# DENOTATIONAL SEMANTICS OF TYPE THEORY

DANIEL GRATZER

Abstract. These are the lecture notes associated to a short talk given to a local seminar at Aarhus University. Below is the abstract for this talk.

Many of us use proof assistants like Coq or Agda to formalize the logics and arguments that we study. In so doing, however, it is not uncommon to bump into uncomfortable restrictions imposed by the type theory these tools support. In these circumstances, it is usually quickest to extend the proof assistant through postulates, rewrite rules, or plugins to better host our informal arguments. Doing this, however, carries a substantial risk: how do we know that we have not introduced an inconsistency?

In this talk I review the basics of (denotational) semantics for type theory with the aim of answering this question. We will work through the definition of a model of type theory and discuss how to construct one. Throughout, our focus be on how semantics can provide us with confidence that what we write in a proof assistant corresponds with our intentions.

## Contents



## 1. Introduction

Suppose we are working within a proof assistant based on dependent type theory (Coq, Agda, Idris, etc.) and suppose further that we end up requiring a reasoning principle that is not by-default available within these systems (function extensionality, propositional irrelevance, excluded middle, etc.). When are we allowed to simply postulate this principle as an axiom?

This question is far from obvious. There are many different extensions we can add to Coq and they are not always compatible. So, how are we to decide if it is OK to add something. As a first step, let us consider what "OK" means in this context. The coarsest answer is surely the following:

*Principle* 1. It is OK to add an axiom to type theory if the result is still consistent i.e., we cannot prove $\bot$ after assuming the axiom.

*Date*: Monday 24th April, 2023.

So this leads us to a new question: how do we tell if a type theory is consistent? A typical answer for type theory might be to prove something stronger and deduce consistency as a corollary.

For instance, we might prove that our type theory satisfies canonicity and then note that there are no canonical elements of $\bot$. This approach, however, is ill-suited for our situation. Our goal was to consider type theories which have extra axioms thrown in which will certainly produce stuck terms.

We opt for a different approach and argue for consistency by constructing a model. At a high level, a model consists of the following:

(1) A set $\mathcal{C}$ which represents contexts.
(2) A family of sets $\{\mathsf{Ty}(c) \mid c \in \mathcal{C}\}$ representing the types in each context.
(3) A family of sets $\{\mathsf{Tm}(c,t) \mid c \in \mathcal{C} \wedge t \in \mathsf{Ty}(c)\}$ representing terms of a given type in a context.
(4) A function $[\![-]\!]$ assigning well-formed contexts, types, and terms to elements of the appropriate sets which respects definitional equality.

As a quick remark on this last requirement, we demand that if a pair of terms are definitionally equal $\Gamma \vdash M = N : A$ then their interpretations are equal:

$$[\![M]\!] = [\![N]\!] \in \mathsf{Tm}([\![\Gamma]\!], [\![A]\!])$$

Even with just this fuzzy definition in place, we can present a roadmap for our proof of consistency will go. We will construct a model and ensure that $\mathsf{Tm}([\![\mathbf{1}]\!], [\![\bot]\!]) = \emptyset$. We can then easily argue that there is no term $\mathbf{1} \vdash M \colon \bot$.

**Theorem 1.1.** *If we have a model of our type theory such that* $\mathsf{Tm}([\![\mathbf{1}]\!], [\![\bot]\!]) = \emptyset$*, our type theory is consistent.*

*Proof.* We must argue that there are no terms $M : \bot$ in the empty context. Suppose that we had such a term. We may then interpret this term into our model $[\![M]\!] \in \mathsf{Tm}([\![\mathbf{1}]\!], [\![\bot]\!])$. By assumption, $\mathsf{Tm}([\![\mathbf{1}]\!], [\![\bot]\!])$ is empty, so no such $M$ may exist. □

**Definition 1.2.** We will call any model of type theory satisfying the above condition *non-trivial*.

We may summarize this line of reasoning as follows:

*Principle* 2. It is OK to add an axiom if the result has non-trivial models.

## 2. What is a model? What is a type theory?

Unfortunately, we cannot get very far in this process without getting a little bit more explicit about various definitions. We really need to crystallize a definition of model. Even worse, since the point of this exercise is to define models for various extensions of type theory, we really must define what a type theory is as well. We begin with the latter question.

Let us swallow the bitter pill first: there is not presently a good definition which captures everything we call type theory. There are several definitions, each capturing a different class of type theories with trade-offs in their convenience and what metatheorems they provide off-the-shelf.

In essence, we are after a definition which is somehow a compromise between two things. First, the definition should faithfully represent the syntax (*adequacy*). The point of this is to assure ourselves that what we're typing into Coq actually means something after all; we had better have a way to connect "type theory" to that.

As a second point, however, we also want a definition which concise, tractable, and close to the semantics if possible.

These two goals are often directly opposed to each other. In Coq, for instance, a great deal of work goes into ensuring the user does not have to write certain arguments or type annotations. A model of type theory, however, should be based on something not subject to change with every minor revision of Coq. It also intuitively feels like type theory without type annotations here is roughly the same as type theory with annotations, so it would be ideal if our definition of type theory was somehow invariant over these small details of syntax.[1]

For these notes, we deal with this issue by... not dealing with it. The goal is to focus on the high-level goals of semantics, so we will simply specify what a model is and offer only a few citations to assure the reader that this notion can be "linked up" with syntax [Car78; Dyb96; KKA19; Uem19; Uem21].

The bare-bones model of dependent type theory without any connectives is a slightly souped-up version of the structure introduced in the introduction.

