Notes on Categories with Families

Jonathan Chan

Contents

1 Introduction

Categories with families (CwFs) [Dybjer, 1996] are a framework for giving a semantic model of a type theory, typically one that resembles some form of Martin-Löf Type Theory,[1] that closely mirrors its syntactic structure. A CwF itself is not a model, but rather a signature into which both the syntax and the semantics fit, and provides a more succinct justification of the interpretation from the syntax into the semantics. To prove, for instance, the consistency of a type theory, we would proceed roughly as follows:

1. Starting from the definition of a CwF, extend it with the desired type structures as a presentation of a *generalized algebraic theory* (GAT) [Cartmell, 1978]. These form a category of CwFs with structures.

2. Construct from this the syntax of a type theory, with which a CwF called the *term model* can be defined.

3. Construct an interpretation function from the term model to an arbitrary CwF, and show that the interpretation is *sound*. This implies that the term model is initial in the category of CwFs with structures [Bezem et al., 2021].

4. Provide a semantic model of the type theory as another CwF.

5. Show that consistency of the type theory corresponds to some property that holds in the semantic model; this should follow from initiality.

The semantic model can be defined in a variety of suitable mathematical theories: it could be a set-theoretic model, a type-theoretic model, a realizability model, and so on. **TODO**: Be more specific and cite.

This note is aimed towards those familiar with the usual formulations of and concepts in type theory. Even so, some minimal category-theoretic knowledge is needed to begin promptly. The reader should be comfortable with the definitions of categories, functors, initial and terminal objects, and natural transformations, although the latter will covered in a little detail when needed. All further category-theoretic concepts will be defined in due time.

2 Components of a CwF

As a terse summary, a category with families consists of the following:

- A category $\mathcal{C}$ with a terminal object $\bullet$ and morphisms $\langle\rangle_\Gamma : \Gamma \to \bullet$ for every $\Gamma \in \mathcal{C}$;

[1] Uemara [2021] summarizes the kinds of type theories that cannot be modelled as *categories with representable maps* (CwRs):

> Although the notion of a CwR covers a wide range of type theories including Martin-Löf type theory, univalent type theory, and cubical type theory, it cannot cover some important type theories. Characteristics of type theories considered in [Uemara [2021]] are that contexts are single-layered and that assumptions in a context can be used at any time, any number of times. Type theories with dual-contexts are not of this sort. Substructural type theories such as linear logic are out of scope since assumptions can be used a limited number of times.

As CwRs are more or less extensions of CwFs, the same limitations apply to CwFs as well.

- A *contravariant functor* $T : \mathcal{C}^{\mathrm{op}} \to \mathbf{Fam}$ to the *category of families* with objects $T(\Gamma) = \{\mathrm{Tm}_{\mathcal{C}}(\Gamma, A)\}_{A \in \mathrm{Ty}_{\mathcal{C}}(\Gamma)}$;
- A *comprehension* corresponding to the *universal element* of a particular contravariant functor $S_{\Gamma,A} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Fam}$ defined over $\Gamma \in \mathcal{C}, A \in \mathrm{Ty}_{\mathcal{C}}(\Gamma)$; and
- Type formers, induction and elimination forms, and computation rules, as well as uniqueness rules if needed.

In the following subsections, I detail what each of these pieces is meant to mean in the context of type theory.

2.1 The Category of Contexts

- The objects of $\mathcal{C}$ represents typing contexts, with the terminal object $\bullet$ representing the empty context. The morphisms of $\mathcal{C}$ represents sequences of substitutions.
- Given contexts $\Delta, \Gamma \in \mathcal{C}$, the morphism $\gamma : \Delta \to \Gamma$ represents a substitution of the variables in Γ by terms that are well typed under the context Δ. As an example, if $\Gamma = x_1 : A_1, \ldots, x_n : A_n$, the morphism $\gamma = \langle a_1, \ldots, a_n \rangle$ represents the substitution $\_[x_1 \mapsto a_1] \ldots [x_n \mapsto a_n]$, where $\Delta \vdash a_1 : A_1, \ldots, \Delta \vdash a_n : A_n[x_1 \mapsto a_1] \ldots [x_{n-1} \mapsto a_{n-1}]$. Applying a substitution to a term or a type, then, changes its typing context from Γ to Δ; we'll see shortly how performing the substitution is defined.
- The empty substitution $\langle\rangle_\Gamma : \Gamma \to \bullet$ performs no substitutions and weakens the context into Γ.
- Given a context Γ, the identity substitution $\mathrm{id}_\Gamma : \Gamma \to \Gamma$ performs the trivial substitutions $[x_i \mapsto x_i]$.
- Given substitutions $\gamma : \Delta \to \Gamma, \delta : \Xi \to \Delta$, their composition $\gamma \circ \delta : \Xi \to \Gamma$ performs first the substitutions in γ, followed by the substitutions in δ.

2.2 Types and Terms as a Contravariant Functor

To understand the structure of T, we first need the definition of the category of families.

Definition 2.1 (Category **Fam**)**.**

- The objects of **Fam** are families of sets $\{U_x\}_{x \in X}$. In type theory, we would write a dependent pair $(x : X) \times U_x$, where $X : \mathsf{Set}, U : X \to \mathsf{Set}$.
- The morphisms of **Fam** from $\{U_x\}_{x \in X}$ to $\{V_y\}_{y \in Y}$ are pairs of a function $f : X \to Y$ and a family of functions $(g_x)_{x \in X}$ where $g_x : U_x \to V_{f(x)}$. In type theory, we would write a dependent pair $(f : X \to Y) \times ((x : X) \to U_x \to V_{f(x)})$. We recover the function $(x : X) \times U_x \to (y : Y) \times V_y$ using the above functions f and g given the pair (x, u) by constructing the pair $(f(x), g_x(u))$.
- The identity morphisms for $\{U_x\}_{x \in X}$ are pairs of identity functions, *i.e.* $(\lambda x.\, x, \lambda x.\, \lambda u.\, u)$.
- The composition of two morphisms of **Fam** $(f_2, g_2) \circ (f_1, g_1)$ are pairs of the function compositions, *i.e.* $(\lambda x.\, f_2(f_1(x)), \lambda x.\, \lambda u.\, g_{2,f_1(x)}(g_{1,x}(u)))$.

Exercise 2.2. Verify that the left and right identity laws hold for **Fam**.

We can now dissect what T consists of.

- T maps contexts $\Gamma \in \mathcal{C}$ to a set $\mathrm{Ty}_{\mathcal{C}}(\Gamma)$, representing well-formed types under Γ, indexing over a set $\mathrm{Tm}_{\mathcal{C}}(\Gamma, A)$, representing well-typed terms of type $A \in \mathrm{Ty}_{\mathcal{C}}(\Gamma)$. I omit the subscript $_{\mathcal{C}}$ when the category of contexts is clear from context. In short, $T(\Gamma)$ is the family $\{\mathrm{Tm}(\Gamma, A)\}_{A \in \mathrm{Ty}(\Gamma)}$. Alternatively, we can view these as functions $\mathrm{Ty} : \mathcal{C} \to \mathsf{Set}$ and $\mathrm{Tm} : (\Gamma : \mathcal{C}) \to \mathrm{Ty}(\Gamma) \to \mathsf{Set}$.
- T maps substitutions $\gamma : \Delta \to \Gamma$ to morphisms $T(\gamma) : T(\Gamma) \to T(\Delta)$; T is a *contravariant* functor, so Δ and Γ are swapped. ($\mathcal{C}^{\mathrm{op}}$ is merely notation for the contravariance of T, and we aren't actually considering the meaning of the dual category of $\mathcal{C}$.) By Definition 2.1, $T(\gamma)$ consists of a pair of a function $\_[\gamma] : \mathrm{Ty}(\Gamma) \to \mathrm{Ty}(\Delta)$ and a family of functions $\{\_[\gamma]\}_{A \in \mathrm{Ty}(\Gamma)}$ where $\_[\gamma] : \mathrm{Tm}(\Gamma, A) \to \mathrm{Tm}(\Delta, A[\gamma])$, omitting the subscript A and using the same notation for both functions since it should be clear from context whether it acts on a type or a term. These represent applying substitutions to types under Γ and to terms of type A under Γ. In short, given $A \in \mathrm{Ty}(\Gamma)$ and $a \in \mathrm{Tm}(\Gamma, A)$, we have $\Delta \vdash a[\gamma] : A[\gamma]$.

Note that we aren't *defining* a particular functor T, merely describing what such a functor means, and defining new notation $\mathrm{Ty}, \mathrm{Tm}, \_[\_], \_[\_]$ for its components.

Exercise 2.3. State the functor laws for the (contravariant) functorial action of T on the composition of substitutions, one for types and one for terms. Does it make sense which substitution is performed first?

2.3 Context Comprehensions

Consider the following contravariant functor $\mathcal{C}^{\mathrm{op}} \to \mathbf{Fam}$, defined for some $\Gamma \in \mathcal{C}$ and $A \in \mathrm{Ty}(\Gamma, A)$. (This functor seems to go unnamed in the literature; I will call it the comprehension functor over Γ and A, and name it $S_{\Gamma,A}$.)

Definition 2.4 (Comprehension functor $S_{\Gamma,A}$)**.**

- $S_{\Gamma,A}$ maps contexts $\Delta \in \mathcal{C}$ to the family $\{\mathrm{Tm}(\Delta, A[\gamma])\}_{\gamma:\Delta\to\Gamma}$. In type theory, we would write $S_{\Gamma,A}(\Delta) \triangleq (\gamma : \Delta \to \Gamma) \times \mathrm{Tm}(\Delta, A[\gamma])$.
- Given $\Xi, \Delta \in \mathcal{C}$, $S_{\Gamma,A}$ maps substitutions $\delta : \Xi \to \Delta$ to a pair of a function $(\Delta \to \Gamma) \to (\Xi \to \Gamma)$ defined by $S_{\Gamma,A}(\delta)(\gamma) \triangleq \gamma \circ \delta$ and a family of functions over $\gamma : \Delta \to \Gamma$ in $\mathrm{Tm}(\Delta, A[\gamma]) \to \mathrm{Tm}(\Xi, A[\gamma \circ \delta])$ defined by $S_{\Gamma,A}(\delta)(\gamma)(a) \triangleq a[\delta]$. In type theory, we would write $S_{\Gamma,A}(\delta)(\gamma, a) \triangleq (\gamma \circ \delta, a[\delta])$.

This definition is a little convoluted, but we won't need it for much longer after we define and work only with the unrolled definition of a context comprehension. But first, we need the notion of a representable (contravariant) functor into **Fam**.

Definition 2.5 (Representability and universal element)**.** Let $F : \mathcal{C}^{\mathrm{op}} \to \mathbf{Fam}$ be a contravariant functor. It is said to be *representable* if it has a *universal element* $(A, a) \in (A : \mathcal{C}) \times F(A)$, meaning that for any other $(B, b) \in (B : \mathcal{C}) \times F(B)$, there is a unique morphism $f : B \to A$ such that $F(f)(a) = b$. (Of course, a and b are pairs of elements in the families $F(A)$ and $F(B)$, respectively, so that $F(f)$ is a pair of functions mapping the components of a to the components of b.)

In our definition of a CwF, we require that $S_{\Gamma,A}$ be representable. The context comprehension of Γ and A is defined as its universal element.

Definition 2.6 (Context comprehension)**.** The context comprehension of $\Gamma \in \mathcal{C}$ and $A \in \mathrm{Ty}(\Gamma)$ is the universal element of $S_{\Gamma,A}$, which consists of the tuple of a context $\Gamma \triangleright A \in \mathcal{C}$, a substitution $\mathrm{p}_{\Gamma,A} : \Gamma \triangleright A \to \Gamma$, and a term $\mathrm{q}_{\Gamma,A} \in \mathrm{Tm}(\Gamma \triangleright A, A[\mathrm{p}_{\Gamma,A}])$. Given any other tuple $(\Delta, \gamma, a) \in (\Delta : \mathcal{C}) \times (\gamma : \Delta \to \Gamma) \times \mathrm{Tm}(\Delta, A[\gamma])$, there is a unique substitution $\langle \gamma, a \rangle : \Delta \to \Gamma \triangleright A$ such that $S_{\Gamma,A}(\langle \gamma, a \rangle)(\mathrm{p}_{\Gamma,A}, \mathrm{q}_{\Gamma,A}) = (\gamma, a)$, *i.e.* $\mathrm{p}_{\Gamma,A} \circ \langle \gamma, a \rangle = \gamma$ and $\mathrm{q}_{\Gamma,A}[\langle \gamma, a \rangle] = a$.

Given a substitution $\gamma : \Delta \to \Gamma$, substitution the variables in Γ, the extended substitution $\langle \gamma, a \rangle$ substitutes on top of that an additional variable of type A in the extended context $\Gamma \triangleright A$, with $\mathrm{p}_{\Gamma,A}$ as the first projection yielding the prior projections and $\mathrm{q}_{\Gamma,A}$ as the second projection yielding the new substitution term a. The context and type subscripts of p and q may be omitted if they can be deduced from context.

2.3.1 Uniqueness properties

We also have the following properties:

Lemma 2.7. *Let $\Gamma \in \mathcal{C}$ and $A \in \mathrm{Ty}(\Gamma)$. Then $\langle \mathrm{p}_{\Gamma,A}, \mathrm{q}_{\Gamma,A} \rangle = \mathrm{id}_{\Gamma \triangleright A}$.*

Proof. We have $\mathrm{p}_{\Gamma,A} \circ \mathrm{id}_{\Gamma \triangleright A} = \mathrm{p}_{\Gamma,A}$ by right identity and $\mathrm{q}_{\Gamma,A}[\mathrm{id}_{\Gamma \triangleright A}] = \mathrm{q}_{\Gamma,A}$ by functoriality on identity, so the substitutions must be equal by uniqueness. □

Lemma 2.8. *Let $\Xi, \Delta, \Gamma \in \mathcal{C}$, $\gamma : \Delta \to \Gamma$, $\delta : \Xi \to \Delta$, $A \in \mathrm{Ty}(\Gamma)$, and $a \in \mathrm{Tm}(\Gamma, A)$. Then $\langle \gamma, a \rangle \circ \delta = \langle \gamma \circ \delta, a[\delta] \rangle$.*

Proof. We have $\mathrm{p}_{\Gamma,A} \circ (\langle \gamma, a \rangle \circ \delta) = (\mathrm{p}_{\Gamma,A} \circ \langle \gamma, a \rangle) \circ \delta = \gamma \circ \delta$ and $\mathrm{q}_{\Gamma,A}[\langle \gamma, a \rangle \circ \delta] = \mathrm{q}_{\Gamma,A}[\langle \gamma, a \rangle][\delta] = a[\delta]$, so the substitutions must be equal by uniqueness. □

In fact, Lemmas 2.7 and 2.8 are sufficient to derive uniqueness of the comprehension substitution.

