Notes on Type Theory

[DRAFT: February 27, 2025]

Steve Awodey

with contributions from Andrej Bauer

Contents

Chapter 3

Dependent Type Theory

The Curry-Howard correspondence from Chapter **??** can be extended to natural deduction proofs in *first-order* logic, providing an extension of the "propositions as types/proofs as terms" idea from propositional logic to first-order logic (see [Sco70, How80] . In addition to simple types $A, B, ...$ representing propositions, one then has *dependent* types $x : A \vdash B(x)$ representing "propositional functions" or predicates. In addition to the simple type formers $A \times B$ and $A \to B$, one has dependent type formers $\Sigma_{x:A} B(x)$ and $\Pi_{x:A} B(x)$, representing the quantified propositions $\exists_{x:A} B(x)$ and $\forall_{x:A} B(x)$. As before, these types may have different terms $s, t : \Pi_{x:A} B(x)$, resulting from different proofs of the corresponding propositions, so that the calculus of terms again records more information than mere *provability*. Also as before, the resulting abstract structure turns out to be one that is shared by other categories *not* arising from logic—and now the coincidence is even more remarkable, because the structure at issue is a much more elaborate one. Where proofs in the propositional calculus gave rise to a cartesian closed category, the category of proof terms of first-order logic will be seen to be *locally cartesian closed*, a mathematical structure also shared by sheaves on a space, Grothendieck toposes, categories of fibrations, and other important examples.

Before stating a formal dependent type theory, we begin by infomally "categorifying" first-order logic with an abstraction (due to Lawvere [Law70]) called a *hyperdoctrine*. A hyperdoctrine is a contravariant functor $P : \mathcal{C}^{\mathsf{op}} \to \mathsf{Cat}$ (see Section 3.1), and there are in particular both poset-valued and "proper" category-valued ones. The former correspond to propositional and predicate logic, while the latter correspond more closely to dependent type theory, where the individual value categories $P(C)$ may be proper cartesian closed categories (rather than just Heyting algebras or CCC posets). Moreover, the reindexing functors along all projections $p_A : X \times A \to A$ in the index category $\mathcal{C}$ of contexts are also required to admit both left and right adjoints $\Sigma_A \dashv p_A^* \dashv \Pi_A$, according to Lawvere's adjoint analysis of quantification. An important *difference* between hyperdoctrines and dependent type theories, however, is that the indexing category of contexts in dependent type theory has not just finite products, but also some additional structure resulting from an operation of *context extension*, which takes as input a type in context $\Gamma \vdash A$ and returns a new context $(\Gamma, x : A)$, together with a substitution arrow $(\Gamma, x : A) \to \Gamma$. This is taking

the "propositions-as-types" idea even more seriously, by allowing every proposition $\Gamma \vdash \varphi$ in first-order logic to form a new type $\{\Gamma \vdash \varphi\}$, thus turning the objects $A \in P(C)$ in the value-categories of hyperdoctrine $(\mathcal{C}, P)$ into arrows $\{A\} \to C$ in $\mathcal{C}$.[1]

3.1 Hyperdoctrines

Given an algebraic signature, let $\mathcal{C}$ be the category of contexts, with (non-dependent) tuples of typed variables $\Gamma = (x_1 : C_1, ..., x_n : C_n)$ as objects, and as arrows $\gamma : \Delta \to \Gamma$ the n-tuples of terms $c_1 : C_1, \ldots, c_n : C_n$, all in context $\Delta = (y_1 : D_1, ..., y_m : D_m)$,

$$\Delta \vdash c_i : C_i\,, \quad 1 \leq i \leq n\,.$$

Composition is given by substitution of terms for variables,

$$\gamma \circ \delta = (c_1[d_1/y_1, \ldots, d_m/y_m], \ldots, c_n[d_1/y_1, \ldots, d_m/y_m],)$$

for $\delta = (d_1, \ldots, d_m) : \mathsf{E} \to \Delta$ with $\mathsf{E} = (z_1 : E_1, ..., z_k : E_k)$, and the identity arrows are the variables themselves (terms are identified up to α-renaming of variables, as in Lawvere algebraic theories, see Chapter **??**). The category $\mathcal{C}$ then has all finite products, essentially given by tupling.