**Definition 2.1** (Category with Families)**.** A *category with families (CwF)* is a category $\mathcal{C}$ equipped with any of the following equivalent structures:

(1) A functor $\mathsf{Ty} : \mathcal{C}^{\mathsf{op}} \longrightarrow \mathbf{Set}$ and an assignment of sets $\mathsf{Tm} : (c : \mathcal{C}) \times \mathsf{Ty}(c) \longrightarrow \mathbf{Set}$ together with a $\mathcal{C}$-action satisfying appropriate functoriality conditions:

$$- \cdot f : \mathsf{Tm}(c, A) \longrightarrow \mathsf{Tm}(c', A \cdot f) \qquad M \cdot \mathsf{id} = M \qquad M \cdot f \cdot g = M \cdot (f \circ g)$$

(2) A functor $\mathsf{Ty} : \mathcal{C}^{\mathsf{op}} \longrightarrow \mathbf{Set}$ and a functor $\mathsf{Tm} : \int \mathcal{U}^{\mathsf{op}} \longrightarrow \mathbf{Set}$.
(3) A functor $\tau : \mathcal{C}^{\mathsf{op}} \longrightarrow \mathbf{Fam}$ into the category of *families of sets*.
(4) A natural transformation $\tau : \mathcal{U}^{\bullet} \longrightarrow \mathcal{U}$ in $\mathbf{PSh}(\mathcal{C})$.

*Remark* 3. Of these formulations, (1) and (4) are the most useful in practice.

The *category* portion of a CwF is meant to represent the context and simultaneous substitutions between them. If we take the first description of the structure, $\mathsf{Ty}(c)$ represents the set of valid types in context $c$ and $\mathsf{Tm}(c, A)$ represents the terms in context $c$ of type $A$. The functoriality of $\mathsf{Ty}$ and $\mathsf{Tm}$ encodes substitution.

Given a type $A$ context $c$ and a simultaneous substitution $f : c' \longrightarrow c$ we can apply $f$ to $A$ to obtain a new type $A \cdot f \in \mathsf{Ty}(c')$. The functor laws assure us that these substitution operations behave as expected.

*Remark* 4. Why do simultaneous substitutions play such an important role? After all, it is not as though users frequently write substitutions between contexts when using Coq! In fact, the emphasis on simultaneous substitutions (hereafter, substitutions) and contexts is an important maneuver in making category theory a suitable language for the semantics of type theory.

Dependent type theory is predominantly concerned with families. A type is, morally, somehow a family indexed by the context. The primary objects of category theory are, however, emphatically not families! Objects are standalone gadgets with no sort of "context" at play. It is only by emphasizing contexts and substitutions between them that we can even view some part of dependent type theory as a category in any recognizable way.

For semantics of simple type theory, this distinction is basically linguistic. After all, in the simply-typed lambda calculus every type is precisely a length one context

[1]Small does not, however, mean unimportant. The ability of a proof assistant to minimize user-provided type annotation is critical for usability.

but the same is only true of *closed* types when we switch to Martin-Löf type theory. To directly encode dependent types, we therefore need more than just a category.

The definition of a CwF is essentially a compromise between these two facts. We isolate contexts and substitutions to have something categorical to base our semantics on, but keep track of open types and terms through the $\mathsf{Ty}$ and $\mathsf{Tm}$ functors. Other definitions of model split the difference differently and emphasize contexts and substitutions even more heavily [Joy17]. This is far more categorically natural but makes it more work to connect a model with syntax.

*Remark* 5. CwFs are closely linked to presentations of the syntax of type theory with explicit substitutions. In this style, we have syntactic objects representing substitutions in addition to types and terms. Applying a substitution becomes a first-class operation in syntax and definitional equalities are then added to ripple a substitution through a term to the variables [Mar92]. When presented in this style, the definition of a CwF can be mechanically obtained. In fact, this is how CwFs were first introduced [Dyb96].

*Exercise* 1. Convince yourself that these structures are all in fact equivalent.

I've somewhat atypically chosen to break this definition into three parts. The definition of CwF above does not force the category of contexts $\mathcal{C}$ to have any particular structure. To form a model of type theory, we must ensure that $\mathcal{C}$ contains an empty context and is closed under context extension.

**Definition 2.2.** A CwF $\mathcal{C}$ has an empty context just when $\mathcal{C}$ has a terminal object.

*Exercise* 2. What does the universal property of the terminal object encode here?

Context extension is a little more involved. It is also the construction which ties the presheaves of types and terms firmly to the category of contexts:

**Definition 2.3** (Context extension)**.** A CwF on $\mathcal{C}$ is said to have *context extensions* if any of the following properties are satisfied:

(1) For any $c : \mathcal{C}$ and $A \in \mathsf{Ty}(c)$, there exists a $c'$ along with a morphism $\mathsf{p} : c' \longrightarrow c$ and an element $\mathsf{q} \in \mathsf{Tm}(c', A \cdot f)$. Moreover, given a map $f : d \longrightarrow c$ and an element $a \in \mathsf{Tm}(d, A \cdot g)$ there exists a unique factorization $f = \mathsf{p} \circ \langle f, a \rangle$ such that $\mathsf{q} \cdot \langle f, a \rangle = a$.

(2) For any $c : \mathcal{C}$ and $A \in \mathsf{Ty}(c)$, there exists a $c'$ along with an isomorphism:

$$\mathsf{Hom}(-, c') \cong (f : \mathsf{Hom}(-, d)) \times \mathsf{Tm}(-, A \cdot f)$$

(3) Each fiber of $\tau$ is representable; for each $c : \mathcal{C}$ there exists some $c'$ such that $\mathcal{U}^\bullet \times_{\mathcal{U}} \mathbf{y}(c) \cong \mathbf{y}(c')$.

*Remark* 6. When (3) holds, we call $\tau$ a *representable natural transformation* [Awo18].

It is quite difficult to see why these definitions should model $\Gamma.A$. The most direction connection is to compare (1) with the rules for substitutions around $\Gamma.A$:

$$\frac{}{\uparrow : \Gamma.A \longrightarrow \Gamma} \qquad \frac{}{\Gamma.A \vdash \mathbf{v}_0 : A[\uparrow]} \qquad \frac{\gamma : \Delta \longrightarrow \Gamma \qquad \Delta \vdash M : A[\gamma]}{\gamma.M : \Delta \longrightarrow \Gamma.A}$$

In our semantics, $\mathsf{p}$ corresponds to $\uparrow$, $\mathsf{q}$ is $\mathbf{v}_0$, and $\langle \gamma, M \rangle$ is $\gamma.M$. If one reads a full description of the equational theory of of substitutions in Martin-Löf type theory, versions of all the equalities specified in (1) are present [Hof97; Mar92].

