \qquad \left[\!\!\left[\frac{\delta_\Gamma :\ \Gamma \vdash \qquad \delta_A :\ \Gamma \vdash A}{\Gamma \cdot A \vdash}\right]\!\!\right] = [\![\delta_\Gamma]\!] \cdot [\![\delta_A]\!] \qquad \left[\!\!\left[\frac{\Gamma \vdash \mathrm{set}\,\Gamma}{\delta_\Gamma :\ \Gamma \vdash}\right]\!\!\right] = \mathbf{set}[\![\delta_\Gamma]\!]$$

$$\left[\!\!\left[\frac{\delta_\Gamma :\ \Gamma \vdash \qquad \delta_x :\ \Gamma \vdash x : \mathrm{set}}{\Gamma \vdash \mathrm{el}(\Gamma, x)}\right]\!\!\right] = \mathbf{el}([\![\delta_\Gamma]\!], [\![\delta_A]\!])$$

$$\left[\!\!\left[\frac{\delta_\Gamma :\ \Gamma \vdash \qquad \delta_A :\ \Gamma \vdash A \qquad \delta_B :\ \Gamma \cdot A \vdash B}{\Gamma \vdash \mathrm{el}(\Gamma, x)}\right]\!\!\right] = \Pi([\![\delta_\Gamma]\!], [\![\delta_A]\!], [\![\delta_B]\!])$$

$$\left[\!\!\left[\frac{\Gamma \vdash \qquad \Delta \vdash \qquad \delta_A :\ \Gamma \vdash A \qquad \delta_f :\ \Delta \vdash f : \Gamma}{\Delta \vdash A[f]^\Gamma_\Delta}\right]\!\!\right] = [\![\delta_A]\!]\{\delta_f\} \qquad \left[\!\!\left[\frac{\delta_\Gamma :\ \Gamma \vdash}{\Gamma \vdash \mathrm{id}\,\Gamma : \Gamma}\right]\!\!\right] = \mathrm{id}_{[\![\delta_\Gamma]\!]}$$

$$\left[\!\!\left[\frac{\delta_\Gamma :\ \Gamma \vdash}{\Gamma \vdash \varepsilon\Gamma : \varepsilon}\right]\!\!\right] = !_{[\![\delta_\Gamma]\!]} \qquad \left[\!\!\left[\frac{\delta_0 :\ \Gamma' \vdash A \qquad \Gamma = \Gamma' \vdash}{\Gamma \vdash A}\right]\!\!\right] = [\![\delta_0]\!]$$

$$\left[\!\!\left[\frac{\Gamma \vdash \qquad \Delta \vdash \qquad \Theta \vdash \qquad \delta_g :\ \Gamma \vdash g : \Delta \qquad \delta_f :\ \Delta \vdash f : \Theta}{\Gamma \vdash \circ(\Gamma, \Delta, \Theta, f, g) : \Theta}\right]\!\!\right] = [\![\delta_f]\!] \circ [\![\delta_g]\!]$$

$$\left[\!\!\left[\frac{\delta_\Gamma :\ \Gamma \vdash \qquad \delta_A :\ \Gamma \vdash A}{\Gamma \cdot A \vdash p : \Gamma}\right]\!\!\right] = p([\![\delta_\Gamma]\!], [\![\delta_A]\!])$$

$$\left[\!\!\left[\frac{\Gamma \vdash \qquad \Delta \vdash \qquad \Delta \vdash A \qquad \delta_f :\ \Gamma \vdash f : \Delta \qquad \delta_t :\ \Gamma \vdash t : A[f]}{\Gamma \vdash \langle f, t\rangle : \Delta \cdot A}\right]\!\!\right] = \langle [\![\delta_f]\!], [\![\delta_t]\!]\rangle$$

$$\left[\!\!\left[\frac{\delta_0 :\ \Gamma' \vdash f : \Delta' \qquad \Gamma = \Gamma' \vdash \qquad \Delta = \Delta' \vdash}{\Gamma \vdash f : \Delta}\right]\!\!\right] = [\![\delta_0]\!]$$

$$\left[\!\!\left[\frac{\delta_\Gamma :\ \Gamma \vdash \qquad \delta_A :\ \Gamma \vdash A}{\Gamma \cdot A \vdash q(\Gamma, A) : A}\right]\!\!\right] = q([\![\delta_\Gamma]\!], [\![\delta_A]\!])$$