Lemma 2.9. *Let $\Delta, \Gamma \in \mathcal{C}$, $\gamma : \Delta \to \Gamma$, $A \in \mathrm{Ty}(\Gamma)$, and $a \in \mathrm{Tm}(\Delta, A[\gamma])$, so that $\langle \gamma, a \rangle : \Delta \to \Gamma \triangleright A$. Suppose there were another substitution $\delta : \Delta \to \Gamma \triangleright A$ such that $p_{\Gamma,A} \circ \delta = \gamma$ and $q_{\Gamma,A}[\delta] = a$. Then it must be that $\delta = \langle \gamma, a \rangle$.*

Proof. Consider $\langle \gamma, a \rangle = \langle p_{\Gamma,A} \circ \delta, q_{\Gamma,A}[\delta] \rangle$. By Lemma 2.8, this is equal to $\langle p_{\Gamma,A}, q_{\Gamma,A} \rangle \circ \delta$. By Lemma 2.7, this is equal to $\mathrm{id}_{\Gamma \triangleright A} \circ \delta$. Finally, by left identity, this is equal to δ. □

A context comprehension can then be specified by the context extension operator $\_ \triangleright \_$, the substitution extension operator $\langle \_, \_ \rangle$, the projections p and q, and Lemmas 2.7 and 2.8 as properties rather than lemmas in place of uniqueness.

2.3.2 Projection and lifting, weakening and substitution

Alone, the term $\mathrm{q}_{\Gamma,A}$ represents the newest variable in $\Gamma \triangleright A$, or the 0th de Bruijn index. Notice that $\mathrm{p}_{\Gamma,A}$ is also a weakening substitution, since the substitution $\_[\mathrm{p}_{\Gamma,A}]$ turns types and terms in Γ into ones in $\Gamma \triangleright A$. To get the ith de Bruijn index of type A_i in some context

$\Gamma' = \Gamma \rhd A_i \rhd \dots A_0$, then, we use q_{Γ, A_i} followed by weakening the context by $A_{i-1}, \dots, A_0$, yielding

$$\mathrm{q}_{\Gamma, A_i}[\vec{\mathrm{p}}_{\Gamma', i}] \in \mathrm{Tm}(\Gamma \rhd A_i \rhd \dots \rhd A_0, A_i[\mathrm{p}_{\Gamma, A_i} \circ \vec{\mathrm{p}}_{\Gamma', i}]), \text{where}$$
$$\vec{\mathrm{p}}_{\Gamma', i} \triangleq \mathrm{p}_{\Gamma \rhd A_i, A_{i-1}} \circ \dots \circ \mathrm{p}_{\Gamma \rhd A_i \rhd \dots \rhd A_1, A_0}.$$

Definition 2.10 (Context projection)**.** Given a context $\Gamma \in \mathcal{C}$ and types $A_i \in \mathrm{Ty}(\Gamma), \dots, A_0 \in \mathrm{Ty}(\Gamma \rhd A_i \rhd \dots \rhd A_1)$, we define the ith projection of $\Gamma' = \Gamma \rhd A_i \rhd \dots \rhd A_0$ as

$$\pi_{\Gamma', i} \triangleq q_{\Gamma, A_i}[\vec{\mathrm{p}}_{\Gamma', i}] \in \mathrm{Tm}(\Gamma', A_i[\mathrm{p}_{\Gamma, A_i} \circ \vec{\mathrm{p}}_{\Gamma', i}]),$$

where $\vec{p}_{\Gamma', i}$ is defined as above.

Given a substitution $\gamma : \Delta \to \Gamma$, we can also *lift* the substitution to substitute over the extended context $\Gamma \rhd A$.

Definition 2.11 (Lifting)**.** Given substitution $\gamma : \Delta \to \Gamma$ and a type $A \in \mathrm{Ty}(\Gamma)$, we define lifting γ by A as

$$\gamma \uparrow A \triangleq \langle \gamma \circ \mathrm{p}_{\Delta, A[\gamma]}, \mathrm{q}_{\Delta, A[\gamma]} \rangle : \Delta \rhd A[\gamma] \to \Gamma \rhd A.$$

If we lift $\mathrm{p}_{\Gamma, A} \uparrow A_i$, we obtain a substitution in $\Gamma \rhd A \rhd A_i[\mathrm{p}_{\Gamma, A}] \to \Gamma \rhd A_i$, which weakens the context by A in between Γ and A_i. By chaining a sequence of lifts by $A_i, \dots, A_0$, we then obtain a substitution that weakens Γ' by A in the middle of the context. Informally, in type theory, this is weakening some context Γ, Δ to $\Gamma, x : A, \Delta$ for some fresh variable x.

Definition 2.12 (Context weakening)**.** Given a context $\Gamma \in \mathcal{C}$ and types $A, A_i \in \mathrm{Ty}(\Gamma), A_{i-1} \in \mathrm{Ty}(\Gamma \rhd A_i), \dots, A_0 \in \mathrm{Ty}(\Gamma \rhd A_i \rhd \dots \rhd A_1)$, we define weakening $\Gamma' = \Gamma \rhd A_i \rhd \dots \rhd A_0$ by A as the substitution

$$\mathrm{wk}_{\Gamma', i}(A) \triangleq \mathrm{p}_{\Gamma, A} \uparrow A_i \uparrow \dots \uparrow A_0$$
$$: \Gamma \rhd A \rhd A_i[\mathrm{p}_{\Gamma, A}] \rhd \dots \rhd A_0[\mathrm{p}_{\Gamma, A} \uparrow A_i \uparrow \dots \uparrow A_1] \to \Gamma'.$$

In the opposite direction, we have context substitution: informally in type theory, given some context $\Gamma, x : A, \Delta$ and a term $\Gamma \vdash a : A_i$, we yield the context $\Gamma, \Delta[x \mapsto a]$. Given a term $a \in \mathrm{Ty}(\Gamma, A_i)$, this is achieved by lifting the identity substitution extended by a. Lifting once by A_{i-1} yields the substitution $\langle \mathrm{id}_\Gamma, a \rangle \uparrow A_{i-1} : \Gamma \rhd A_{i-1}[\langle \mathrm{id}_\Gamma, a \rangle] \to \Gamma \rhd A_i \rhd A_{i-1}$.

Definition 2.13 (Context substitution)**.** Given a context $\Gamma \in \mathcal{C}$, types $A_i \in \mathrm{Ty}(\Gamma), \dots, A_0 \in Ty(\Gamma \rhd A_i \rhd \dots \rhd A_1)$, and a term $a \in \mathrm{Tm}(\Gamma, A_i)$, we define substitution in $\Gamma' = \Gamma \rhd A_i \rhd \dots \rhd A_0$ by a as the substitution

$$\mathrm{sb}_{\Gamma', i}(a) \triangleq \langle \mathrm{id}_\Gamma, a \rangle \uparrow A_{i-1} \uparrow \dots \uparrow A_0$$
$$: \Gamma \rhd A_{i-1}[\langle \mathrm{id}_\Gamma, a \rangle] \rhd \dots \rhd A_0[\langle \mathrm{id}_\Gamma, a \rangle \uparrow A_{i-1} \uparrow \dots \uparrow A_1] \to \Gamma'.$$

Exercise 2.14. Consider $\Gamma \rhd A_i \rhd \dots \rhd A_0 \in \mathcal{C}$, $A \in \mathrm{Ty}(\Gamma)$, and $a \in \mathrm{Tm}(\Gamma, A)$. What is the type of the substitution $\mathrm{sb}_{i+1}(a) \circ \mathrm{wk}_i(A)$? What does it do?

2.4 CwFs as a Generalized Algebraic Theory

Although working through the structure of what a CwF is requires a lot of machinery, ultimately only the unrolled definitions are needed for now; we don't really need to know that T is a contravariant functor or that comprehensions are universal elements. All we need to know are the types of the various objects and morphisms we have, along with the equations that they satisfy, making the signature of a CwF an algebraic theory, but since later definitions and equations can depend on earlier ones, these form a GAT. A CwF can then be described as consisting of:

- A collection $\mathcal{C}$ of contexts;
- For contexts $\Delta, \Gamma : \mathcal{C}$ a collection of substitutions $\Delta \to \Gamma$;
- For contexts $\Xi, \Delta, \Gamma : \mathcal{C}$ and substitutions $\gamma : \Delta \to \Gamma$, $\delta : \Xi \to \Delta$ a notion of their composition $\gamma \circ \delta : \Xi \to \Gamma$;
- For a context $\Gamma : \mathcal{C}$ an identity substitution $\mathrm{id}_\Gamma : \Gamma \to \Gamma$;
- An empty context $\bullet : \mathcal{C}$;
- For a context $\Gamma : \mathcal{C}$ an empty substitution $\langle\rangle : \Gamma \to \bullet$;
- For a context $\Gamma : \mathcal{C}$ a collection of types $\mathrm{Ty}(\Gamma)$ under Γ;
- For a context $\Gamma : \mathcal{C}$ and a type $A : \mathrm{Ty}(\Gamma)$ a collection of terms $\mathrm{Tm}(\Gamma, A)$ under Γ of type A;
- For contexts $\Delta, \Gamma : \mathcal{C}$ and a substitution $\gamma : \Delta \to \Gamma$ a substitution function $\_[\gamma] : \mathrm{Ty}(\Gamma) \to \mathrm{Ty}(\Delta)$ on types;
- For contexts $\Delta, \Gamma : \mathcal{C}$, a substitution $\gamma : \Delta \to \Gamma$, and a type $A : \mathrm{Ty}(\Gamma)$, a substitution function $\_[\gamma] : \mathrm{Tm}(\Gamma, A) \to \mathrm{Ty}(\Delta, A[\gamma])$ on terms;
- For a context Γ and a type $A : \mathrm{Ty}(\Gamma)$ a context extension operator $\Gamma \triangleright A : \mathcal{C}$;
- For contexts Δ, Γ, a substitution $\gamma : \Delta \to \Gamma$, a type $A : \mathrm{Ty}(\Gamma)$, and a term $a : \mathrm{Tm}(\Delta, A[\gamma])$, a substitution extension operator $\langle \gamma, a \rangle : \Delta \to \Gamma \triangleright A$;
- For a context Γ and a type $A : \mathrm{Ty}(\Gamma)$ a substitution $\mathrm{p}_{\Gamma,A} : \Gamma \triangleright A \to \Gamma$; and
- For a context Γ and a type $A : \mathrm{Ty}(\Gamma)$ a term $\mathrm{q}_{\Gamma,A} : \mathrm{Tm}(\Gamma \triangleright A, A[\mathrm{p}_{\Gamma,A}])$,

Subject to the following properties:

- Substitution composition is associative;
- Identity substitutions are left and right identities with respect to substitution;
- Empty substitutions are unique;
- The substitution function over an identity substitution is the identity function (on both types and terms);
- The substitution function over a composition of substitutions is the composition of the substitution functions over each substitution (on both types and terms);

- p composed with an extended substitution projects out the original substitution;
- The substitution function over an extended substitution applied to q projects out the extension term;
- The substitution p extended by the term q yields the identity substitution; and
- An extended substitution composed with a second substitution is equal to the original substitution composed with the second substitution extended by the substitution function over the second substitution applied to the original extension term.

The properties that must be satisfied, especially the uniqueness ones, become a little heavy when written out as prose, but presenting a CwF as a GAT means that we can define it as an instance of a giant record type containing these structures and the equations as equalities. I use Agda syntax below to as an example of such a record type, but this will not type check in Agda: later types require that earlier equalities be either definitional or that coercions along earlier equalities be explicit.

```
record CwF {ℓ} : Set (suc ℓ) where
  infixl 30 _▷_
  infixl 40 _[_]
  field
    -- Category with terminal element
    C    : Set ℓ
    _⇒_ : C → C → Set ℓ
    id   : ∀ {Γ} → Γ ⇒ Γ
    _∘_  : ∀ {Ξ Δ Γ} → (Δ ⇒ Γ) → (Ξ ⇒ Δ) → (Ξ ⇒ Γ)
    •    : C
    ⟨⟩   : ∀ {Γ} → Γ ⇒ •

    -- Category laws and terminality
    ass : ∀ {Θ Ξ Δ Γ} {γ : Δ ⇒ Γ} {δ : Ξ ⇒ Δ} {ε : Θ ⇒ Ξ} →
          (γ ∘ δ) ∘ ε ≡ γ ∘ (δ ∘ ε)
    idl : ∀ {Δ Γ} {γ : Δ ⇒ Γ} → id ∘ γ ≡ γ
    idr : ∀ {Δ Γ} {γ : Δ ⇒ Γ} → γ ∘ id ≡ γ
    ⟨⟩η : ∀ {Γ} {γ : Γ ⇒ •} → γ ≡ ⟨⟩

    -- Type functor and functor laws
    Ty   : C → Set ℓ
    _[_] : ∀ {Δ Γ} → Ty Γ → (Δ ⇒ Γ) → Ty Δ
    [id] : ∀ {Γ} {A : Ty Γ} → A [ id ] ≡ A
    [∘]  : ∀ {Ξ Δ Γ} {γ : Δ ⇒ Γ} {δ : Ξ ⇒ Δ} {A : Ty Γ} →
           A [ γ ∘ δ ] ≡ A [ γ ] [ δ ]

    -- Term functor and functor laws
    Tm   : ∀ Γ → Ty Γ → Set ℓ
    _⟨_⟩ : ∀ {Δ Γ} {A : Ty Γ} → Tm Γ A → (γ : Δ ⇒ Γ) → Tm Δ (A [ γ ])
    ⟨id⟩ : ∀ {Γ} {A : Ty Γ} {a : Tm Γ A} → a ⟨ id ⟩ ≡ a
    ⟨∘⟩  : ∀ {Ξ Δ Γ} {γ : Δ ⇒ Γ} {δ : Ξ ⇒ Δ} {A : Ty Γ} {a : Tm Γ A} →
           a ⟨ γ ∘ δ ⟩ ≡ a ⟨ γ ⟩ ⟨ δ ⟩
```

```
-- Context comprehension
_▷_   : ∀ Γ → Ty Γ → C
⟨_,_⟩ : ∀ {Δ Γ} {A : Ty Γ} → (γ : Δ ⇒ Γ) → Tm Δ (A [ γ ]) → (Δ ⇒ Γ ▷ A)
p     : ∀ {Γ} {A : Ty Γ} → (Γ ▷ A ⇒ Γ)
q     : ∀ {Γ} {A : Ty Γ} → Tm (Γ ▷ A) (A [ p ])

-- Context comprehension laws
pβ    : ∀ {Δ Γ} {A : Ty Γ} {γ : Δ ⇒ Γ} {a : Tm Δ (A [ γ ])} →
        p ∘ ⟨ γ , a ⟩ ≡ γ
qβ    : ∀ {Δ Γ} {A : Ty Γ} {γ : Δ ⇒ Γ} {a : Tm Δ (A [ γ ])} →
        q [ ⟨ γ , a ⟩ ] ≡ a
⟨pq⟩  : ∀ {Γ} {A : Ty Γ} → ⟨ p , q ⟩ ≡ id {Γ ▷ A}
⟨⟩∘   : ∀ {Ξ Δ Γ} {γ : Δ ⇒ Γ} {δ : Ξ ⇒ Δ} {A : Ty Γ} {a : Tm Δ (A [ γ ])} →
        ⟨ γ , a ⟩ ∘ δ ≡ ⟨ γ ∘ δ , a [ δ ] ⟩
```

Exercise 2.15. The types of `[id]`, `[∘]`, `qβ`, and `⟨⟩∘` don't type check in Agda. Which prior equalities in the record would need to be definitional for type checking to succeed? Add transports across these equalities to their types so that they *do* type check in Agda.