Henceforth, we shall assume that all CwFs we consider have both an empty context and context extensions. We will also usually presenting a CwF with a pair of functors $\mathsf{Ty}$ and $\mathsf{Tm}$ and occasionally reference how various structures are manifested if we encoded a CwF through a representable natural transformation.

*Notation* 7. We will write $c.A$ for the context extension of $c$ by $A \in \mathsf{Ty}(c)$.

*Exercise* 3. Convince yourself that these properties are in fact properties. That is, show that $c'$ is determined up to unique isomorphism in each of these items.

*Exercise* 4. Convince yourself that these properties are all equivalent.

With these three elements, we've successfully defined a model of type theory with no connectives. That is, this is sufficient to interpret the basic constructions of type theory. It only remains to heap a bunch of auxiliary structures on top of a CwF to let us interpret each connective.

*Remark* 8. Thus far only the first part of the definition of a CwF legitimately specified *structure*; the empty contexts and context extensions were constrained up to unique isomorphism if they existed. This is a good state of affairs for category theory. It works best with relatively little structure but lots of properties.[2]

Unfortunately, this nice state of affairs goes out the window when we start adding in type connectives. Each connective will add new structure to the CwF and the data they add is not determined up to isomorphism. This is particularly dissatisfying as most type connectives *should* really be determined up to unique isomorphism and the fact that they are not leads to genuine proof obligations in practice.

## 3. Specifying type constructors

We will work through the addition of three representative connectives to CwFs: a unit type, an empty type, and dependent products. We refer to Awodey [Awo18] for an explanation for how these constructors can be specified when systematically using representable natural transformations.

### 3.1. **The unit type.**

**Definition 3.1.** A CwF $(\mathcal{C}, \mathsf{Ty}, \mathsf{Tm})$ interprets the *unit type* when it equipped with the following structure:

(1) For all contexts $c$, a type $\mathsf{unit}_c \in \mathsf{Ty}(c)$.
(2) For all contexts $c$, an isomorphism $\iota_c : \mathsf{Tm}(c, \mathsf{unit}_c) \cong \{\star\}$.

Moreover, given a substitution $f : c' \longrightarrow c$, we require $\mathsf{unit}_{c'} = \mathsf{unit}_c \cdot f$.

These structures are the simplest case of a general pattern: we add structure for the type constructors then operations on terms corresponding to introduction and elimination. In this case, we were able to specify the introduction, elimination, $\beta$, and $\eta$ rules in one move. One direction of the isomorphism we required above is introduction rule, one is the elimination rule. The $\beta$ and $\eta$ rules are encoded by requiring these operations be inverse to each other. We must also stipulate that all our operations are stable under substitutions (the final requirement).

[2]Consider the complexity of monoidal category theory compared to categories with finite products.

*Remark* 9. Again, if one consults a sufficiently detailed presentation of type theory with explicit substitutions, all of these elements will be present [Hof97]. This definition should be seen as *translating* the existing rules into the language of CwFs instead of inference rules.

*Exercise* 5. One could ask that $\iota_c^{-1}$ is likewise stable under substitution:

$$\iota_c^{-1}(\star) \cdot f = \iota_{c'}^{-1}(\star)$$

Show that this equation holds automatically.

*Exercise* 6. Show that to equip $(\mathcal{C}, \mathsf{Ty}, \mathsf{Tm})$ with a unit type it suffices to give $\mathsf{unit}_{\mathbf{1}}$ and $\mathsf{Tm}(c, \mathsf{unit} \cdot !) \cong \{\star\}$ for each $c : \mathcal{C}$.

*Exercise* 7 $(*)$. Show that $(\mathcal{C}, \tau)$ supports a unit type just when there is morphism $\mathsf{unit} : \mathbf{1} \longrightarrow \mathcal{U}$ such that $\mathcal{U}^\bullet \times_{\mathcal{U}} \mathbf{1} \cong \mathbf{1}$.

3.2. **Dependent products.** The case of unit types was particularly straightforward because we were able to bundle up all the operations and equations regarding terms into a single isomorphism. In fact, we are able to execute this maneuver for essentially all types with an $\eta$ law.

**Definition 3.2.** A CwF $(\mathcal{C}, \mathsf{Ty}, \mathsf{Tm})$ interprets dependents products when it is equipped with the following structure:

- An operation on types $\mathsf{prod} : (c : \mathcal{C})(A : \mathsf{Ty}(c)) \to \mathsf{Ty}(c.A) \to \mathsf{Ty}(c)$.
- For each $c : \mathcal{C}$ along with types $A \in \mathsf{Ty}(c)$ and $B \in \mathsf{Ty}(c.A)$, an isomorphism:

$$\iota_c : \mathsf{Tm}(c, \mathsf{prod}_c(A, B)) \cong \mathsf{Tm}(c.A, B)$$

We further require that both $\iota$ and $\mathsf{prod}$ are natural in $c$.

Once again the first piece of required structure is the formation rule; an operation which assigns the formation data of a dependent product type to an actual type. The naturality requirement ensures that the formation rule is stable under substitution. This would typically be represented by an equation stating e.g., $(A \to B)[\sigma] = A[\sigma] \to B[(\sigma \circ \uparrow).\mathbf{v}_0]$.

The isomorphism bundles up the lambda rule, the application rule, and the $\beta$ and $\eta$ equalities. The lambda rule is easier to recognize: it corresponds to $\iota_c^{-1}$. The application rule is presented somewhat atypically here. Rather than allowing application to an arbitrary element of $\mathsf{Tm}(c, A)$, the elimination operation encoded here turns an element of the dependent product type back into an open term. This operation is sometimes referred to as *fun split* or *unlam* [NPS90]. It is completely interderivable with the normal function application. To apply a function $M \in \mathsf{Tm}(c, \mathsf{prod}(A, B))$ to a normal argument $N \in \mathsf{Tm}(c, A)$, we use the isomorphism to obtain $\iota_c(M) \in \mathsf{Tm}(c.A, B)$. We then use the functoriality of $\mathsf{Tm}$ together with the substitution $\langle \mathsf{id}, N \rangle : c \longrightarrow c.A$ to obtain the following:

$$M(N) \triangleq \iota_c(M) \cdot \langle \mathsf{id}, N \rangle \in \mathsf{Tm}(c, B \cdot \langle \mathsf{id}, N \rangle)$$

*Exercise* 8. Show that the $\beta$ and $\eta$ rules hold for this definition of application using the fact that $\iota$ is an isomorphism.