$$\left[\!\!\left[\frac{\delta_\Gamma :\ \Gamma \vdash \qquad \delta_A :\ \Gamma \vdash A \qquad \delta_B :\ \Gamma \cdot A \vdash B \qquad \delta_t :\ \Gamma \cdot A \vdash t : B}{\Gamma \vdash \lambda t : A \to_\Gamma B}\right]\!\!\right] = \lambda([\![\delta_\Gamma]\!], [\![\delta_A]\!], [\![\delta_B]\!], [\![\delta_t]\!])$$

$$\left[\!\!\left[\frac{\delta_\Gamma :\ \Gamma \vdash \qquad \delta_A :\ \Gamma \vdash A \qquad \delta_B :\ \Gamma \cdot A \vdash B \qquad \delta_t :\ \Gamma \vdash t : A \to_\Gamma B}{\Gamma \cdot A \vdash \mathrm{ap}(t) : B}\right]\!\!\right] = \mathbf{ap}([\![\delta_\Gamma]\!], [\![\delta_A]\!], [\![\delta_B]\!], [\![\delta_t]\!])$$

$$\left[\!\!\left[\frac{\Gamma \vdash \qquad \Delta \vdash \qquad \Gamma \vdash A \qquad \delta_t :\ \Gamma \vdash t : A \qquad \delta_f :\ \Delta \vdash f : \Gamma}{\Delta \vdash (t : A)[f]^\Gamma_\Delta : A[f]^\Gamma_\Delta}\right]\!\!\right] = [\![\delta_t]\!]\{[\![\delta_f]\!]\}$$

$$\left[\!\!\left[\frac{\delta_0 :\ \Gamma' \vdash t : A' \qquad \Gamma = \Gamma' \qquad \Gamma \vdash A = A'}{\Gamma \vdash t : A}\right]\!\!\right] = [\![\delta_0]\!]$$

Figure 1: Definition of the interpretation

$$\frac{\Gamma \vdash A \quad \Gamma \cdot A \vdash B \quad \Gamma \cdot A \vdash t : B}{\Gamma \vdash \lambda(\Gamma, A, B, t) : \Pi(\Gamma, A, B)}$$

By induction we know that $\varphi([\Gamma]) = \psi([\Gamma])$, $\varphi([\Gamma \vdash A]) = \psi([\Gamma \vdash A])$ and $\varphi([\Gamma \cdot A \vdash B])$. Thus we have $\varphi([\Gamma \vdash \Pi(\Gamma, A, B)]) = \varphi([\Gamma \vdash A]) \to \varphi([\Gamma \cdot A \vdash B]) = \psi([\Gamma \vdash A]) \to \psi([\Gamma \cdot A \vdash B]) = \psi([\Gamma \vdash \Pi(\Gamma, A, B)])$. And finally

$$\begin{aligned}
&\varphi([\Gamma \vdash \lambda(\Gamma, A, B, t) : \Pi(\Gamma, A, B)]) \\
&= \varphi(\lambda^{\mathbb{T}}([\Gamma], [\Gamma \vdash A], [\Gamma \cdot A \vdash B], [\Gamma \cdot A \vdash t : B])) && \text{by def. of the } \lambda \text{ on } \mathbb{T} \\
&= \lambda^{\mathscr{C}}(\varphi([\Gamma]), \varphi([\Gamma \vdash A]), \varphi([\Gamma \cdot A \vdash B]), \varphi(\Gamma \cdot A \vdash t : B)) && \text{because } \varphi \text{ preserves } \lambda \\
&= \lambda^{\mathscr{C}}(\psi([\Gamma]), \psi([\Gamma \vdash A]), \psi([\Gamma \cdot A \vdash B]), \psi(\Gamma \cdot A \vdash t : B)) && \text{by induction hypothesis} \\
&= \psi([\Gamma \vdash \lambda(\Gamma, A, B, t) : \Pi(\Gamma, A, B)]) && \text{rolling back}
\end{aligned}$$

□

***Theorem* 1.** — *$\mathbb{T}$ is an initial object in the category of CwFLFs.*

3 Regular syntax

In this section, we define an explicit syntax for type theory without annotations and we prove that these two syntax are equivalent (meaning that the two term models are isomorphic).