2.5 Supporting More Structures

Currently, the only CwF possible is trivial, since the only context that exists is the empty context, and there are no types or terms at all. For a minimally useful type theory, we may want at the very least an empty type and dependent function types, perhaps along with a single type universe. We therefore augment CwFs by adding more types, terms, and equations to support these structures. In general, there needs to be an element in Ty for each type former, elements in Tm for its introduction and elimination forms, equations reflecting their computation and uniqueness rules (if any), and equations describing the behaviour of applying substitutions to the types and terms.

Definition 2.16 ($\bot$-structure)**.** A CwF supports a $\bot$-structure if for any context $\Gamma \in \mathcal{C}$,

- There is a type Empty $\in \mathrm{Ty}(\Gamma)$; and
- Given $A \in \mathrm{Ty}(\Gamma)$ and $a \in \mathrm{Tm}(\Gamma, \mathrm{Empty})$ there is a term $\mathrm{absurd}(A, a) \in \mathrm{Tm}(\Gamma, A)$,

such that given a substitution $\gamma : \Delta \to \Gamma$,

- $\mathrm{Empty}[\gamma] = \mathrm{Empty} \in \mathrm{Ty}(\Delta)$; and
- $\mathrm{absurd}(A, a)[\gamma] = \mathrm{absurd}(A[\gamma], a[\gamma]) \in \mathrm{Tm}(\Delta, A[\gamma])$.

Definition 2.17 (Π-structures)**.** A CwF supports a Π-structure if for any context $\Gamma \in \mathcal{C}$,

- Given $A \in \mathrm{Ty}(\Gamma)$ and $B \in \mathrm{Ty}(\Gamma \triangleright A)$ there is a type $\mathrm{Pi}(A, B) \in \mathrm{Ty}(\Gamma)$;
- Given $b \in \mathrm{Tm}(\Gamma \triangleright A, B)$ there is a term $\mathrm{lam}(b) \in \mathrm{Tm}(\Gamma, \mathrm{Pi}(A, B))$; and
- Given $a \in \mathrm{Tm}(\Gamma, \mathrm{Pi}(A, B))$ there is a term $\mathrm{app}(a) \in \mathrm{Tm}(\Gamma \triangleright A, B)$,

such that for terms $a \in \mathrm{Tm}(\Gamma, \mathrm{Pi}(A, B))$ and $b \in \mathrm{Tm}(\Gamma \triangleright A, B)$,

- $\mathrm{app}(\mathrm{lam}(b)) = b$; and
- $\mathrm{lam}(\mathrm{app}(a)) = a$,

and given a substitution $\gamma : \Delta \to \Gamma$,

- $\mathrm{Pi}(A, B)[\gamma] = \mathrm{Pi}(A[\gamma], B[\gamma \uparrow A]) \in \mathrm{Ty}(\Delta)$;
- $\mathrm{lam}(b)[\gamma] = \mathrm{lam}(b[\gamma \uparrow A]) \in \mathrm{Tm}(\Delta, \Pi(A[\gamma], B[\gamma \uparrow A]))$; and
- $\mathrm{app}(a)[\gamma \uparrow A] = \mathrm{app}(a[\gamma]) \in \mathrm{Tm}(\Delta \rhd A[\gamma], B[\gamma \uparrow A])$.

Note that this formulation of application is the inverse of a function abstraction rather than the more familiar formulation of application of a function to an argument. As a typing rule, it would look like the following:

$$\frac{\Gamma \vdash b : \Pi A.B}{\Gamma, x : A \vdash b\, x : B}$$

To obtain the usual notion of application, we apply a substitution that substitutes x by the argument, leaving the remaining variables unchanged.

Definition 2.18 (Argument application)**.** Given a context $\Gamma \in \mathcal{C}$, types $A \in \mathrm{Ty}(\Gamma)$ and $B \in \mathrm{Ty}(\Gamma \rhd A)$, and terms $b \in \mathrm{Tm}(\Gamma, \mathrm{Pi}(A, B))$ and $a \in \mathrm{Tm}(\Gamma, A)$, we define

$$b \mathbin{@} a \triangleq \mathrm{app}(b)[\langle \mathrm{id}_\Gamma, a \rangle] \in \mathrm{Tm}(\Gamma, B[\langle \mathrm{id}_\Gamma, a \rangle])$$

Definition 2.19 (U-structure)**.** A CwF supports a U-structure if for any context $\Gamma \in \mathcal{C}$,

- There is a type $\mathrm{Univ} \in \mathrm{Ty}(\Gamma)$;
- Given $a \in \mathrm{Tm}(\Gamma, \mathrm{Univ})$ there is a type $\mathrm{el}(a) \in \mathrm{Ty}(\Gamma)$,
- There is a term $\mathrm{Empty}^{\mathsf{C}} \in \mathrm{Tm}(\Gamma, \mathrm{Univ})$; and
- Given $A \in \mathrm{Tm}(\Gamma, \mathrm{Univ})$ and $B \in \mathrm{Tm}(\Gamma \rhd \mathrm{el}(A), \mathrm{Univ})$ there is a term $\mathrm{Pi}^{\mathsf{C}}(A, B) \in \mathrm{Tm}(\Gamma, \mathrm{Univ})$,

such that for terms $A \in \mathrm{Tm}(\Gamma, \mathrm{Univ})$, $B \in \mathrm{Tm}(\Gamma \rhd \mathrm{el}(A), \mathrm{Univ})$,

- $\mathrm{el}(\mathrm{Empty}^{\mathsf{C}}) = \mathrm{Empty}$; and
- $\mathrm{el}(\mathrm{Pi}^{\mathsf{C}}(A, B)) = \mathrm{Pi}(\mathrm{el}(A), \mathrm{el}(B))$,

and given a substitution $\gamma : \Delta \to \Gamma$,

- $\mathrm{Univ}[\gamma] = \mathrm{Univ} \in \mathrm{Ty}(\Delta)$;
- $\mathrm{el}(a)[\gamma] = \mathrm{el}(a[\gamma]) \in \mathrm{Ty}(\Delta)$;
- $\mathrm{Empty}^{\mathsf{C}}[\gamma] = \mathrm{Empty}^{\mathsf{C}} \in \mathrm{Tm}(\Delta, \mathrm{Univ})$; and
- $\mathrm{Pi}^{\mathsf{C}}(A, B)[\gamma] = \mathrm{Pi}^{\mathsf{C}}(A[\gamma], B[\gamma]) \in \mathrm{Tm}(\Delta, \mathrm{Univ})$.

Normally, with a hierarchy of universes, rather than explicitly defining a code for each type individually, given an arbitrary type $A \in \mathrm{Ty}_\ell(\Gamma)$, there is a term $\mathrm{code}(A) \in \mathrm{Tm}(\Gamma, \mathrm{Univ}_\ell)$ such that $\mathrm{el}(\mathrm{code}(A)) = A$, and dually given a term $a \in \mathrm{Tm}(\Gamma, \mathrm{Univ}_\ell)$ we have $\mathrm{code}(\mathrm{el}(a)) = a$. Predicativity then arises from $\mathrm{Univ}_\ell \in \mathrm{Ty}_{\ell+1}(\Gamma)$. However, since there is only one universe and Univ itself is a type, we would then have $\mathrm{code}(\mathrm{Univ}) \in \mathrm{Tm}(\Gamma, \mathrm{Univ})$, resulting in inconsistencies due to type-in-type. Rather than Univ not being a type (which would defeat the purpose of a U-structure!), we simply don't define a code for it: there is no $\mathrm{Univ}^{\mathsf{C}}$.

Exercise 2.20. Define a $\bot$-structure for the empty type, a Bool-structure for booleans, and Σ-structures for dependent pairs.

Once again, these can be presented as extensions to the GAT for a CwF, written as fields of functions into types, terms, and equalities in the record type. For convenience, here I use Agda's syntax for extending record types, but again, some types won't type check because the required equalities aren't definitional, such as those for `abs()`, `lam()`, and `code()`.

```
open CwF {{...}}

_↑_ : ∀ {ℓ} {{cwf : CwF {ℓ}}} {Δ Γ : C} →
      (γ : Δ ⇒ Γ) → (A : Ty Γ) → (Δ ▷ A [ γ ] ⇒ Γ ▷ A)
γ ↑ A = ⟨ γ ∘ p , q ⟩

record Structures {ℓ} : Set (suc ℓ) where
  field
    {{cwf}} : CwF {ℓ}

    -- ⊥-structure
    ⊥     : ∀ {Γ} → Ty Γ
    abs   : ∀ {Γ} → (A : Ty Γ) → Tm Γ ⊥ → Tm Γ A
    ⊥[]   : ∀ {Δ Γ} {γ : Δ ⇒ Γ} → ⊥ [ γ ] ≡ ⊥
    abs[] : ∀ {Δ Γ} {γ : Δ ⇒ Γ} → {A : Ty Γ} → {a : Tm Γ ⊥} →
            (abs A a) [ γ ] ≡ abs (A [ γ ]) (a [ γ ])

    -- Π-structure
    Π     : ∀ {Γ} → (A : Ty Γ) → Ty (Γ ▷ A) → Ty Γ
    lam   : ∀ {Γ} {A : Ty Γ} {B : Ty (Γ ▷ A)} →
            Tm (Γ ▷ A) B → Tm Γ (Π A B)
    app   : ∀ {Γ} {A : Ty Γ} {B : Ty (Γ ▷ A)} →
            Tm Γ (Π A B) → Tm (Γ ▷ A) B
    Πβ    : ∀ {Γ} {A : Ty Γ} {B : Ty (Γ ▷ A)} {b : Tm (Γ ▷ A) B} →
            app (lam b) ≡ b
    Πη    : ∀ {Γ} {A : Ty Γ} {B : Ty (Γ ▷ A)} {a : Tm Γ (Π A B)} →
            lam (app a) ≡ a
    Π[]   : ∀ {Δ Γ} {A : Ty Γ} {B : Ty (Γ ▷ A)} {γ : Δ ⇒ Γ} →
            (Π A B) [ γ ] ≡ Π (A [ γ ]) (B [ γ ↑ A ])
    lam[] : ∀ {Δ Γ} {A : Ty Γ} {B : Ty (Γ ▷ A)} {γ : Δ ⇒ Γ} {b : Tm (Γ ▷ A) B} →
            (lam b) [ γ ] ≡ lam (b [ γ ↑ A ])
    app[] : ∀ {Δ Γ} {A : Ty Γ} {B : Ty (Γ ▷ A)} {γ : Δ ⇒ Γ} {a : Tm Γ (Π A B)} →
            (app a) [ γ ↑ A ] ≡ app (a [ γ ])
```

```
-- U-structure
U      : ∀ {Γ} → Ty Γ
el     : ∀ {Γ} → Tm Γ U → Ty Γ
⊥ᶜ     : ∀ {Γ} → Tm Γ U
Πᶜ     : ∀ {Γ} → (A : Tm Γ U) → Tm (Γ ▷ el A) U → Tm Γ U
⊥ᶜβ    : ∀ {Γ} → el ⊥ᶜ ≡ ⊥
Πᶜβ    : ∀ {Γ} {A : Tm Γ U} {B : Tm (Γ ▷ el A) U} →
         el (Πᶜ A B) ≡ Π (el A) (el B)
U[]    : ∀ {Δ Γ} {γ : Δ ⇒ Γ} → U [ γ ] ≡ U
el[]   : ∀ {Δ Γ} {γ : Δ ⇒ Γ} {a : Tm Γ U} → (el a) [ γ ] ≡ el (a ⟨ γ ⟩)
⊥ᶜ[]   : ∀ {Δ Γ} {γ : Δ ⇒ Γ} → ⊥ᶜ ⟨ γ ⟩ ≡ ⊥ᶜ
Πᶜ[]   : ∀ {Δ Γ} {γ : Δ ⇒ Γ} {A : Tm Γ U} {B : Tm (Γ ▷ el A) U} →
         (Πᶜ A B) ⟨ γ ⟩ ≡ Πᶜ (A ⟨ γ ⟩) (B ⟨ γ ↑ el A ⟩)
```

Exercise 2.21. ↑, `app⟨⟩`, and `el[]` also don't type check in Agda. Add coercions to `q` and `(a ⟨ γ ⟩)` across prior equalities to ensure that they do.

3 Type Theories from CwFs

In the next section, we'll see that CwFs themselves form a category under some definition of morphisms between them. Following that, we construct a concrete type theory, which corresponds to the *term model*, along with an interpretation function for the syntax, which corresponds to a CwF morphism from the term model into arbitrary CwFs. Finally, we show that the interpretation is *sound*, implying that the term model is initial in the category of CwFs.