*Exercise* 9. Show that the normal application rule with $\beta$ and $\eta$ laws is sufficient to define $\iota$ (reversing the process we described above).

3.3. **The empty type.** We conclude with one type which lacks an $\eta$ law: the empty type $\mathsf{empty}$. Since this type is perhaps less well-known than the unit type and dependent products, we recall the traditional inference rules for it here:

$$\frac{}{\Gamma \vdash \mathbf{0}\ \mathit{type}} \qquad \frac{\Gamma \vdash M : \mathbf{0}}{\Gamma \vdash \mathsf{absurd}(M) : A}$$

Notice that (1) unlike our other types, there are no introduction rules and (2) there are no $\beta$ or $\eta$ laws.[3]

*Remark* 10. It is certainly possible to state an $\eta$ law for $\mathbf{0}$. Unfortunately, adding this rule to a type theory will invariable preclude decidable type-checking. Interestingly, it is the case that most of the models one encounters "in the wild" do support the $\eta$ law for $\mathbf{0}$:

$$\frac{\Gamma \vdash M : \mathbf{0} \qquad \Gamma \vdash N : A}{\Gamma \vdash \mathsf{absurd}(M) = N : A}$$

**Definition 3.3.** A CwF $(\mathcal{C}, \mathsf{Ty}, \mathsf{Tm})$ interprets an empty type when it is equipped with the following structure:

- Given $c : \mathcal{C}$, a choice of $\mathsf{empty}_c \in \mathsf{Ty}(c)$
- Given $c : \mathcal{C}$, $A \in \mathsf{Ty}(c)$, $M \in \mathsf{Tm}(c, \mathsf{empty}_c)$, a term $\mathsf{absurd}_c(M) \in \mathsf{Tm}(c, A)$.

We also require that both $\mathsf{empty}$ and $\mathsf{absurd}$ are natural in $c$ so that, e.g.:

$$\mathsf{absurd}(c, M) \cdot f = \mathsf{absurd}(c', M \cdot f)$$

*Exercise* 10 $(*)$. It might be natural to suggest that a $\tau : \mathcal{U}^\bullet \longrightarrow \mathcal{U}$ interprets an empty type if there is a point $\mathsf{empty} : \mathbf{1} \longrightarrow \mathcal{U}$ such that the fiber over this point $\mathcal{U}^\bullet \times_{\mathcal{U}} \mathbf{1}$ is the initial object. Show that no such point can ever exist.

3.4. **The complete definition of a model.**

**Definition 3.4.** A $(\Pi, \mathbf{1}, \mathbf{0})$-CwF is a CwF which interprets the unit type, the empty type, and dependent products.

Recall from Section 1 that our coarse definition of a model required a way to interpret each term, type, and context into the sets of terms, types, and contexts. Having now laid out the various definitions of a model, the reader may notice that this requirement has not explicitly reappeared. Instead, we have required that the various operations available for building up terms, types, and contexts are somehow supported by the model. This still gives a way to interpret a way to interpret entire syntactic objects: we proceed by inductively to build up a corresponding object in the model by replaying each operation.

**Definition 3.5.** Given a CwF $(\mathcal{C}, \mathsf{Ty}, \mathsf{Tm})$, we will write $[\![\Gamma]\!]$, $[\![A]\!]$, or $[\![M]\!]$ for the inductive interpretation of a given context, type, or term into $(\mathcal{C}, \mathsf{Ty}, \mathsf{Tm})$.

Requiring the interpretation to be defined in this manner is the hallmark of *denotational semantics*. It is often referred to as a compositional interpretation and is a stronger requirement than merely requiring that each term can be interpreted somehow. For instance, we have required that we can construct an inhabitant of

[3] It is difficult to imagine stating a $\beta$ rule without an introduction rule.

$\mathsf{Tm}(c, A)$ for any $A \in \mathsf{Ty}(c)$ when we have a proof of $\mathsf{empty}$, but not every inhabitant of $\mathsf{Ty}(c)$ needs to correspond to a syntactically definable type.[4]

For another example, consider dependent products. Our rules state that elements of $\mathsf{Tm}(c, \mathsf{prod}(A, B))$ correspond to all terms in an extended context. This will frequently mean that $\mathsf{Tm}(c, \mathsf{prod}(A, B))$ has elements corresponding to non-computable functions but we still require that the expected rules apply to these 'exotic' elements. Conversely, we require that syntactically-definable functions can be applied to all arguments, not merely those which are definable.

There are two principle benefits to these extra constraints. First, it gives a convenient way to compute the interpretations of terms and types within our model. Moreover, it allows us extend an interpretation of type theory to support an axiom without having to start from scratch. To argue that a model supports an axiom of type $\Gamma \vdash A$, it is sufficient to show that there is some inhabitant of $\mathsf{Tm}(\llbracket \Gamma \rrbracket, \llbracket A \rrbracket)$. Without compositionality, we would have to argue that we could still interpret a function applied to this axiom and that we could interpret a pair consisting of this axiom and something else and so on. Having ensured that our model supported a compositional interpretation essentially frontloads all this work.

Beyond this, compositionality gives us immense power in showing the *undefinability* of various operations. We can argue that if a function were definable then we could interpret it into some model and apply it to some non-definable argument and thereby derive a contradiction. This style of argument, for instance, is the foundation of many independence or continuity results.

## 4. Our first model of type theory

We now turn to actually constructing a category with families. We will begin with the so-called syntactic model.

**Definition 4.1.** The syntactic category of contexts $\mathsf{Cx}$ has contexts for objects and (equivalence classes of) substitutions for morphisms.