3.1 Syntax

The raw syntax of the regular syntax is as follows:

$$\begin{aligned}
\Gamma, \Delta &::= \diamond && | \ \Gamma \cdot A \\
A, B &::= \mathrm{set} && | \ \Pi(A, B) \\
&| \ \mathrm{el}(x) && | \ A[f] \\
t, u &::= q && | \ \lambda t \\
&| \ \mathrm{ap}(t) | && t[f] \\
f, g &::= \mathrm{id} && | \ p \\
&| \ \mathrm{comp}(f, g) && | \ ! \\
&| \ \langle f, t \rangle
\end{aligned}$$

One can also define the judgements corresponding to this raw syntax along the lines of Rules. To distinguish the two sets of judgements, we use the symbol $\vdash^i$ for judgements in this implicit syntax. Given a annotated context Γ (resp. type A, term t, substitution f) we note $s(\Gamma)$ (resp. $s(A)$, $s(t)$, $s(f)$) the corresponding object in the traditional syntax where annotations are left out. s is called the stripping operator.

The first property to notice is that s preserves typing.

3.2 Normalization

To prove that s can be viewed as an isomorphism of CwFLFs, we need to know the normalization of our annotated calculus.[4]

A proof of normalization for a similar calculus can be found in [ACD07]. We believe that this proof could easily be extended to our calculus.

Normal forms look like (annotations are left out for readability)

$n ::=$		**normal forms**
	$\mid k$	Neutral terms
	$\mid \lambda(n)$	Lambda-abstractions
$k ::=$		**neutral terms**
	$\mid q[p^i]$	Variables
	$\mid \mathrm{ap}(k)\langle p, n\rangle$	Applications
$N, N' ::=$		**normal types**
	$\mid \mathrm{set}$	Universe of small types
	$\mid \Pi(N, N')$	Dependent product
	$\mid \mathrm{el}(n)$	Elements of a small type

The normalization lemma states that

***Lemme* 1.** — *Let t be a term such that $\Gamma \vdash t : A$. There exists a term $nf(t)$ such that*

- *$nf(t)$ is normal,*
- $\Gamma \vdash t = nf(t) : A$,
- *If t' is such that $\Gamma \vdash t = t' : A$, then $\Gamma \vdash nf(t) = nf(t') : A$,*
- *If $s(t) \equiv s(t')$ then $s(nf(t)) \equiv s(nf(t'))$*

3.3 Injectivity of s

We first show that s is injective with respect to definitional equality, meaning that:

***Lemme* 2.** — *[Injectivity of the stripping] We have:*

- *If $s(\Gamma) = s(\Gamma') \vdash$ and $\Gamma, \Gamma' \vdash$, then $\Gamma = \Gamma' \vdash$*
- *If $s(\Gamma) \vdash^i s(A) = s(A')$ and $\Gamma \vdash A, A'$ then $\Gamma \vdash A = A'$*
- *If $s(\Gamma) \vdash^i s(t) = s(t') : s(A)$ and$\Gamma \vdash t, t' : A$ then $\Gamma \vdash t = t' : A$*
- *If $s(\Gamma) \vdash^i s(f) = s(g) : s(\Delta)$ and $\Gamma \vdash f, g : \Delta$ then $\Gamma \vdash f = g : \Delta$.*

Proof. By induction on the equality derivation. All cases are straightforward but the reflexivity case. This is where normalization is needed. The proof, omitted can be found in [Str91] (for a different calculus but the idea is the same).