The initiality of the term model acts like a reduction to *any* model (that is, any CwF that supports the same structures), similar to how in computability theory one could reduce a problem A to a problem B that is, say, recursively enumerable, to show that A is also recursively enumerable. If we wanted to show that a type theory is consistent, for example, this corresponds to there being a type $\bot \in \mathrm{Ty}(\bullet)$ such that $\mathrm{Tm}(\bullet, \bot)$ is empty, and it would suffice to provide another model in which this property is known to hold, since initiality should preserve this property.

Of course, the difficulty lies in demonstrating that there *is* another model with the desired properties, but the benefit is that once the structure of the CwFs is established, this step is separate from the syntax of the type theory and from showing its soundness. If another model, sometimes referred to as a *semantic model*, has been found, we are free to make minor alterations to the particular presentation of the type theory, and only the proof of soundness will need to be repaired.

3.1 The Category of CwFs

A CwF can be succinctly described by a tuple $\mathfrak{C} = (\mathcal{C}, T_\mathcal{C}, \bullet_\mathcal{C}, \langle\rangle_\mathcal{C}, \_ \triangleright_\mathcal{C} \_, \langle \_, \_ \rangle_\mathcal{C}, \mathrm{p}_\mathcal{C}, \mathrm{q}_\mathcal{C})$ satisfying the equations listed in Section 2.4. Each component is labelled by the category $\mathcal{C}$ so they can be distinguished, but I will omit most of them except on T and $\bullet$ since the category can be inferred from various arguments and subscripts, and to avoid clashing with the subscripts themselves. Suppose we have another CwF $\mathfrak{D} = (\mathcal{D}, T_\mathcal{D}, \bullet_\mathcal{D}, \langle\rangle_\mathcal{D}, \_ \triangleright_\mathcal{D} \_, \langle \_, \_ \rangle_\mathcal{D}, \mathrm{p}_\mathcal{D}, \mathrm{q}_\mathcal{D})$.

Definition 3.1 (CwF morphism)**.** A morphism between CwFs $\mathfrak{C}$ and $\mathfrak{D}$ consists of:

- A functor $F : \mathcal{C} \to \mathcal{D}$; and
- A natural transformation $\sigma : T_{\mathcal{C}} \to T_{\mathcal{D}} \circ F$

such that the following hold (holding off on introducing σ_Γ and $\sigma_{\Gamma,A}$ for the moment):

- $F(\bullet_{\mathcal{C}}) = \bullet_{\mathcal{D}}$;
- For every $\Gamma \in \mathcal{C}$, we have $F(\langle\rangle_\Gamma) = \langle\rangle_{F(\Gamma)}$;
- For every $\Gamma \in \mathcal{C}, A \in \mathrm{Ty}_{\mathcal{C}}(\Gamma)$, we have $F(\Gamma \triangleright A) = F(\Gamma) \triangleright \sigma_\Gamma(A)$;
- For every $\Delta, \Gamma \in \mathcal{C}, \gamma : \Delta \to \Gamma, A \in \mathrm{Ty}_{\mathcal{C}}(\Gamma), a \in \mathrm{Tm}_{\mathcal{C}}(\Delta, A[\gamma])$, we have $F(\langle\gamma, a\rangle) = \langle F(\gamma), \sigma_{\Gamma,A}(a)\rangle$;
- For every $\Gamma \in \mathcal{C}, A \in \mathrm{Ty}_{\mathcal{C}}(\Gamma)$, we have $F(\mathrm{p}_{\Gamma,A}) = \mathrm{p}_{F(\Gamma),\sigma_\Gamma(A)}$; and
- For every $\Gamma \in \mathcal{C}, A \in \mathrm{Ty}_{\mathcal{C}}(\Gamma)$, we have $F(\mathrm{q}_{\Gamma,A}) = \mathrm{q}_{F(\Gamma),\sigma_\Gamma(A)}$.

The natural transformation σ is a family of morphisms over $\mathcal{C}$; that is, given a context $\Gamma \in \mathcal{C}$, we have the morphism $\sigma(\Gamma) : T_{\mathcal{C}}(\Gamma) \to T_{\mathcal{D}}(F(\Gamma))$. Given a substitution $\gamma : \Delta \to \Gamma$, naturality of σ makes the following square commute:

$$\begin{array}{ccc} T_{\mathcal{C}}(\Gamma) & \xrightarrow{\sigma(\Gamma)} & T_{\mathcal{D}}(F(\Gamma)) \\ \downarrow{\scriptstyle T_{\mathcal{C}}(\gamma)} & & \downarrow{\scriptstyle T_{\mathcal{D}}(F(\gamma))} \\ T_{\mathcal{C}}(\Delta) & \xrightarrow[\sigma(\Delta)]{} & T_{\mathcal{D}}(F(\Delta)) \end{array}$$

In other words, $\sigma(\Delta) \circ T_{\mathcal{C}}(\gamma) = T_{\mathcal{D}}(F(\gamma)) \circ \sigma(\Gamma)$.

Recall, however, that T is actually a pair of functors Ty and Tm, so there are in fact two natural transformations where, given $\Gamma \in \mathcal{C}, A \in \mathrm{Ty}_{\mathcal{C}}(\Gamma)$,

- $\sigma_\Gamma : \mathrm{Ty}_{\mathcal{C}}(\Gamma) \to \mathrm{Ty}_{\mathcal{D}}(F(\Gamma))$, and
- $\sigma_{\Gamma,A} : \mathrm{Tm}_{\mathcal{C}}(\Gamma, A) \to \mathrm{Tm}_{\mathcal{D}}(F(\Gamma), \sigma_\Gamma(A))$.

Naturality then involves the type and term substitution functions in the following commuting squares:

$$\begin{array}{ccc} \mathrm{Ty}_{\mathcal{C}}(\Gamma) & \xrightarrow{\sigma_\Gamma} & \mathrm{Ty}_{\mathcal{D}}(F(\Gamma)) \\ \downarrow{\scriptstyle \_[\gamma]} & & \downarrow{\scriptstyle \_[F(\gamma)]} \\ \mathrm{Ty}_{\mathcal{C}}(\Delta) & \xrightarrow[\sigma_\Delta]{} & \mathrm{Ty}_{\mathcal{D}}(F(\Delta)) \end{array} \qquad \begin{array}{ccc} \mathrm{Tm}_{\mathcal{C}}(\Gamma, A) & \xrightarrow{\sigma_{\Gamma,A}} & \mathrm{Tm}_{\mathcal{D}}(F(\Gamma), \sigma_\Gamma(A)) \\ \downarrow{\scriptstyle \_[\gamma]} & & \downarrow{\scriptstyle \_[F(\gamma)]} \\ \mathrm{Tm}_{\mathcal{C}}(\Delta, A[\gamma]) & \xrightarrow[\sigma_{\Gamma,A[\gamma]}]{} & \begin{array}{c}\mathrm{Tm}_{\mathcal{D}}(F(\Delta), \sigma_\Gamma(A)[F(\gamma)]) \\ \mathrm{Tm}_{\mathcal{D}}(F(\Delta), \sigma_\Delta(A[\gamma]))\end{array} \end{array}$$

Given $\Gamma \in \mathcal{C}, A \in \mathrm{Ty}_{\mathcal{C}}(\Gamma), a \in \mathrm{Tm}_{\mathcal{C}}(\Gamma, A)$, the naturality equations are $\sigma_\Delta(A[\gamma]) = \sigma_\Gamma(A)[F(\gamma)]$ and $\sigma_{\Gamma,A[\gamma]}(a[\gamma]) = \sigma_{\Gamma,A}(a)[F(\gamma)]$; the first equation lets us write the bottom right corner of the second square in two different ways. In short, σ for types and terms preserves the notion of substitution across the functor, similar to how the equations listed in Definition 3.1 preserve all of the structure of a CwF *on the nose*, as they say.

Exercise 3.2. Show that the naturality equations for σ_Γ and $\sigma_{\Gamma,A}$ follow from the naturality equation for σ.

Definition 3.3 (Category **CwF**)**.** The category of CwFs consists of CwFs as objects and CwF morphisms as morphisms.

Exercise 3.4. What are the identity morphisms of **CwF**?

3.2 A Simple Type Theory

There are different ways to proceed with defining the type theory such that the corresponding term model is initial. The traditional way is to define equality judgements for contexts, substitutions, types, and terms, define an interpretation function $[\![\_]\!]$ from each of these into an arbitrary CwF, show that the interpretation is sound and preserves equality, and lift the interpretation function to act on equivalence classes of contexts, substitutions, types, and terms with respect to their equality judgements, with the term model consisting of the equivalence classes.

$$\boxed{\vdash \Gamma = \Gamma} \qquad \boxed{\Delta \vdash \gamma = \gamma : \Gamma} \qquad \boxed{\Gamma \vdash A = A} \qquad \boxed{\Gamma \vdash a = a : A}$$

This yields a type theory that uses explicit substitutions and de Bruijn indices, with $\vdash \Gamma$, $\Delta \vdash \gamma : \Gamma$, $\Gamma \vdash A$, and $\Gamma \vdash a : A$ as sugar for the reflexive judgements. The closeness of the type theory to the signature of the CwF makes showing initiality easy, since it falls from the structure and the soundness of the interpretation. The interpretation of contexts and substitutions is the functor F, and the interpretation of types and terms is the natural transformation σ; the required equations for a CwF morphism essentially define the interpretation, and soundness ensures the types all line up, since the interpretation is defined only on syntax, which may not be well formed or well typed. Castellan et al. [2017], Bezem et al. [2021] detail this systematic procedure and provide examples of concrete syntactic models of various CwFs, not limited to type theories.

However, most will begin with the typing judgements and derivation rules, and most type theories aren't shaped like the above. Typically substitution is implicit (*i.e.* a metafunction rather than syntax), there is no notion of context equality, and equality judgements are separate from well-formedness and well-typedness ones. Many will also use variables and capture-avoiding substitution rather than de Bruijn indices.

$$\boxed{\vdash \Gamma} \qquad \boxed{\Gamma \vdash A} \qquad \boxed{\Gamma \vdash A = A} \qquad \boxed{\Gamma \vdash a : A} \qquad \boxed{\Gamma \vdash a = a : A}$$

There are two options to rigourously show that the second presentation of the syntax corresponds to an initial term model:

1. Show that the first presentation's term model is initial, as usual, then show that the second presentation is equivalent to the first.

2. Directly construct a term model and its interpretation for the second presentation and show initiality from scratch.

The latter appears to be the most common strategy. However, the term model is often never explicitly stated, usually only the interpretation for the syntax (and not their equivalence classes) or for the derivation trees is given, and the soundness theorem is often stated without detailed proof. Quickly surveying a few works that use CwF for their

semantics, Hofmann [1997] defines the interpretation over syntax and states the soundness theorem with additional lemmas and hints for its proof; Birkedal et al. [2020] defines both the interpretation over derivations and the term model while explicitly proving soundness of the interpretation for relevant cases; Altenkirch [1999] defines the interpretation over syntax and states the soundness theorem in an appendix; Pientka and Schöpp [2020] defines the interpretation over derivations and states the soundness theorem; Atkey [2018] defines a CwF with additional structures and states the soundness theorem for an interpretation over syntax that isn't fully explicitly defined; and Atkey et al. [2014] defines a CwF with additional structures with no interpretation function. The consensus appears to be that at the very least, the interpretation function should be given and its corresponding soundness theorem stated, with novel proof cases explicit if they are tricky or notable.

In Figure 1, I define the derivation rules for the type theory corresponding to our CwF with structures in the second, more conventional presentation. Following the above past work, in the upcoming section I define the interpretation function and state the soundness theorem, but do not prove the theorem nor explicitly construct the term model or the interpretation function lifted to equivalence classes.

While the second presentation doesn't use explicit substitutions, I choose to retain de Bruijn indices. v_i refers to the ith de Bruijn index, and $b^{\uparrow}$ denotes raising all de Bruijn indices in b by 1. Substitution is a metafunction denoted as $b[a]$, which replaces v_0 in b by a and lowers all other de Bruijn indices in b by 1. Letting $b[\mathsf{v}_i \mapsto a]$ denote substitution of v_i by a and lowering all other de Bruijn indices larger than i by 1, given a context $\Gamma, \Delta = \Gamma, A_0, \ldots, A_n$, $\Delta[a]$ denotes $A_0[\mathsf{v}_0 \mapsto a], \ldots, A_i[\mathsf{v}_i \mapsto a], \ldots, A_n[\mathsf{v}_n \mapsto a]$, substituting references to the rightmost element of Γ by a. Finally, dependent function types $\Pi A.B$, functions $\lambda A.a$, and codes of function types $\widehat{\Pi} A.B$ introduce new bindings of type A, A, and $\mathsf{el}\, A$ in B, a, and B, respectively. Note also that congruence rules for the equality judgements have been omitted for space.