It is necessary to take substitutions up to definitional equality in order to ensure that $\mathsf{Cx}$ forms a category.

*Notation* 11. Given a type $\Gamma \vdash A$, we write $[A]$ for the equivalence class of types definitionally equal to $A$. Likewise, we write $[M]$ for the equivalence class of terms definitionally equal to $M$.

**Definition 4.2.** The category of contexts $\mathsf{Cx}$ can be equipped with a CwF structure by defining the presheaves of types and terms as follows:

$$\mathsf{Ty}(\Gamma) = \{[A] \mid \Gamma \vdash A\,\mathsf{type}\} \qquad \mathsf{Ty}(\Gamma, [A]) = \{[M] \mid \Gamma \vdash M : A\}$$

Once again, we must consider terms and types up to definitional equality in order to ensure that the definitions are functorial.

*Exercise* 11. Show that $(\mathsf{Cx}, \mathsf{Ty}, \mathsf{Tm})$ is a $(\Pi, \mathbf{0}, \mathbf{1})$-CwF.

[4] In a typical model $\mathsf{Ty}(c)$ will be uncountable so that nearly every type in the model cannot be realized syntactically.

## 5. The standard model

While the syntactic CwF is perhaps the simplest example we can construct, it tells us precious little that we didn't already know. In particular, it does not help us in our goal to show the consistency of our type theory. Let us instead focus on a model that offers some new insights in this regard: the set model of type theory.

5.1. **A digression on size.** The set model of type theory quickly leads to some fundamentally unimportant size issues. Essentially: we have already committed to having the collection of types in a context form a set but we will obtain a proper class of types if we proceed naively. There are a number of ways to circumvent this[5] but we will opt for a solution which gives us an opportunity to discuss a concept of fundamental importance in the semantics of type theory.

**Definition 5.1.** A Grothendieck universe $\mathcal{V}$ is a transitive set closed under the formation of pairs, $\mathcal{V}$-indexed unions, and powersets. Unless otherwise specified, we will also assume $\mathcal{V}$ contains the set of natural numbers.

*Remark* 12. A Grothendieck universe is precisely an inaccessible cardinal.

Grothendieck universes give us a workable proxy for the non-existent "set of all sets". Essentially, the definition above ensures that we can do any of the normal operations of set theory and if the inputs to these operations are elements of $\mathcal{V}$, the output will be an element of $\mathcal{V}$ as well. In fact, this can be precisely stated: $\mathcal{V}$ serves as a model of set theory. This crystallization makes it clear that incompleteness precludes us from obtaining a Grothendieck universe without adding some extra strength to our metatheory. This should not be a source of undue concern: the existence of a hierarchy of Grothendieck universes is a reasonably weak and unobjectionable addition. Accordingly, for the remainder of this document fix a Grothendieck universe $\mathcal{V}$.

**Definition 5.2.** A set is $\mathcal{V}$-small if it is isomorphic to an element of $\mathcal{V}$.

5.2. **Types as families of sets.** In this model a context is interpreted as a set. Writing **Set** for the category of sets and functions between them, we will therefore take **Set** as our category of contexts. It now remains to endow **Set** with the structure of a CwF. Ideally, we would like to interpret a type in context $X$ as a family of sets indexed by $X$.

*Remark* 13. While working in the category of sets, we have many different definitions of families of sets. Only one readily generalizes to an arbitrary category with finite limits: a family of sets $(Y_x)_{x\in X}$ is exactly a set $Y$ together with a map $f : Y \longrightarrow X$. The set at index $x$ is realized precisely by the preimage of $x$ i.e. $Y_x = f^{-1}(x) = \{y \in Y \mid f(y) = x\}$ i.e. the pullback $\mathbf{1} \times_X Y$ or the *fiber over* $x$. This definition arises frequently in geometry under the name *fibration* where it is used to capture the notion of a smoothly-varying families of shapes.

We would ideally now define $\mathsf{Ty}(X) =$ "families over $X$". Unfortunately, this is where we encounter size issues: the collection of families over a set is too large to form a set. To salvage the situation, we restrict types to families of $\mathcal{V}$-small sets. This restriction is not too draconian; we have already noted that we can perform any reasonable set-theoretic operations on $\mathcal{V}$-small sets without leaving $\mathcal{V}$.

[5]For the sake of exposition, handwaving is a defensible choice

*Remark* 14. In fact, it is entirely possible to restrict the category of contexts to the full subcategory spanned by $\mathcal{V}$-small sets.

While we have already defined the notion of a $\mathcal{V}$-small set, we will take a moment to crystallize the definition of a *family of $\mathcal{V}$-small sets*:

**Definition 5.3.** An $X$-family of $\mathcal{V}$-small sets is a function $X \longrightarrow \mathcal{V}$.

*Exercise* 12. Show that a family $f : X \longrightarrow \mathcal{V}$ determines a set $Y$ and a map $\pi : Y \longrightarrow X$ such that $\pi$ induces $f$ by taking preimages. What is the result of this procedure when applied to the identity map $\mathcal{V} \longrightarrow \mathcal{V}$? Can this process be reversed: does every family $\pi : Y \longrightarrow X$ determine a map $X \longrightarrow \mathcal{V}$? If not, what conditions can be placed upon $\pi$ to ensure this is the case?

With this definition to hand, we can now state the presheaf of types in our model:

$$\mathsf{Ty}(X) = \mathsf{Hom}(X, \mathcal{V})$$

Even more tersely: $\mathsf{Ty} = \mathbf{y}(\mathcal{V})$.

The collection of terms is slightly more involved. Fix a set $X$ and a family $f : X \longrightarrow \mathcal{V}$. A term in this model will consists of a family of elements $(a_x)_{x \in X}$ such that $a_x \in f(x)$. While this definition is perfectly serviceable as-is, we can rephrase it in a way which will make properties later easier to prove.

**Definition 5.4.** Define $\mathcal{V}_\bullet$ to be the set $\{(A, x) \mid A \in \mathcal{V} \wedge x \in A\}$. Let $\pi : \mathcal{V}_\bullet \longrightarrow \mathcal{V}$ be the evident map induced by projection.