□

[4] Normalization plays an important role in this proof, as it is not valid anymore in non-normalizing calculus, for instance Girard's U^-, see [BC06]

3.4 Surjectivity of s

***Lemme* 3.** — *We have*

- *If* $\Gamma \vdash^i$, *there exists a* Γ^0 *in the annotated syntax such that* $\Gamma^0 \vdash$ *and* $s(\Gamma) = \Gamma^0$
- *If* $\Gamma \vdash^i A$, *there exists a* Γ^0 *and an* A^0 *in the annotated syntax such that* $\Gamma^0 \vdash A^0$ *and* $s(\Gamma^0) = \Gamma$ *and* $s(A^0) = A$
- *If* $\Gamma \vdash^i t : A$, *there exists a* Γ^0, t^0, A^0 *in the annotated syntax such that* $s(\Gamma^0) = \Gamma, s(A^0) = A, s(t^0) = t$ *and* $\Gamma^0 \vdash t^0 : A^0$.
- *If* $\Gamma \vdash^i f : \Delta$, *there exists a* Γ^0, f^0, Δ^0 *in the annotated syntax such that* $s(\Gamma^0) = \Gamma, s(f^0) = f, s(\Delta^0) = \Delta$ *and* $\Gamma^0 \vdash f^0 : \Delta^0$.

Proof. Straightforward with the lemma inj_strip.

□

3.5 $\mathbb{T}$ and $\mathbb{T}^i$ are isomorphic

It is easy to see (thanks to lemma inj_strip that s can be extended to an injective CwF morphism $s : \mathbb{T} \to \mathbb{T}^i$. (By initiality we know that this is the the interpretation).

We can build the inverse t with the lemma surj_strip:

- For $\Gamma \in |\mathbb{T}^i|$, $t(\Gamma)$ [U+225D] Γ^0
- For $\Gamma \in |\mathbb{T}^i|$ and $A \in Ty(\Gamma)$, we have $\Gamma^1 \vdash A^1$ with $s(\Gamma^1) \equiv \Gamma$ and $s(A^1) \equiv A$. Since $s(t(\Gamma))$ is Γ, we have $s(t(\Gamma)) = s(\Gamma^1) \vdash^i$ derivable so that by inj_strip $t(\Gamma) = \Gamma^1 \vdash$. Thus $A^1 \in Ty(t(\Gamma))$ and we can define $t(\Gamma, A)$ [U+225D] A^1
- and so on for terms and substitutions.

This defines a CwF morphism $t : \mathbb{T}^i \to \mathbb{T}$. It is clear that $s \circ t = \mathrm{id}_{\mathbb{T}^i}$, and by initiality we have $t \circ s = \mathrm{id}_{\mathbb{T}}$, thus the following

***Theorem* 2.** — *[Equivalence of the annotated syntax with the regular syntax] The stripping operator is an isomorphism between* $\mathbb{T}$ *and* $\mathbb{T}^i$.

Conclusion

In this document, we have shown a new way to solve the coherence problem by enriching the syntax in a systematic way (contrary to Streicher's approach that was somewhat *ad-hoc*). Two initial CwFs have been built in this document (fully annotated term model and regular term model), and it might be possible to build a third one, the CwF of semantic normal forms, a structure that appears in proof by normalization by evaluation [ACD07]. Using the initiality result, one could greatly simplify the normalization proof, since the normalization procedure nbe could be seen as a map of CwF between the term model and itself, thus being equal to the identity, meaning that the definitional equality $t = \mathrm{nbe}(t)$ is derivable.

References

[ACD07] Andreas Abel, Thierry Coquand, and Peter Dybjer. Normalization by evaluation for Martin-Löf Type Theory with typed equality judgements. In *22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wroclaw, Poland, Proceedings*, pages 3–12. IEEE Computer Society Press, 2007.

[BC06] Gilles Barthe and Thierry Coquand. Remarks on the equational theory of non-normalizing pure type systems. *J. Funct. Program.*, 16(2):137–155, 2006.

[BdMDdm73] N.G. Bruijn and Université de Montréal. Département de mathématiques. *Automath: a language for mathematics*. Les Presses de L'Université de Montréal, 1973.

[Car85] John Cartmell. Generalised algebraic theories and contextual categories. 1985.

[CD11] Pierre Clairambault and Peter Dybjer. The biequivalence of locally cartesian closed categories and martin-löf type theories. *CoRR*, abs/1112.3456, 2011.

[Cur93] P.L. Curien. Substitution up to isomorphism. *Fundamenta Informaticae*, 19(1-2):51–85, 1993.

[Dyb96] P. Dybjer. Internal type theory. *Types for Proofs and Programs*, pages 120–134, 1996.

[Str91] T. Streicher. *Semantics of type theory: correctness, completeness, and independence results*. Birkhauser Boston Inc., 1991.