3.3 Soundness of the Interpretation

The interpretation consists of three mutually defined functions over the derivations of the contexts, types, and terms into some arbitrary CwF $(\mathcal{C}, T_{\mathcal{C}}, \bullet_{\mathcal{C}}, \langle\rangle, \_ \triangleright \_, \langle \_, \_ \rangle, \mathrm{p}, \mathrm{q})$. It's defined over the derivations rather than the syntax alone since this presentation of the type theory doesn't have enough annotations; in particular, in a syntax-directed interpretation of terms, while the interpretation of function application $b\,a$ additionally requires the relevant context Γ and type $B[a]$, since substitution is a metafunction, the types A and B of $\Pi A.B$ cannot be recovered from $B[a]$ alone to interpret a and b. By interpreting the derivations instead, the required contexts and types are provided by the shape of the derivation itself.

$$
\begin{aligned}
[\![\vdash \bullet]\!] &= \bullet_{\mathcal{C}} \\
[\![\vdash \Gamma, A]\!] &= [\![\vdash \Gamma]\!] \triangleright [\![\Gamma \vdash A]\!] \\
[\![\Gamma \vdash \bot]\!] &= \mathrm{Empty}\{[\![\vdash \Gamma]\!]\} \\
[\![\Gamma \vdash \Pi A.B]\!] &= \mathrm{Pi}([\![\Gamma \vdash A]\!], [\![\Gamma, A \vdash B]\!]) \\
[\![\Gamma \vdash \mathsf{Set}]\!] &= \mathrm{Univ}\{[\![\vdash \Gamma]\!]\} \\
[\![\Gamma \vdash \mathsf{el}\, A]\!] &= \mathrm{el}([\![\Gamma \vdash A : \mathsf{Set}]\!]) \\
[\![\Gamma \vdash a : B]\!] &= [\![\Gamma \vdash a : A]\!] \qquad \textit{when } \Gamma \vdash A = B \\
[\![\Gamma \vdash \mathsf{v}_i : A_i]\!] &= \pi_{[\![\vdash \Gamma]\!], i} \\
[\![\Gamma \vdash \mathsf{absurd}_A\, a : A]\!] &= \mathrm{absurd}([\![\Gamma \vdash A]\!], [\![\Gamma \vdash a : \bot]\!])
\end{aligned}
$$

$\boxed{\vdash \Gamma}$

$$\frac{}{\vdash \bullet} \qquad \frac{\vdash \Gamma \qquad \Gamma \vdash A}{\vdash \Gamma, A}$$

$\boxed{\Gamma \vdash A}$

$$\frac{\vdash \Gamma}{\Gamma \vdash \bot} \qquad \frac{\Gamma \vdash A \qquad \Gamma, A \vdash B}{\Gamma \vdash \Pi A.B} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \mathsf{Set}} \qquad \frac{\Gamma \vdash A : \mathsf{Set}}{\Gamma \vdash \mathsf{el}\, A}$$

$\boxed{\Gamma \vdash A = A}$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A = A} \qquad \frac{\Gamma \vdash A = B}{\Gamma \vdash A = B} \qquad \frac{\Gamma \vdash A = B \qquad \Gamma \vdash B = C}{\Gamma \vdash A = C}$$

$$\frac{\Gamma \vdash A = A' \qquad \Gamma, A \vdash B = B'}{\Gamma \vdash \Pi A.B = \Pi A'.B'} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \mathsf{el}\, \widehat{\bot} = \bot} \qquad \frac{\Gamma \vdash A : \mathsf{Set} \qquad \Gamma, \mathsf{el}\, A \vdash B : \mathsf{Set}}{\Gamma \vdash \mathsf{el}\, (\widehat{\Pi} A.B) = \Pi(\mathsf{el}\, A).(\mathsf{el}\, B)}$$

$\boxed{\Gamma \vdash a : A}$

$$\frac{\Gamma \vdash A = B \qquad \Gamma \vdash a : A}{\Gamma \vdash a : B} \qquad \frac{\vdash \Gamma, A_i, \ldots, A_0}{\Gamma, A_i, \ldots, A_0 \vdash \mathsf{v}_i : A_i} \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash a : \bot}{\Gamma \vdash \mathsf{absurd}_A\, a : A}$$

$$\frac{\Gamma \vdash A \qquad \Gamma, A \vdash b : B}{\Gamma \vdash \lambda A.b : \Pi A.B} \qquad \frac{\Gamma \vdash b : \Pi A.B \qquad \Gamma \vdash a : A}{\Gamma \vdash b\, a : B[a]} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \widehat{\bot} : \mathsf{Set}}$$

$$\frac{\Gamma \vdash A : \mathsf{Set} \qquad \Gamma, \mathsf{el}\, A \vdash B : \mathsf{Set}}{\Gamma \vdash \widehat{\Pi} A.B : \mathsf{Set}}$$

$\boxed{\Gamma \vdash a = a : A}$

$$\frac{\Gamma \vdash A = B \qquad \Gamma \vdash a = b : A}{\Gamma \vdash a = b : B} \qquad \frac{\Gamma \vdash a : A}{\Gamma \vdash a = a : A} \qquad \frac{\Gamma \vdash a = b : A}{\Gamma \vdash b = a : A}$$

$$\frac{\Gamma \vdash a = b : A \qquad \Gamma \vdash b = c : A}{\Gamma \vdash a = c : A} \qquad \frac{\Gamma \vdash a : A \qquad \Gamma, A \vdash b : B}{\Gamma \vdash (\lambda A.b)\, a = b[a] : B[a]} \qquad \frac{\Gamma, A \vdash a = b^{\uparrow}\, \mathsf{v}_0 : B}{\Gamma \vdash \lambda A.a = b : \Pi A.B}$$

Figure 1: Judgement forms and derivation rules for the simple type theory.

$$
\begin{aligned}
[\![\Gamma \vdash \lambda A.b : \Pi A.B]\!] &= \mathrm{lam}([\![\Gamma, A \vdash b : B]\!]) \\
[\![\Gamma \vdash b\, a : B[a]]\!] &= \mathrm{app}([\![\Gamma \vdash b : \Pi A.B]\!])[\langle \mathrm{id}_{[\![\vdash \Gamma]\!]}, [\![\Gamma \vdash a : A]\!] \rangle] \\
[\![\Gamma \vdash \widehat{\bot} : \mathsf{Set}]\!]_{\Gamma;\mathsf{Set}} &= \mathrm{Empty}^{\mathcal{C}}\{[\![\vdash \Gamma]\!]\} \\
[\![\Gamma \vdash \widehat{\Pi} A.B : \mathsf{Set}]\!]_{\Gamma;\mathsf{Set}} &= \mathrm{Pi}^{\mathcal{C}}([\![\Gamma \vdash A : \mathsf{Set}]\!], [\![\Gamma, \mathsf{el}\, A \vdash B : \mathsf{Set}]\!])
\end{aligned}
$$

At various points in the interpretation, the derivations of admissible judgements are needed despite not being direct premises. These derivations come from applying the following lemma as needed, which is proven by mutual induction over the derivations. Some inversion lemmas may be needed as well to handle the conversion rule.

Lemma 3.5 (Regularity)**.**

1. *If* $\Gamma \vdash A$ *then* $\vdash \Gamma$.
2. *If* $\Gamma \vdash A = B$ *then* $\Gamma \vdash A$ *and* $\Gamma \vdash B$.
3. *If* $\Gamma \vdash a : A$ *then* $\Gamma \vdash A$.
4. *If* $\Gamma \vdash a = b : A$ *then* $\Gamma \vdash a : A$ *and* $\Gamma \vdash b : A$.

Finally, soundness states that the interpretations yield elements of the correct sets (*i.e.* are well typed, as it were), and that the equality judgements respect the actual equalities of the interpretations. Again, any missing derivations are generated by application of Lemma 3.5.

Theorem 3.6 (Soundness of the interpretation)**.**

1. *If* $\vdash \Gamma$ *then* $[\![\vdash \Gamma]\!] \in \mathcal{C}$.
2. *If* $\Gamma \vdash A$ *then* $[\![\Gamma \vdash A]\!] \in \mathrm{Ty}_{\mathcal{C}}([\![\vdash \Gamma]\!])$.
3. *If* $\Gamma \vdash A = B$ *then* $[\![\Gamma \vdash A]\!] = [\![\Gamma \vdash B]\!]$.
4. *If* $\Gamma \vdash a : A$ *then* $[\![\Gamma \vdash a : A]\!] \in \mathrm{Tm}_{\mathcal{C}}([\![\vdash \Gamma]\!], [\![\Gamma \vdash A]\!])$.
5. *If* $\Gamma \vdash a = b : A$ *then* $[\![\Gamma \vdash a : A]\!] = [\![\Gamma \vdash b : A]\!]$.

According to Hofmann [1997], to prove soundness, additional lemmas for preservation of weakening and substitution are required. Of course, weakening and substitution themselves are required of the type theory first.

Lemma 3.7 (Weakening)**.** *Let* $\Delta = A_{i-1}, \ldots, A_0$ *and suppose* $\Gamma \vdash A_i$.

1. *If* $\Gamma, \Delta \vdash B$ *then* $[\![\Gamma, A_i, \Delta \vdash B]\!] = [\![\Gamma, \Delta \vdash B]\!][\mathrm{wk}_{[\![\vdash \Gamma, A_i, \Delta]\!], i}]$.
2. *If* $\Gamma, \Delta \vdash b : B$ *then* $[\![\Gamma, A_i, \Delta \vdash b : B]\!] = [\![\Gamma, \Delta \vdash b : B]\!][\mathrm{wk}_{[\![\vdash \Gamma, \Delta]\!], i}(A_i)]$.

Lemma 3.8 (Substitution)**.** *Let* $\Delta = A_i, A_{i-1}, \ldots, A_0$ *and suppose* $\Gamma \vdash a : A_i$.

1. *If* $\Gamma, A_i, \Delta \vdash B$ *then* $[\![\Gamma, \Delta[a] \vdash B[a]]\!] = [\![\Gamma, A_i, \Delta \vdash B]\!][\mathrm{sb}_{[\![\Gamma, A_i, \Delta]\!], i}(a)]$.
2. *If* $\Gamma, A_i, \Delta \vdash b : B$ *then* $[\![\Gamma, \Delta[a] \vdash b[a] : B[a]]\!] = [\![\Gamma, A_i, \Delta \vdash b : B]\!][\mathrm{sb}_{[\![\Gamma, A_i, \Delta]\!], i}(a)]$.

4 Semantic Models

Our goal in using CwFs is to show consistency, which for our simple type theory is defined as the metatheoretical statement of uninhabitance of the empty type, or $\bullet \nvdash e : \bot$ for any closed term e. If there were such a term, then *all* types A are inhabited via $\mathsf{absurd}_A\, e$. By the interpretation, consistency is stated in the CwF as emptiness of the set $\mathrm{Ty}(\bullet, \mathrm{Empty})$.

To show consistency, we need to find a semantic model in which $\mathrm{Tm}(\bullet, \mathrm{Empty})$ is empty, since if there were some closed term of type $\bot$, initiality of the term model yields an element of $\mathrm{Tm}(\bullet, \mathrm{Empty})$ by the interpretation, which would be a contradiction in the semantic model.

4.1 Type-Theoretic Model using Induction–Recursion

We've seen that the signature of a CwF can be presented as a GAT and implemented as a record type in Agda — up to transporting some propositional equalities as needed. Can this record type be inhabited? And more importantly, can it be inhabited in a way such that the propositional equalities hold definitionally so that annoying transports aren't needed to even state them?

Such a semantic model is known as a *shallow embedding* (as opposed to a *deep embedding* where equalities aren't restricted to definitional equality of the host type theory), and Kaposi et al. [2019] show how to construct such a semantic model in Agda. Our simple type theory is significantly simpler, but the same principles apply. The key idea is that the set of contexts is modelled by a universe `Set`, and a substitution `Δ ⇒ Γ` is exactly the type of functions `Δ → Γ`. Following that, types in the CwF are types in Agda that depend on a context. Everything else more or less falls into place by following the types.

```
-- Set is a category with a terminal element ⊤
open import Agda.Builtin.Unit

C : Set₁
C = Set

_⇒_ : C → C → Set
Δ ⇒ Γ = Δ → Γ

id : ∀ {Γ} → Γ ⇒ Γ
id x = x

_∘_ : ∀ {Ξ Δ Γ} → (Δ ⇒ Γ) → (Ξ ⇒ Δ) → (Ξ ⇒ Γ)
(γ ∘ δ) x = γ (δ x)

• : C
• = ⊤

⟨⟩ : ∀ {Γ} → Γ ⇒ •
⟨⟩ _ = tt

-- Ty are types that depend on a context
```

```
Ty : C → Set₁
Ty Γ = Γ → Set

infixl 40 _[_]
_[_] : ∀ {Δ Γ} → Ty Γ → (Δ ⇒ Γ) → Ty Δ
(A [ γ ]) x = A (γ x)

-- Tm are terms of Ty

Tm : ∀ Γ → Ty Γ → Set
Tm Γ A = (x : Γ) → A x

infixl 40 _(_)
_(_) : ∀ {Δ Γ} {A : Ty Γ} → Tm Γ A → (γ : Δ ⇒ Γ) → Tm Δ (A [ γ ])
(a ( γ )) x = a (γ x)
```

Context extension by some type is defined as a dependent pair of a context and a term of a type in that context. Since the only contexts we have are the empty context and constructed by context extension, contexts are essentially lists of types, each one depending on all prior types, packaged up as an arbitrary-length dependent pair. Here, we see why p and q are called the projections of a context extension.

```
-- Contexts are lists of types

infixl 30 _▷_
record _▷_ (Γ : C) (A : Ty Γ) : C where
  constructor _::_
  field
    p : Γ
    q : A p
open _▷_

⟨_,_⟩ : ∀ {Δ Γ} {A : Ty Γ} → (γ : Δ ⇒ Γ) → Tm Δ (A [ γ ]) → (Δ ⇒ Γ ▷ A)
⟨ γ , a ⟩ x = γ x :: a x

_↑_ : ∀ {Δ Γ} → (γ : Δ ⇒ Γ) → (A : Ty Γ) → (Δ ▷ A [ γ ] ⇒ Γ ▷ A)
γ ↑ A = ⟨ γ ∘ p , q ⟩
```

The equations that these definitions must obey all hold definitionally, since they correspond to the rules of function application and composition in Agda. The propositional equalities, all proven by `refl`, can be found in Appendix A.

Finally, we define the types in the simple type theory as exactly the corresponding types in Agda, albeit with additional dependencies on contexts. In particular, ⊥ truly is the empty type, and Π truly is a dependent function type former. Notice that `lam` corresponds to currying from a function on a context, while `app` corresponds to currying into a function on a context.

```
-- ⊥-structure
open import Data.Empty renaming (⊥ to ⊥′)
```

```
⊥ : ∀ {Γ} → Ty Γ
⊥ _ = ⊥′

abs : ∀ {Γ} → (A : Ty Γ) → Tm Γ ⊥ → Tm Γ A
abs _ b x = ⊥-elim (b x)

-- Π-structure

Π : ∀ {Γ} → (A : Ty Γ) → Ty (Γ ▷ A) → Ty Γ
Π A B x = (a : A x) → B (x ∷ a)

lam : ∀ {Γ} {A : Ty Γ} {B : Ty (Γ ▷ A)} → Tm (Γ ▷ A) B → Tm Γ (Π A B)
lam b x a = b (x ∷ a)

app : ∀ {Γ} {A : Ty Γ} {B : Ty (Γ ▷ A)} → Tm Γ (Π A B) → Tm (Γ ▷ A) B
app b (x ∷ a) = b x a
```

Finally, Univ and el are implemented in the usual inductive–recursive manner, where Univ is an inductive type consisting of constructors for each code, and el is a recursive function on codes into types. We have to be a little careful about the appearance of contexts: if we implement Univ directly as `data U : Γ : U → Ty Γ`, then given `x, y : Γ`, we have `U x ≠ U y` since the context isn't irrelevant in the type, and `U[]` won't hold definitionally. Therefore, `U′` and `el′` are first defined without contexts, and then the definitions involving contexts are defined in terms of them. This recovers definitional equality of all of the equations we need to prove, which again are found in Appendix A.

```
-- U-structure

data U′ : Set
el′ : U′ → Set

data U′ where
  ⊥ᶜ′ : U′
  Πᶜ′ : (A : U′) → (el′ A → U′) → U′

el′ ⊥ᶜ′ = ⊥′
el′ (Πᶜ′ A B) = (a : el′ A) → el′ (B a)

U : ∀ {Γ} → Ty Γ
U _ = U′

el : ∀ {Γ} → Tm Γ U → Ty Γ
el t x = el′ (t x)

⊥ᶜ : ∀ {Γ} → Tm Γ U
⊥ᶜ _ = ⊥ᶜ′
```

```
Πᶜ : ∀ {Γ} → (A : Tm Γ U) → Tm (Γ ▷ el A) U → Tm Γ U
Πᶜ A B x = Πᶜ′ (A x) (λ a → B (x :: a))
```

Here is also where the inconsistency of type-in-type would manifest. Suppose we added a code for Univ as a constructor `Uᶜ′ : U′`. To complete the definition of `el′`, we need to implement the case for `Uᶜ′`; it should decode to `U` itself, so we have `el′ Uᶜ = U′`. But this causes Agda to complain about the lack of strict positivity of `U`.