We now define terms as follows:

$$\mathsf{Tm}(X, f) = \left\{ \tilde{f} : X \longrightarrow \mathcal{V}_\bullet \;\middle|\; \pi \circ \tilde{f} = f \right\}$$

**Lemma 5.5.** $\mathsf{Ty}$ *is a presheaf and* $\mathsf{Tm}$ *is equipped with a $\mathcal{C}$-action satisfying the necessary axioms.*

*Proof.* We have already mentioned the first part of this in passing. Both actions are induced by precomposition. It is straightforward to show that they are functorial. □

We next turn to defining a context extension structure and the relevant type formers. We leave the empty context as an exercise.

*Exercise* 13. The CwF $(\mathbf{Set}, \mathsf{Ty}, \mathsf{Tm})$ has an empty context.

**Lemma 5.6.** *The CwF* $(\mathbf{Set}, \mathsf{Ty}, \mathsf{Tm})$ *supports context extension.*

*Proof.* We begin by fixing $X : \mathbf{Set}$ and $f : X \longrightarrow \mathcal{V}$. We now wish to define $X.f$. Intuitively, a context extend by $f$ should consist of (1) an element $x \in X$ and (2) an element of $f(x)$. This is essentially already enough for a definition:

$$X.f = \textstyle\sum_{x \in X} f(x)$$

The weakening map $\mathsf{p} : X.f \longrightarrow X$ is projection of the first component. The term $\mathsf{q} \in \mathsf{Tm}(X.f, f \circ \mathsf{p})$ is essentially projection of the second component:

$$\mathsf{q}(x \in X, a \in f(x)) = (f(x), a)$$

Finally, we must show that given a $Y$ together with a map $\gamma : Y \longrightarrow X$ and $a \in \mathsf{Tm}(Y, f \circ \gamma)$ there exists a unique map $\langle \gamma, a \rangle : Y \longrightarrow X.f$ such that $\mathsf{p} \circ \langle \gamma, a \rangle = \gamma$ and $\mathsf{q} \cdot \langle \gamma, a \rangle = a$. This follows from the definition of $X.f$ as a disjoint sum:

$$\langle \gamma, a \rangle (y \in Y) = (\gamma(y), \pi_2(a(y)))$$

Check that this satisfies the required equations and is unique with this property is routine. □

**Lemma 5.7.** *The CwF* $(\mathbf{Set}, \mathsf{Ty}, \mathsf{Tm})$ *interprets the unit type.*

*Proof.* As mentioned in an exercise, it suffices to define a map $\mathsf{unit} : \mathbf{1} \longrightarrow \mathcal{V}$ along with a family of isomorphisms $\mathsf{Tm}(X, \mathsf{unit} \circ !) \cong \{\star\}$. To this end, we define $\mathsf{unit}(\star) = \mathbf{1}$ and we are essentially done. □

**Lemma 5.8.** *The CwF* $(\mathbf{Set}, \mathsf{Ty}, \mathsf{Tm})$ *interprets the empty type.*

*Proof.* We first define $\mathsf{empty}_X$ for each $X : \mathbf{Set}$ as follows:

$$\mathsf{empty}_X(\_) = \emptyset$$

To define $\mathsf{absurd}_X$, let us fix $f : X \longrightarrow \mathcal{V}$ and $e \in \mathsf{Tm}(X, \mathsf{empty}_X)$. We must construct $\mathsf{absurd}_X(f, e) \in \mathsf{Tm}(X, f) \subseteq \mathsf{Hom}(X, \mathcal{V}_\bullet)$. It therefore suffices to construct an element of $f(x)$ for each $x \in X$. We may now observe that any $x \in X$, we have $\pi_2(e(x)) \in \emptyset$, an absurdity. □

**Lemma 5.9.** *The CwF* $(\mathbf{Set}, \mathsf{Ty}, \mathsf{Tm})$ *interprets the dependent products.*

*Proof.* This is more involved than the previous two examples as it suffers from an abundance of indices. We begin by defining $\mathsf{prod}$.

Fix $X$ along with $f : X \longrightarrow \mathcal{V}$ and $g : X.f \longrightarrow \mathcal{V}$. We now define $\mathsf{prod}_X(f, g)$ as follows:

$$\mathsf{prod}_X(f, g) = \prod_{x \in X} \prod_{a \in f(x)} g(\langle x, a \rangle)$$

We must argue that this definition is natural in $X$, but we leave this routine calculation to the reader.

It remains to construct the following natural isomorphism:

$$\iota_X : \mathsf{Tm}(X, \mathsf{prod}(f, g)) \cong \mathsf{Tm}(X.f, g)$$

This follows essentially by definition. The left-hand side is isomorphic to the following by definition:

$$\prod_{x \in X} \prod_{a \in f(x)} g(\langle x, a \rangle)$$

The right-hand side, meanwhile, is isomorphic to the following:

$$\prod_{\langle x, a \rangle \in \sum_{x \in X} f(x)} g(\langle x, a \rangle)$$

There is an evident natural isomorphism given by currying:

$$\iota(\phi) = \lambda \langle x, a \rangle.\ \phi\, x\, a$$

□

.

## 6. Consequences

We can derive quite a few interesting consequences from the existence of the **Set** model. The most important is the following:

**Theorem 6.1.** *Type theory with* $\Pi$, $\mathbf{0}$, *and* $\mathbf{1}$ *is consistent.*

Of course, it is routine to extend the set model with other connectives (sums, identity types, universes, etc.). Once we have done this, we can argue that various axioms are consistent additions to type theory by observing that they are validated by this model. The following axioms are validated in this model:

(1) Function extensionality

(2) Law of the excluded middle (untruncated and truncated forms)
(3) The axiom of choice
(4) ...

## 7. Validating function extensionality

In order to expand on this last point and show that function extensionality is validated by the standard model, we must first add enough connectives to our model to interpret the axiom. Recall that function extensionality in type theory is a constant of the following type:

$$\prod_{fg:(a:A)\to B(a)} \left( \prod_{a:A} \mathsf{Id}(f\,a, g\,a) \right) \to \mathsf{Id}(f, g) \tag{1}$$

Our discussion thus far as not touched on a crucial type within the above expression: identity types. As a case study for how denotational semantics work with an extension of type theory, let us examine the effects of adding intensional identity types to our type theory, the definition of a model, and extend the standard model to support them. Finally, we examine equality reflection within the standard model.

### 7.1. Intensional identity types in type theory.

We consider the *intensional* formulation of identity types characteristic of Martin-Löf type theory [Mar84]. The formation and introduction (reflexivity) rules are given below:

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash M, N \colon A}{\Gamma \vdash \mathsf{Id}(M, N)} \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash M \colon A}{\Gamma \vdash \mathsf{refl}(a) \colon \mathsf{Id}(M, M)}$$

The character of the intensional identity type is given by its (rather complex) elimination, often named simply denoted "J":[6]

$$\frac{\begin{array}{c}\Gamma \vdash A \qquad \Gamma \vdash M_0, M_1 \colon A \\ \Gamma.A.A[\mathsf{p}].\mathsf{Id}(\mathsf{q}[\mathsf{p}], \mathsf{q}) \vdash B \qquad \Gamma.A \vdash N \colon B[\mathsf{id}.\mathsf{q}.\mathsf{refl}(\mathsf{q})] \qquad \Gamma \vdash P \colon \mathsf{Id}(M_0, M_1)\end{array}}{\Gamma \vdash \mathsf{ind}(B, N, P) \colon B[\mathsf{id}.M_0.M_1.P]}$$

$$\mathsf{ind}(B, N, \mathsf{refl}(M)) = N[\mathsf{id}.M]$$

*Remark* 15. While $\mathsf{ind}$ is certainly a mouthful, type theory with this formulation of identity types admits a normalization result and therefore decidable type-checking. Moreover, this elimination principle admits an interesting homotopical perspective. See Univalent Foundations Program [Uni13] for a more gentle introduction to this type former and the aforementioned homotopical interpretation.

### 7.2. Specifying identity types in CwFs.

Following Section 3, we specify when a CwF supports identity types. As the connective does not support an $\eta$ principle, the formulation is closest to the empty type.

**Definition 7.1.** A CwF $(\mathcal{C}, \mathsf{Ty}, \mathsf{Tm})$ interprets intensional identity types when it is equipped with the following structure:

[6] Near as I can tell, this name mostly stems from the fact that I is followed by J in the Latin alphabet.

- Given $c : \mathcal{C}$, a type $A \in \mathsf{Ty}(c)$, and two terms $a_0, a_1 \in \mathsf{Tm}(c, A)$, a choice of $\mathsf{id}_c(A, a_0, a_1) \in \mathsf{Ty}(c)$.
- Given $c : \mathcal{C}$, $A \in \mathsf{Ty}(c)$, and $a \in \mathsf{Tm}(c, A)$, a term $\mathsf{refl}_c(A, a) \in \mathsf{Ty}(c, \mathsf{id}(A, a, a))$.
- Fix the following:
  (1) $c : \mathcal{C}$
  (2) $A \in \mathsf{Ty}(c)$
  (3) $B \in \mathsf{Ty}(c.A.A.\mathsf{Id}(\mathsf{q} \cdot \mathsf{p}, \mathsf{q}))$
  (4) $N \in \mathsf{Tm}(c.A, B \cdot \langle\langle \mathsf{id}, \mathsf{q}\rangle, \mathsf{refl}_{c.A}(A \cdot \mathsf{p}, \mathsf{q})\rangle)$
  (5) $M_0, M_1 : \mathsf{Tm}(c, A)$
  (6) $P : \mathsf{Tm}(c, \mathsf{id}(A, M_0, M_1))$

  We then require a term $\mathsf{J}(c, A, B, N, M_0, M_1, P) \in \mathsf{Tm}(c, B \cdot \langle\langle\langle \mathsf{id}, M_0\rangle, M_1\rangle, P\rangle)$.

We further require that $\mathsf{id}$, $\mathsf{refl}$, and $\mathsf{J}$ are all natural and that the following equation holds:

$$\mathsf{J}(c, A, B, N, M, M, \mathsf{refl}_c(A, M)) = N \cdot \langle \mathsf{id}, M\rangle$$

In order to actually witness this structure on a model in practice, the following lemma is frequently helpful:

**Lemma 7.2.** *To equip a CwF $(\mathcal{C}, \mathsf{Ty}, \mathsf{Tm})$ with intensional identity types, it suffices to provide the following data:*

- *Given $c : \mathcal{C}$, a type $A \in \mathsf{Ty}(c)$, and two terms $a_0, a_1 \in \mathsf{Tm}(c, A)$, a natural choice of $\mathsf{id}_c(A, a_0, a_1) \in \mathsf{Ty}(c)$.*
- *Given $c : \mathcal{C}$, $A \in \mathsf{Ty}(c)$, and terms $M, N \in \mathsf{Tm}(c, A)$, a natural isomorphism of the following shape:*

$$\iota : \mathsf{Tm}(c, \mathsf{Id}(A, M, N)) \cong \{\star \mid M = N\}$$

*Proof.* The first two components of Definition 7.1 are given by $\mathsf{id}$ and $\iota^{-1}$. The final component is defined as follows:

$$\mathsf{J}(c, A, B, N, M_0, M_1, P) = N \cdot \langle \mathsf{id}, M_0\rangle$$

Here we take advantage of the fact that $\iota(P) \in \{\star \mid M_0 = M_1\}$ so that, in particular, $M_0 = M_1$ and $P = \mathsf{refl}(M_0)$. The naturality and computation equations follow from either assumption or definition. □

*Remark* 16. This lemma can be rephrased syntactically to say that extensional identity types interpret intensional identity types.

7.3. **Identity types in the standard model.** We now discuss the extension of standard model to support identity types and—finally—consider the validity of function extensionality in the model. Rather than directly constructing Definition 7.1, we will factor our proof through Lemma 7.2.