This is due to the presence of `Πᶜ`: its second explicit argument is `el′ A → U′`, but since `el′` could potentially yield `U′`, the second argument's type could have `U′` to the left of an arrow, making `U′` non–strictly-positive.

Finally, consistency is the noninhabitance of $\mathrm{Tm}(\bullet, \bot)$, which is definitionally `(x : ⊤) → ⊥ x`. Since this implies that the empty Agda type `⊥ tt` is inhabited, which in turn yields an inhabitant of *any* type, this is a contradiction (assuming Agda is consistent), so our simple type theory must be consistent as well.

```
absurd : ∀ {ℓ} {A : Set ℓ} → Tm • ⊥ → A
absurd b = ⊥-elim (b tt)
```

4.1.1 Variations on Universes

There are other ways to implement Univ and el, especially if the type theory has a universe hierarchy. For instance, following Kaposi et al. [2019], we can define `U i` at level `i` as exactly the Agda universe `Set i`, forgoing explicit type encodings. Then `el` would be a trivial identity function, and the codes can be constructed directly as Agda types. In this situation, Ty would also need to be indexed by a universe level.

```
Ty : ∀ i Γ → Set (lsuc i)
Ty i Γ = Γ → Set i

Tm : ∀ {i} Γ → (A : Ty i Γ) → Set (lsuc i)
Tm Γ A = (x : Γ) → A x

U : ∀ {Γ} i → Ty (lsuc i) Γ
U i _ = Set i

el : ∀ {Γ} {i} → Tm Γ (U i) → Ty i Γ
el t = t

⊥ᶜ : ∀ {i} {Γ} → Ty i Γ
⊥ᶜ _ = ⊥

Πᶜ : ∀ {i} {Γ} → (A : Tm Γ (U i)) → Tm (Γ ▹ el A) (U i) → Tm Γ (U i)
Πᶜ A B x = (a : A x) → B (x :: a)
```

This would be difficult to adapt to a type theory with a single universe, since we would need `U` to live in `Ty (lsuc lzero)` while all other types live in `Ty lzero`, preventing quantification over `Set` that is currently possible in our type theory. In a sense, this is the *shallowest* embedding within Agda, since even encodings of types and universes in the type theory must correspond exactly to Agda's.

On the other hand, the implementation we have could also be considered *too* shallow, since if we generalize to a universe hierarchy, `el` still has to return an Agda type of the desired universe level, so the codes we're allowed to decode must also fit within Agda's hierarchy of universes. Instead, following the technique of Kovács [2022], `Ty` corresponds to encodings of types in Agda, while `Tm` corresponds to terms of their decodings. As usual, `U` and `el` form an inductive–recursive definition. This tangles up the implementations of Ty, Tm, Univ, and el, but it still fits within the CwF signature supporting U-structures.

```
Ty : ∀ i Γ → Set
Tm : ∀ {i} Γ → Ty i Γ → Set
data U′ : ∀ i → Set
el′ : ∀ {i} → U′ i → Set

Ty i Γ = Γ → U′ i
Tm Γ A = (x : Γ) → el′ (A x)

data U′ where
  ...
  Uᶜ : ∀ i → U′ (i + 1)

...
el′ (Uᶜ i) = U′ i

U : ∀ {Γ} i → Ty (i + 1) Γ
U i _ = Uᶜ i

el : ∀ {Γ} {i} → Tm Γ (U i) → Ty i Γ
el t x = t x
```

Universe levels are no longer restricted to naturals, or even to Agda's universe levels: Kovács [2022] show that they can be an arbitrary well-founded order. Even so, we can't apply this technique to our type theory with a single universe, since it fundamentally relies on there being a code for `U`.

Note that because types are modelled entirely by the codes, everything can live in `Set`; using induction–recursion to model types moves them further away from Agda's own types and the model can be thought of as being even less shallow. However, terms in the type theory are still Agda terms of the appropriate type. Once Tm starts being implemented as, say, an inductive type, we really start moving into deep embeddings, and showing consistency becomes less straightforward (unless there's a way to interpret the terms of the inductive type back into Agda terms, as is done by McBride [2010]).

4.2 Set-Theoretic Model

Moving from type theory to set theory, it's possible to construct a set-theoretic model exactly analogous to the type-theoretic model, assuming a set theory with inductive–recursive functions. We also need general cartesian products $\prod_{a \in A} B_a$ and disjoint unions $\coprod_{a \in A} B_a$; I write these instead as $(a \in A) \to B(a)$ and $(a \in A) \times B(a)$, respectively, and informally treat their elements like dependent functions and dependent pairs.

The first step is to find a "large" enough set to play the role of `Set`. The standard method is to postulate a *Grothendieck* universe $\mathcal{G}$, which satisfies the following properties:

- $\varnothing \in \mathcal{G}$;
- If $A \in \mathcal{G}$ and $B \in A$ then $B \in \mathcal{G}$;
- If $A \in \mathcal{G}$ then so is its powerset $\wp(A) \in \mathcal{G}$; and
- If $A \in \mathcal{G}$ and $B : A \to \mathcal{G}$ then $\bigcup_{a \in A} B(a) \in \mathcal{G}$.

Exercise 4.1. Show that the following closure properties hold:

- If $A \in \mathcal{G}$ and $B \subseteq A$ then $B \in \mathcal{G}$.
- If $A, B \in \mathcal{G}$ then $A \times B \in \mathcal{G}$ and $A \cup B \in \mathcal{G}$.
- If $A \in \mathcal{G}$ and $B : A \to \mathcal{G}$ then $(a \in A) \to B(a) \in \mathcal{G}$ and $(a \in A) \times B(a) \in \mathcal{G}$.

The set-theoretic model excluding U-structures can then be defined in direct analogy to the type-theoretic model. For concision, I exclude the types of definitions are arguments (that is, the sets to which they belong). Below, *efq* is the elimination principle for elements of the empty set: *ex falso quodlibet*.

$$
\begin{aligned}
\mathcal{C} &= \mathcal{G} \\
\Delta \to \Gamma &= \Delta \to \Gamma \quad \text{(the function space)} \\
\mathrm{id}_\Gamma(x) &= x \\
(\gamma \circ \delta)(x) &= \gamma(\delta(x)) \\
\bullet &= \{\varnothing\} \\
\langle\rangle(x) &= \varnothing \\
\mathrm{Ty}(\Gamma) &= \Gamma \to \mathcal{G} \\
\mathrm{Tm}(\Gamma, A) &= (x \in \Gamma) \to A(x) \\
A[\gamma](x) &= A(\gamma(x)) \\
a[\gamma](x) &= a(\gamma(x)) \\
\Gamma \triangleright A &= (x \in \Gamma) \times A(x) \\
\mathrm{p}_{\Gamma,A}(x, a) &= x \\
\mathrm{q}_{\Gamma,A}(x, a) &= a \\
\langle \gamma, a \rangle(x) &= (\gamma(x), a(x)) \\
\mathrm{Empty}(\Gamma) &= \varnothing \\
\mathrm{absurd}(A, b) &= \mathit{efq}(b) \\
\mathrm{Pi}(A, B)(x) &= (a \in A(x)) \to B(x, a) \\
\mathrm{lam}(b)(x)(a) &= b(x, a) \\
\mathrm{app}(b)(x, a) &= b(x)(a)
\end{aligned}
$$

Now we simultaneously define a set $U \in \mathcal{G}$ inductively and a function $el : U \to \mathcal{G}$ recursively:

- $U_0 = \{\bot^{\mathtt{C}}\}$;

- If $A \in U_i$ and $B : el(A) \to U_i$ then $\Pi^{\mathtt{C}}(A, B) \in U_{i+1}$;
- $el_0(\bot^{\mathtt{C}}) = \varnothing$;
- $el_{i+1}(\Pi^{\mathtt{C}}(A, B)) = (a \in el_i(A)) \to el_i(B(a))$; and
- $U = \bigcup_{i \geq 0} U_i$ and $el = \bigcup_{i \geq 0} el_i$

Importantly, by the closure properties of $\mathcal{G}$, el_i always returns a set in $\mathcal{G}$. Furthermore, this construction bars us from including a faithful code of U, since we would need $el_0(U^{\mathtt{C}}) = U$ while we've not yet finished defining U. To finish things off, the final definitions of the set-theoretic model are below.

$$\begin{aligned}
\mathrm{Univ}(x) &= U \\
\mathrm{el}(t)(x) &= el(t(x)) \\
\mathrm{Empty}^{\mathtt{C}}(x) &= \bot^{\mathtt{C}} \\
\mathrm{Pi}^{\mathtt{C}}(A, B)(x) &= \Pi^{\mathtt{C}}(A(x), a \mapsto B(x, a))
\end{aligned}$$

Exercise 4.2. Verify that the required equations hold under set equality.

Consistency arises from emptiness of the empty set, for if $\mathrm{Tm}(\bullet, \mathrm{Empty}) = \{\varnothing\} \to \varnothing$ were inhabited by some set A, then $A(\varnothing)$ would be an element of the empty set, which is a contradiction.

4.2.1 Other Models in Sets

Our simple type theory is incredibly simple. Adding well-known features such as a propositional equality type will immediately require a different set-theoretic model. One option is to use *setoids* instead of sets to interpret types: these are sets equipped with an equivalence relation, and equality at a type corresponds to the equivalence relation for that set. Hofmann [1995], Altenkirch [1999], for example, use setoid models for extensional Martin-Löf type theory (MLTT), and Pujet and Tabareau [2022] use a setoid model for a type theory with observational equality. Another option is to use *groupoids*, which are categories where all morphisms are isomorphisms, and equality between two terms at a type corresponds to morphisms between them in a groupoid. Hofmann and Streicher [1998] use a groupoid model to show independence of UIP from intensional MLTT.

4.3 Presheaf Model

An $\mathcal{S}$-valued presheaf over $\mathcal{K}$ is a contravariant functor $\mathcal{K}^{\mathrm{op}} \to \mathcal{S}$. For instance, in CwFs, T is a **Fam**-valued presheaf over the category of contexts. Generally, presheaves are valued over **Set** if not further specified.

Presheaves can also be used as yet another model for CwFs and, according to Hofmann [1997],

> [...] generalises the set-theoretic model in that types are interpreted as variable sets (presheaves) or families of such.

Rather than modelling contexts as sets, we model them as presheaves over some category $\mathcal{K}$, with substitution as natural transformations between presheaves, and the rest accordingly. Indeed, many parts of the presheaf model will simply resemble the set theoretic model indexed by an additional category $\mathcal{K}$.

4.3.1 Contexts

Definition 4.3. Let $\mathcal{K}$ be a category. Then $\widehat{\mathcal{K}}$ is the category of presheaves over $\mathcal{K}$, whose objects are presheaves and morphisms are natural transformations between them. That is, given two presheaves $F, G : \mathcal{K}^{\text{op}} \to \mathbf{Set}$, two objects $x, y \in \mathcal{K}$, a morphism $\gamma : x \to y$, and a natural transformation $\sigma : F \to G$, the following square commutes,

$$\begin{array}{ccc} F(x) & \xrightarrow{\sigma(x)} & G(x) \\ {\scriptstyle F(\gamma)}\downarrow & & \downarrow{\scriptstyle G(\gamma)} \\ F(y) & \xrightarrow[\sigma(y)]{} & G(y) \end{array}$$

giving the equation $\sigma(y) \circ F(\gamma) = G(\gamma) \circ \sigma(x)$. For convenience, I also write the natural transformation applied to an object as a subscript, *e.g.* σ_x.

Letting $\mathcal{K}$ be some category, we use $\widehat{\mathcal{K}}$ as the category of contexts. The terminal context $\bullet$ is a presheaf mapping all objects in $\mathcal{K}$ to the set $\{\varnothing\}$ and all morphisms of $\mathcal{K}$ to the identity morphism $\text{id}_\varnothing$. Given a presheaf $\Gamma \in \widehat{\mathcal{K}}$, the terminal map $\langle\rangle_\Gamma$ is then the natural transformation $\Gamma \to \bullet$ where, for any $x \in \mathcal{K}, y \in \Gamma(x)$, we have $\langle\rangle_\Gamma(x)(y) = \varnothing$.

4.3.2 Types and Terms

Definition 4.4. Let Γ be a presheaf over $\mathcal{K}$. Then we have the category of its elements $\{\Gamma(I)\}_{I \in \mathcal{K}}$, whose objects are pairs (I, γ) where $I \in \mathcal{K}$ and $\gamma \in \Gamma(I)$, and for morphisms $f : J \to I$ in $\mathcal{K}$ there is a morphism $(J, \Gamma(f)(\gamma)) \to (I, \gamma)$, noting that $\Gamma(f)(\gamma) \in \Gamma(J)$. This category is also written as $\int_{\mathcal{K}} \Gamma$, and the morphism is also written as f_γ.

Definition 4.5. Let Δ, Γ be presheaves over $\mathcal{K}$, and let $\gamma : \Delta \to \Gamma$ be a natural transformation. We define the functor $\int_{\mathcal{K}} \gamma : \int_{\mathcal{K}} \Delta \to \int_{\mathcal{K}} \Gamma$ with its action on objects $(I, \delta) \in \int_{\mathcal{K}} \Delta$ as $(\int_{\mathcal{K}} \gamma)(I, \delta) = (I, \gamma_I(\delta))$. Given a morphism $f_\delta : (J, \Gamma(f)(\delta)) \to (I, \delta)$, we require that the functor's action on it be a morphism $(J, \gamma_J(\Gamma(f)(\delta))) \to (I, \gamma_I(\delta))$. By naturality, $\gamma_J \circ \Gamma(f) = \Gamma(f) \circ \gamma_I$, so we can define this action as $(\int_{\mathcal{K}} \gamma)(f_\delta) = f_{\gamma_I(\delta)} : (J, \Gamma(f)(\gamma_I(\delta))) \to (I, \gamma_I(\delta))$.

Given a context $\Gamma \in \widehat{\mathcal{K}}$, we define $\text{Ty}(\Gamma)$ as the set of presheaves over $\int_{\mathcal{K}} \Gamma$, so that a type $A \in \text{Ty}(\Gamma)$ is a presheaf $A : (\int_{\mathcal{K}} \Gamma)^{\text{op}} \to \mathbf{Set}$. Given a substitution (*i.e.* a natural transformation of presheaves) $\gamma : \Delta \to \Gamma$, we need to define application of substitution $A[\gamma] : (\int_{\mathcal{K}} \Delta)^{\text{op}} \to \mathbf{Set}$. Since $\int_{\mathcal{K}} \gamma : \int_{\mathcal{K}} \Delta \to \int_{\mathcal{K}} \Gamma$, we can define the application by composition $A[\gamma] = A \circ \int_{\mathcal{K}} \gamma$.

Given a context $\Gamma \in \widehat{\mathcal{K}}$ and a type $A \in \text{Ty}(\Gamma)$, $\text{Tm}(\Gamma, A)$ is the function space $A(\_)$. That is, given $a \in \text{Tm}(\Gamma, A)$ and $(I, \delta) \in \int_{\mathcal{K}} \Gamma$, we have $a(I, \delta) \in A(I, \delta)$. Given a substitution $\gamma : \Delta \to \Gamma$, we need to define application of substitution $a[\gamma] : A[\gamma](\_)$. That is, we need $a[\gamma](I, \delta) \in A[\gamma](I, \delta)$, which by definition is in $A(I, \gamma_I(\delta))$. We can then define the application by $a[\gamma](I, \delta) = a(I, \gamma_I(\delta))$.

4.3.3 Context Comprehension

Let $\Gamma \in \mathcal{K}, A \in \text{Ty}(\Gamma)$. Context extension $\Gamma \triangleright A$ is then a presheaf on $\mathcal{K}$ whose action on $I \in \mathcal{K}$ is the set $(\gamma \in \Gamma(I)) \times A(I, \gamma)$. Its action on a morphism $f : J \to I$ is a function

$(\gamma \in \Gamma(I)) \times A(I,\gamma) \to (\gamma \in \Gamma(J)) \times A(J,\gamma)$ given by $(\Gamma \triangleright A)(f)(\gamma, a) = (\Gamma(f)(\gamma), A(f_\gamma)(a))$, noting that $A(f_\gamma) : A(I,\gamma) \to A(J,\Gamma(f)(\gamma))$. Letting $\gamma : \Delta \to \Gamma$, $A \in \mathrm{Ty}(\Gamma)$, and $a \in \mathrm{Tm}(\Gamma, A[\gamma])$, substitution extension $\langle \gamma, a \rangle$ is a natural transformation $\Delta \to \Gamma \triangleright A$ given by $\langle \gamma, a \rangle(I)(\delta) = (\gamma_I(\delta), a(I, \gamma_I(\delta)))$. Finally, we define $\mathrm{p}_{\Gamma,A}(I)(\gamma, a) = \gamma$ and $\mathrm{q}_{\Gamma,A}(I)(\gamma, a) = a$, as expected.

4.3.4 Structures

I leave adding $\bot$-structures, Π-structures, and U-structures as exercises for the reader. I don't really know how to extend the model to U-structures anyway. Some details have been outlined by Laouar [2017].

4.3.5 Summary

Overall, the presheaf model is essentially adding indexing by an arbitrary category to the set-theoretic model. In the below, I define each component of the CwF using more type-theoretic–like notation, but ultimately they are all set-theoretic or category-theoretic in nature. As such, I omit the required category, functor, and natural transformation laws, and continue to (ab)use the convention of function application notation for the functorial action on both objects and morphisms, although I note their existence explicitly (*e.g.* in $\mathcal{C}$ and in Ty). Additionally, I instead use $\_ \Rightarrow \_$ for substitutions to avoid confusing with ordinary function arrows. Finally, assume some category $\mathcal{K}$.

$$
\begin{aligned}
\mathcal{C} &= (\Gamma : \mathcal{K} \to \mathcal{G}) \times (\{x, y \in \mathcal{K}\} \to (f : x \to y) \to \Gamma(y) \to \Gamma(x)) \\
\Delta \Rightarrow \Gamma &= (x \in \mathcal{K}) \to \Delta(x) \to \Gamma(x) \\
\mathrm{id}_\Gamma(x)(y) &= y \\
(\gamma \circ \delta)(x)(y) &= \gamma(x)(\delta(x)(y)) \\
\bullet(x) &= \{\varnothing\} \\
\bullet(f)(\varnothing) &= \varnothing \\
\langle\rangle(x)(y) &= \varnothing \\
\mathrm{Ty}(\Gamma) &= (A : (x \in \mathcal{K}) \to \Gamma(x) \to \mathcal{G}) \\
&\quad \times (\{x, y \in \mathcal{K}\} \to (f : x \to y) \to (z \in \Gamma(y)) \to A(y)(z) \to A(x)(\Gamma(f)(z))) \\
A[\gamma](x)(y) &= A(x)(\gamma(x)(y)) \\
\mathrm{Tm}(\Gamma, A) &= (x \in \mathcal{K}) \to (y \in \Gamma(x)) \to A(x)(y) \\
a[\gamma](x)(y) &= a(x)(\gamma(x)(y)) \\
(\Gamma \triangleright A)(x) &= (y \in \Gamma(x)) \times A(x)(y) \\
(\Gamma \triangleright A)(f)(y, a) &= (\Gamma(f)(y), A(f)(y)(a)) \\
\mathrm{p}_{\Gamma,A}(x)(y, a) &= y \\
\mathrm{q}_{\Gamma,A}(x)(y, a) &= a \\
\langle \gamma, a \rangle(x)(y) &= (\gamma(x)(y), a(x)(y))
\end{aligned}
$$

5 Natural Models

Natural models are an alternative formulation of the categorical structure involved in CwFs. When fully unrolled, they yield equivalent definitions and equations to those of CwFs, but package them up in a much more succinct form in terms of concepts familiar to most category theorists. In short, a natural model is a representable natural transformation between two presheaves over a category with a terminal object. There's a lot of information hidden inside of these terms; we'll unfold definitions level by level. At the surface level, we have that a natural model consists of:

- A category $\mathcal{C}$ with a terminal object $\bullet$;
- Presheaves (*i.e.* contravariant functors into **Set**) $\mathrm{Ty} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Set}$ and $\mathrm{Tm} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Set}$;
- A natural transformation $\tau : \mathrm{Tm} \to \mathrm{Ty}$; and
- A representation of τ.

The category $\mathcal{C}$ and its terminal object represent as before contexts and the empty context, with its morphisms as subsitutions. Given some context $\Gamma \in \mathcal{C}$, $\mathrm{Ty}(\Gamma)$ represents the set of well-formed types under Γ and $\mathrm{Tm}(\Gamma)$ represents the set of well-typed terms under Γ. The functorial actions of Ty and Tm on a substitution γ are once again applications of substitution $\_[\gamma]$, respecting identity substitutions and composition of substitutions.

Given $\Gamma \in \mathcal{C}$, the natural transformation provides a function $\tau_\Gamma : \mathrm{Tm}(\Gamma) \to \mathrm{Ty}(\Gamma)$: given some term, it yields the type of that term. Given a substitution $\gamma : \Delta \to \Gamma$, naturality gives the following commuting square:

$$\begin{array}{ccc} \mathrm{Tm}(\Gamma) & \xrightarrow{\tau_\Gamma} & \mathrm{Ty}(\Gamma) \\ {\scriptstyle \_[\gamma]}\downarrow & & \downarrow{\scriptstyle \_[\gamma]} \\ \mathrm{Tm}(\Delta) & \xrightarrow[\tau_\Delta]{} & \mathrm{Ty}(\Delta) \end{array}$$

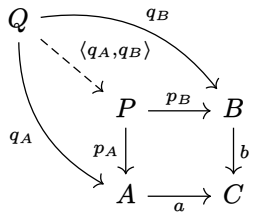
Given a term $a \in \mathrm{Tm}(\Gamma)$, we have that $\tau_\Delta(a[\gamma]) = \tau_\Gamma(a)[\gamma]$, or that the type of $a[\gamma]$ is the type of a applied to the substitution γ. As typing judgements, we would have that $\Gamma \vdash a : A$ implies $\Delta \vdash a[\gamma] : A[\gamma]$.

Before we can dissect the representation of τ, we need additional category-theoretic concepts in order to unfold all of the definitions.

5.1 Category Theory Speedrun any%

To get through all of the definitions, we only need to know the concepts of *pullbacks*, *Yoneda embeddings*, and one direction of the isomorphism in the *Yoneda lemma*. Rather than stating them in the most general form, I'll specialize some concepts to our application.

Definition 5.1 (Pullback)**.** Suppose we have three objects A, B, C in some category and morphisms $a : A \to C, b : B \to C$. A pullback consists of an object P and two morphisms $p_A : P \to A, p_B : P \to B$ such that given any other object Q and morphisms $q_A : Q \to A, q_B : Q \to B$, there is a unique morphism $\langle q_A, q_B \rangle : Q \to P$ such that all of the squares below commute.



A commuting square (drawn by P, A, B, C above) is a pullback if it has that unique morphism for every such Q.

Definition 5.2 (Yoneda embedding)**.** Given some category $\mathcal{C}$, the Yoneda embedding is a functor from $\mathcal{C}$ to presheaves over $\mathcal{C}$.

$$\yoneda : \mathcal{C} \to (\mathcal{C}^{\mathrm{op}} \to \mathbf{Set})$$

Given some object $\Gamma \in \mathcal{C}$, $\yoneda_\Gamma$ is a presheaf. When applied to another object $\Delta \in \mathcal{C}$, it yields the set of morphisms in $\mathcal{C}$

$$\yoneda_\Gamma(\Delta) = \Delta \to \Gamma,$$

and when applied to the morphism $\gamma : \Xi \to \Delta$, it is postcomposition of morphisms. Note the contravariance with respect to γ.

$$\begin{aligned} \yoneda_\Gamma(\gamma) : \yoneda_\Gamma(\Delta) &\to \yoneda_\Gamma(\Xi) \\ &: (\Delta \to \Gamma) \to (\Xi \to \Gamma) \\ \yoneda_\Gamma(\gamma)(\delta) &= \delta \circ \gamma \end{aligned}$$

Given some morphism $\gamma : \Xi \to \Delta$, $\yoneda_\gamma$ is a natural transformation between presheaves.

$$\yoneda_\gamma : \yoneda_\Xi \to \yoneda_\Delta$$

When applied to another object $\Gamma \in \mathcal{C}$, it is a precomposition of morphisms.

$$\begin{aligned} \yoneda_\gamma(\Gamma) : \yoneda_\Xi(\Gamma) &\to \yoneda_\Delta(\Gamma) \\ &: (\Gamma \to \Xi) \to (\Gamma \to \Delta) \\ \yoneda_\gamma(\Gamma)(\delta) &= \gamma \circ \delta \end{aligned}$$

Definition 5.3 (Yoneda lemma)**.** We will consider the lemma in terms of the context category $\mathcal{C}$. Given a presheaf T over $\mathcal{C}$ and a context $\Gamma \in \mathcal{C}$, there is an isomorphism between the set $T(\Gamma)$ and the set of natural transformations $\yoneda_\Gamma \to T$ between presheaves.

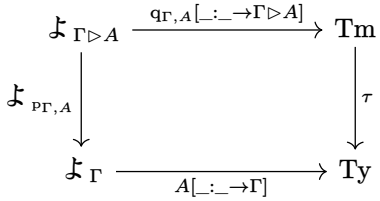
From left to right, consider an element $A \in T(\Gamma)$ with which we'll construct the natural transformation. Given another context $\Delta \in \mathcal{C}$, we then need to define a function from $\gamma : \Delta \to \Gamma$, a substitution, to $T(\Delta)$. This is exactly applying the substitution γ to A, since $A[\gamma] \in T(\Delta)$. I denote the natural transformation as $A[\_ : \_ \to \Gamma]$ and the function after applying Δ as $A[\_ : \Delta \to \Gamma]$, or just $A[\_]$.

From right to left, consider a natural transformation $\tau : \yoneda_\Gamma \to T$, with which we'll produce an element of $T(\Gamma)$. Applying Γ, we have a function $\tau_\Gamma : (\Gamma \to \Gamma) \to T(\Gamma)$. Then $\tau_\Gamma(\mathrm{id}_\Gamma) \in T(\Gamma)$.

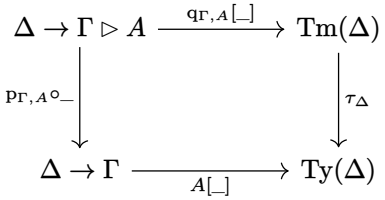
Exercise 5.4. Verify that the above construction is indeed an isomorphism.

5.2 Representability

Now back to our natural transformation $\tau : \mathrm{Tm} \to \mathrm{Ty}$. We say that τ is representable if for any context Γ and type $A \in \mathrm{Ty}(\Gamma)$, there is a context $\Gamma \triangleright A \in \mathcal{C}$, a substitution $\mathrm{p}_{\Gamma,A} : \Gamma \triangleright A \to \Gamma$, and a term $\mathrm{q}_{\Gamma,A} \in \mathrm{Tm}(\Gamma \triangleright A)$ such that the following square is a pullback:

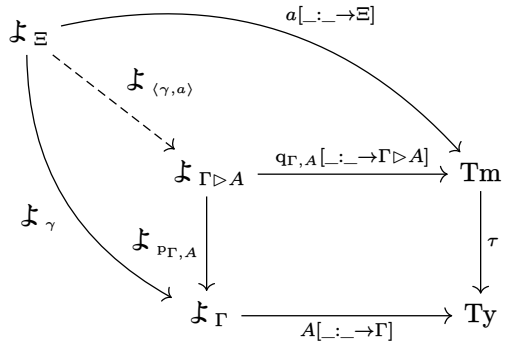


As before, $\Gamma \triangleright A$ is context extension, $\mathrm{p}_{\Gamma,A}$ is a weakening substitution by A, and $q_{\Gamma,A}$ is the newest bound variable in $\Gamma \triangleright A$. But this diagram is a little opaque, so let's take a context $\Delta \in \mathcal{C}$ and apply it across the whole diagram.