**Lemma 7.3.** *The standard model introduced in Section 5 supports the hypotheses of Lemma 7.2.*

*Proof.* We have two pieces of data to supply. First, fix $f : \mathsf{Ty}(X)$ and $s_0, s_1 : \mathsf{Tm}(X, f)$ for some $X : \mathbf{Set}$. We define $\mathsf{Id}(f, s_0, s_1)$ as follows:

$$\mathsf{Id}(f, s_0, s_1) = \lambda x. \begin{cases} \{\star\} & \text{if } s_0(x) = s_1(x) \\ \emptyset & \text{otherwise} \end{cases}$$

By construction, this definition is natural in $X$.

We must provide a natural isomorphism $\mathsf{Tm}(X, \mathsf{Id}(f, s_0, s_1)) \cong \{\star \mid s_0 = s_1\}$. First, note that any such isomorphism is automatically natural (there is at most one element of the right-hand side). Next, notice that if $s_0 = s_1$, then $\mathsf{Id}(f, s_0, s_1) = \lambda\_.\{\star\}$ which has precisely one term: the function which picks out $\star$. By function extensionality in sets, this map is a bijection: if we have a term of $\mathsf{Id}(f, s_0, s_1)$ then $s_0$ and $s_1$ are equal on all elements and thus equal. Accordingly, $\lambda\_.\star$ gives an isomorphism $\mathsf{Tm}(X, \mathsf{Id}(f, s_0, s_1)) \to \{\star \mid s_0 = s_1\}$. □

**Theorem 7.4.** *The standard model validates function extensionality.*

*Proof.* We wish to show that the interpretation of Eq. (1) is inhabited. We begin by unfolding this interpretation in a context $X$ with two types $f : \mathsf{Ty}(X)$ and $g : \mathsf{Ty}(X.f)$ which we will use to interpret $A$ and $B$ respectively.

Unraveling the interpretation and utilizing the various natural isomorphisms we have constructed, we are left with the following:

$$\begin{aligned}&(s_0, s_1 : \mathsf{Tm}(X.f, g))\\ &\quad \to ((a : \mathsf{Tm}(X, f)) \to \{\star \mid \forall x.\ s_0 \langle x, a\rangle = s_1 \langle x, a\rangle\})\\ &\quad \to \{\star \mid \forall x' : X.f.\ s_0\, x' = s_1\, x'\}\end{aligned}$$

At this point, let us examine the second term:

$$(a : \mathsf{Tm}(X, f)) \to \{\star \mid \forall x.\ s_0 \langle x, a\rangle = s_1 \langle x, a\rangle\}$$

Using function extensionality of sets, we see that it is in fact isomorphic to the following:

$$\{\star \mid \forall a, x.\ s_0 \langle x, a\rangle = s_1 \langle x, a\rangle\}$$

Using the fact that $X.f = \sum_{x:X} f(x)$, we can now see that every element of $X.f$ is of the form $\langle x, a\rangle$ for some $x$ and $a$. Accordingly, function extensionality is isomorphic to the following set:

$$(s_0, s_1 : \mathsf{Tm}(X.f, g)) \to \{\star \mid \forall x' : X.f.\, s_0\, x' = s_1\, x'\} \to \{\star \mid \forall x' : X.f.\, s_0\, x' = s_1\, x'\}$$

As this set is clearly inhabited, function extensionality is validated by this model. □

## References

[Awo18] Steve Awodey. "Natural models of homotopy type theory". In: *Mathematical Structures in Computer Science* 28.2 (2018), pp. 241–286. ISSN: 09601295. DOI: 10.1017/S0960129516000268. eprint: 1406.3219 (cit. on pp. 4, 5).

[Car78] John Cartmell. "Generalised Algebraic Theories and Contextual Categories". PhD thesis. University of Oxford, 1978 (cit. on p. 3).

[Dyb96] Peter Dybjer. "Internal type theory". In: *Types for Proofs and Programs*. Ed. by Stefano Berardi and Mario Coppo. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 120–134. DOI: 10.1007/3-540-61780-9_66 (cit. on pp. 3, 4).

[Hof97] Martin Hofmann. "Syntax and Semantics of Dependent Types". In: *Semantics and Logics of Computation*. Ed. by Andrew M. Pitts and P. Dybjer. Cambridge University Press, 1997, pp. 79–130. DOI: 10.1017/CBO9780511526619.004. URL: https://www.tcs.ifi.lmu.de/mitarbeiter/martin-hofmann/pdfs/syntaxandsemanticsof-dependenttypes.pdf (cit. on pp. 4, 6).

[Joy17] Andre Joyal. *Notes on Clans and Tribes*. 2017. URL: https://arxiv.org/abs/1710.10238 (cit. on p. 4).

[KKA19] Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. "Constructing Quotient Inductive-inductive Types". In: *Proc. ACM Program. Lang.* 3.POPL (Jan. 2019), 2:1–2:24. ISSN: 2475-1421. DOI: 10.1145/3290315 (cit. on p. 3).

[Mar84] Per Martin-Löf. *Intuitionistic type theory. Notes by Giovanni Sambin*. Vol. 1. Studies in Proof Theory. Bibliopolis, 1984, pp. iv+91. ISBN: 88-7088-105-9 (cit. on p. 12).

[Mar92] Per Martin-Löf. *Substitution calculus*. Notes from a lecture given in Göteborg. 1992 (cit. on p. 4).

[NPS90] B. Nordström, K. Petersson, and J.M. Smith. *Programming in Martin-Löf's Type Theory: An Introduction*. International series of monographs on computer science. Clarendon Press, 1990. ISBN: 9780198538141 (cit. on p. 6).

[Uem19] Taichi Uemura. "A General Framework for the Semantics of Type Theory". In: (Apr. 2019). eprint: 1904.04097 (math.CT). URL: https://arxiv.org/abs/1904.04097 (cit. on p. 3).

[Uem21] Taichi Uemura. "Abstract and Concrete Type Theories". PhD thesis. Amsterdam: Universiteit van Amsterdam, 2021. URL: https://www.illc.uva.nl/cms/Research/Publications/Dissertations/DS-2021-09.text.pdf (cit. on p. 3).

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study, 2013. URL: https://homotopytypetheory.org/book (cit. on p. 12).