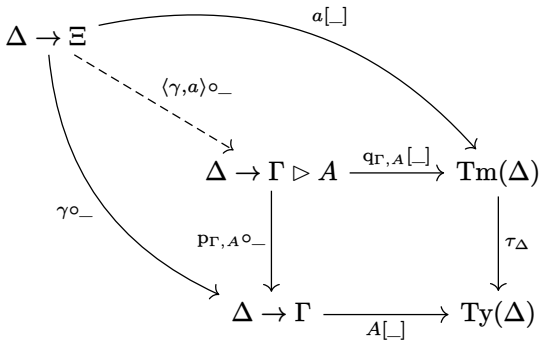


The fact that this square commutes says that given some substitution $\gamma : \Delta \to \Gamma \triangleright A$, $q_{\Gamma,A}[\gamma]$ has type $A[p_{\Gamma,A} \circ \gamma]$. If $\Delta = \Gamma \triangleright A$ and $\gamma = \mathrm{id}_{\Gamma \triangleright A}$, then we recover the fact that $q_{\Gamma,A}$ has type $A[p_{\Gamma,A}]$.

Now let's look at what the pullback gives us. We can take any presheaf and natural transformations from it into $よ_{\Gamma \triangleright A}$ and $よ_{\Gamma}$, but what will be most useful to us is to consider, given a context $\Xi \in \mathcal{C}$, a substitution $\gamma : \Xi \to \Gamma$, and a term $a \in \mathrm{Tm}(\Xi)$, the presheaf $よ_{\Xi}$ and the natural transformations $よ_{\gamma}$ and $a[\_ : \_ \to \Xi]$. We'll call the unique morphism $よ_{\langle \gamma, a \rangle}$, where $\langle \gamma, a \rangle : \Xi \to \Gamma \triangleright A$; evidently this is meant to suggest substitution extension.



Once again, let's apply $\Delta \in \mathcal{C}$ across the whole diagram.



By the commuting triangles, given some substitution $\delta : \Delta \to \Xi$, we have $p_{\Gamma,A} \circ \langle \gamma, a \rangle \circ \delta = \gamma \circ \delta$ and $q_{\Gamma,A}[\langle \gamma, a \rangle \circ \delta] = a[\delta]$. If $\Xi = \Delta$ and $\delta = \mathrm{id}_\Delta$, then we recover the equations $p_{\Gamma,A} \circ \langle \gamma, a \rangle = \gamma$ and $q_{\Gamma,A}[\langle \gamma, a \rangle] = a$. Finally, by uniqueness of the substitution extension and these two equations, we recover Lemmas 2.7 and 2.8.

All of the definitions and equations that were part of a CwF have now been recovered by the definition of a natural model. We can continue in this manner and support a $\bot$-structure, Π-structures, and a U-structure in terms of pullbacks involving τ, but this is beyond the scope of these notes. See Awodey [2016], Newstead [2018], for instance, for details on their construction.

Bibliography

Thorsten Altenkirch. Extensional equality in intensional type theory. In *Proceedings of the 14th Symposium on Logic in Computer Science*, pages 412–420, 1999. doi:10.1109/LICS.1999.782636. ↪ pages 15 and 24

Robert Atkey. Syntax and Semantics of Quantitative Type Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 56–65, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355834. doi:10.1145/3209108.3209189. ↪ page 15

Robert Atkey, Neil Ghani, and Patricia Johann. A Relationally Parametric Model of Dependent Type Theory. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '14, pages 503–515, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450325448. doi:10.1145/2535838.2535852. ↪ page 15

Steve Awodey. Natural models of homotopy type theory. *Mathematical Structures in Computer Science*, 28(2):241–286, Nov 2016. ISSN 1469-8072. doi:10.1017/s0960129516000268. ↪ page 30

Marc Bezem, Thierry Coquand, Peter Dybjer, and Martín Escardó. On generalized algebraic theories and categories with families. *Mathematical Structures in Computer Science*, 31(9):1006–1023, 2021. doi:10.1017/S0960129521000268. ↪ pages 2 and 14

Lars Birkedal, Ranald Clouston, Bassel Mannaa, Rasmus Ejlers Møgelberg, Andrew M. Pitts, and Bas Spitters. Modal dependent type theory and dependent right adjoints. *Mathematical Structures in Computer Science*, 30(2):118138, 2020. doi:10.1017/S0960129519000197. ↪ page 15

John W. Cartmell. *Generalised algebraic theories and contextual categories*. PhD thesis, University of Oxford, 1978. ↪ page 2

Simon Castellan, Pierre Clairambault, and Peter Dybjer. Undecidability of Equality in the Free Locally Cartesian Closed Category (Extended version). *Logical Methods in Computer Science*, 13(4), November 2017. doi:10.23638/LMCS-13(4:22)2017. ↪ page 14

Peter Dybjer. Internal type theory. In Stefano Berardi and Mario Coppo, editors, *Types for Proofs and Programs*, pages 120–134, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. ISBN 978-3-540-70722-6. doi:10.1007/3-540-61780-9_66. ↪ page 2

Martin Hofmann. *Extensional concepts in intensional type theory*. PhD thesis, University of Edinburgh, Edinburgh, Scotland, July 1995. URL http://www.lfcs.inf.ed.ac.uk/reports/95/ECS-LFCS-95-327/. ↪ page 24

Martin Hofmann. *Syntax and Semantics of Dependent Types*, pages 79–130. Publications of the Newton Institute. Cambridge University Press, 1997. doi:10.1017/CBO9780511526619.004. ↪ pages 15, 17, and 24

Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In *Twenty Five Years of Constructive Type Theory*. Oxford University Press, 10 1998. ISBN 9780198501275. doi:10.1093/oso/9780198501275.003.0008. ↪ page 24

Ambrus Kaposi, András Kovács, and Nicolai Kraus. Shallow Embedding of Type Theory is Morally Correct". In Graham Hutton, editor, *Mathematics of Program Construction*, pages 329–365, Cham, 2019. Springer International Publishing. ISBN 978-3-030-33636-3. doi:10.1007/978-3-030-33636-3_12. ↪ pages 18 and 21

András Kovács. Generalized Universe Hierarchies and First-Class Universe Levels. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic (CSL 2022)*, volume 216 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:17, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-218-1. doi:10.4230/LIPIcs.CSL.2022.28. ↪ page 22

Alexis Laouar. A presheaf model of dependent type theory. 2017. URL https://perso.crans.org/alaouar/rapportm1.pdf. ↪ page 26

Conor McBride. Outrageous but Meaningful Coincidences: Dependent Type-Safe Syntax and Evaluation. In *Proceedings of the 6th ACM SIGPLAN Workshop on Generic Programming*, WGP '10, page 112, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450302517. doi:10.1145/1863495.1863497. ↪ page 22

Clive Newstead. *Algebraic Models of Dependent Type Theory*. PhD thesis, Carnegie Mellon University, August 2018. URL https://kilthub.cmu.edu/articles/thesis/Algebraic_Models_of_Dependent_Type_Theory/7195124/1. ↪ page 30

Brigitte Pientka and Ulrich Schöpp. Semantical Analysis of Contextual Types. In *Foundations of Software Science and Computation Structures: 23rd International Conference*, pages 502–521, Berlin, Heidelberg, 2020. Springer-Verlag. ISBN 978-3-030-45230-8. doi:10.1007/978-3-030-45231-5_26. ↪ page 15

Loïc Pujet and Nicolas Tabareau. Observational Equality: Now for Good. *Proceedings of the ACM on Programming Languages*, 6(POPL), jan 2022. doi:10.1145/3498693. ↪ page 24

Taichi Uemara. *Abstract and Concrete Type Theories*. PhD thesis, Institute for Logic, Language and Computation, Jul 2021. URL https://hdl.handle.net/11245.1/41ff0b60-64d4-4003-8182-c244a9afab3b. ↪ page 2

A Equations of the Agda Model

```
open import Relation.Binary.PropositionalEquality.Core

-- Category laws and terminality

ass : ∀ {θ Ξ Δ Γ} {γ : Δ ⇒ Γ} {δ : Ξ ⇒ Δ} {ε : θ ⇒ Ξ} →
      (γ ∘ δ) ∘ ε ≡ γ ∘ (δ ∘ ε)
ass = refl

idl : ∀ {Δ Γ} {γ : Δ ⇒ Γ} → id ∘ γ ≡ γ
idl = refl

idr : ∀ {Δ Γ} {γ : Δ ⇒ Γ} → γ ∘ id ≡ γ
idr = refl

⟨⟩η : ∀ {Γ} {γ : Γ ⇒ •} → γ ≡ ⟨⟩
⟨⟩η = refl

-- Ty and Tm functor laws

[id] : ∀ {Γ} {A : Ty Γ} → A [ id ] ≡ A
[id] = refl

[∘] : ∀ {Ξ Δ Γ} {γ : Δ ⇒ Γ} {δ : Ξ ⇒ Δ} {A : Ty Γ} →
      A [ γ ] [ δ ] ≡ A [ γ ∘ δ ]
[∘] = refl

(id) : ∀ {Γ} {A : Ty Γ} {a : Tm Γ A} → a ( id ) ≡ a
(id) = refl

(∘) : ∀ {Ξ Δ Γ} {γ : Δ ⇒ Γ} {δ : Ξ ⇒ Δ} {A : Ty Γ} {a : Tm Γ A} →
      a ( γ ∘ δ ) ≡ a ( γ ) ( δ )
(∘) = refl

-- Context comprehension laws

infix 40 _ɜ⟨_,_⟩
_ɜ⟨_,_⟩ : ∀ {Δ Γ} → (A : Ty Γ) → (γ : Δ ⇒ Γ) → Tm Δ (A [ γ ]) → (Δ ⇒ Γ ▷ A)
_ ɜ⟨ γ , a ⟩ = ⟨ γ , a ⟩

pβ : ∀ {Δ Γ} {A : Ty Γ} {γ : Δ ⇒ Γ} {a : Tm Δ (A [ γ ])} →
     p {Γ} {A} ∘ ⟨ γ , a ⟩ ≡ γ
pβ = refl

qβ : ∀ {Δ Γ} {A : Ty Γ} {γ : Δ ⇒ Γ} {a : Tm Δ (A [ γ ])} →
     q {Γ} {A} ( ⟨ γ , a ⟩ ) ≡ a
qβ = refl
```

```
⟨pq⟩ : ∀ {Γ} {A : Ty Γ} → ⟨ p , q ⟩ ≡ id {Γ ▷ A}
⟨pq⟩ = refl

⟨⟩∘ : ∀ {Ξ Δ Γ} {γ : Δ ⇒ Γ} {δ : Ξ ⇒ Δ} {A : Ty Γ} {a : Tm Δ (A [ γ ])} →
     A ∋⟨ γ , a ⟩ ∘ δ ≡ ⟨ γ ∘ δ , a [ δ ] ⟩
⟨⟩∘ = refl

-- ⊥-stucture substitution laws

⊥[] : ∀ {Δ Γ} {γ : Δ ⇒ Γ} → ⊥ [ γ ] ≡ ⊥
⊥[] = refl

abs[] : ∀ {Δ Γ} {γ : Δ ⇒ Γ} → {A : Ty Γ} → {a : Tm Γ ⊥} →
        (abs A a) [ γ ] ≡ abs (A [ γ ]) (a [ γ ])
abs[] = refl

-- Π-structure computation, uniqueness, and substitution laws

Πβ : ∀ {Γ} {A : Ty Γ} {B : Ty (Γ ▷ A)} {b : Tm (Γ ▷ A) B} →
     app (lam b) ≡ b
Πβ = refl

Πη : ∀ {Γ} {A : Ty Γ} {B : Ty (Γ ▷ A)} {a : Tm Γ (Π A B)} →
     lam (app a) ≡ a
Πη = refl

Π[] : ∀ {Δ Γ} {A : Ty Γ} {B : Ty (Γ ▷ A)} {γ : Δ ⇒ Γ} →
      (Π A B) [ γ ] ≡ Π (A [ γ ]) (B [ γ ↑ A ])
Π[] = refl

lam[] : ∀ {Δ Γ} {A : Ty Γ} {B : Ty (Γ ▷ A)} {γ : Δ ⇒ Γ} {b : Tm (Γ ▷ A) B} →
        (lam b) [ γ ] ≡ lam (b [ γ ↑ A ])
lam[] = refl

app[] : ∀ {Δ Γ} {A : Ty Γ} {B : Ty (Γ ▷ A)} {γ : Δ ⇒ Γ} {a : Tm Γ (Π A B)} →
        (app a) [ γ ↑ A ] ≡ app (a [ γ ])
app[] = refl

-- U-structure computation and substitution laws

⊥ᶜβ : ∀ {Γ} → el {Γ} ⊥ᶜ ≡ ⊥
⊥ᶜβ = refl

Πᶜβ : ∀ {Γ} {A : Tm Γ U} {B : Tm (Γ ▷ el A) U} →
      el (Πᶜ A B) ≡ Π (el A) (el B)
Πᶜβ = refl
```

```
U[] : ∀ {Δ Γ} {γ : Δ ⇒ Γ} → U [ γ ] ≡ U
U[] = refl

el[] : ∀ {Δ Γ} {γ : Δ ⇒ Γ} {a : Tm Γ U} → (el a) [ γ ] ≡ el (a ⟨ γ ⟩)
el[] = refl

⊥ᶜ() : ∀ {Δ Γ} {γ : Δ ⇒ Γ} → (⊥ᶜ ⟨ γ ⟩) ≡ ⊥ᶜ
⊥ᶜ() = refl

Πᶜ() : ∀ {Δ Γ} {γ : Δ ⇒ Γ} {A : Tm Γ U} {B : Tm (Γ ▷ el A) U} →
       (Πᶜ A B) ⟨ γ ⟩ ≡ Πᶜ (A ⟨ γ ⟩) (B ⟨ γ ↑ el A ⟩)
Πᶜ() = refl
```