For each object Γ, let $P(\Gamma)$ be the *poset* of all first-order formulas $(\Gamma \mid \varphi)$, ordered by entailment $\Gamma \mid \varphi \vdash \psi$ and identified up to provable equivalence $\Gamma \mid \varphi \dashv\vdash \psi$. Substitution of a term $\sigma : \Delta \to \Gamma$ into a formula $(\Gamma \mid \varphi)$ then determines a morphism of posets $\sigma^* : P(\Gamma) \to P(\Delta)$, which also preserves all of the propositional operations,

$$\sigma^*(\varphi \wedge \psi) = \varphi[\sigma/x] \wedge \psi[\sigma/x] = \sigma^*(\varphi) \wedge \sigma^*(\psi), \qquad \text{etc.}$$

(Exercise!). Moreover, since substitutions into formulas and terms commute with each other, $\tau^*\sigma^*\varphi = \varphi[\sigma \circ \tau/x]$, this action is *strictly* functorial, and so we have a contravariant functor

$$P : \mathcal{C}^{\mathsf{op}} \longrightarrow \mathsf{Heyt}$$

from the category of contexts to the category of Heyting algebras.

Now consider the quantifiers $\exists$ and $\forall$. Given a projection of contexts $p_X : \Gamma \times X \to \Gamma$, in addition to the pullback functor

$$p_X^* : P(\Gamma) \longrightarrow P(\Gamma \times X)$$

induced by weakening, there are the operations of quantification

$$\exists_X, \forall_X : P(\Gamma \times X) \longrightarrow P(\Gamma)\,.$$

By the rules for the quantifiers, these are indeed left and right adjoints to weakening,

$$\exists_X \dashv p_X^* \dashv \forall_X\,.$$

The Beck-Chevalley rules assert that substitution commutes with quantification, in the sense that $(\forall_x \varphi)[s/y] = \forall_x(\varphi[s/y])$, and similarly for $(\exists_x \varphi)$.

[1] [Law70] does just this.

Definition 3.1.1. A *(posetal) hyperdoctrine* consists of a Cartesian category $\mathcal{C}$ together with a contravariant functor

$$P : \mathcal{C}^{\mathsf{op}} \longrightarrow \mathsf{Heyt}\,,$$

such that for each $f : D \to C$ the action maps $f^* = Pf : PC \to PD$ have both left and right adjoints

$$\exists_f \dashv f^* \dashv \forall_f$$

that satisfy the Beck-Chavalley conditions.

Exercise 3.1.2. Verify that the syntax of first-order logic can indeed be organized into a hyperdoctrine in the way just described.

Examples

1. We just described the syntactic example of first-order logic. Indeed, for each first-order theory $\mathbb{T}$ there is an associated hyperdoctrine $(\mathcal{C}_\mathbb{T}, P_\mathbb{T})$, with the types and terms of $\mathbb{T}$ as the category of contexts $\mathcal{C}_\mathbb{T}$, and the formulas (in context) of $\mathbb{T}$ as "predicates", i.e. the elements of the Heyting algebras $\varphi \in P_\mathbb{T}(\Gamma)$. A general hyperdoctrine can be regarded as an abstraction of this example.

2. A hyperdoctrine on the index category $\mathcal{C} = \mathsf{Set}$ is given by the powerset functor

   $$\mathcal{P} : \mathsf{Set}^{\mathsf{op}} \longrightarrow \mathsf{Heyt}\,,$$

   which is represented by the Heyting algebra $\mathbb{2}$, in the sense that for each set I one has

   $$\mathcal{P}(I) \cong \mathsf{Hom}(I, \mathbb{2})\,.$$

   Similarly, for any *complete* Heyting algebra H in place of $\mathbb{2}$, there is a hyperdoctrine H-Set, with

   $$P_\mathsf{H}(I) \cong \mathsf{Hom}(I, \mathsf{H})\,.$$

   The adjoints to precomposition along a map $f : J \to I$ are given by

   $$\begin{aligned} \exists_f(\varphi)(i) &= \bigvee_{j \in J} (f(j) = i) \wedge \varphi(j)\,, \\ \forall_f(\varphi)(i) &= \bigwedge_{j \in J} (f(j) = i) \Rightarrow \varphi(j)\,, \end{aligned}$$

   where the value of $x = y$ in H is defined to be $\bigvee\{\top \mid x = y\}$.

   We leave it as an exercise to verify that this is hyperdoctrine, in particular to show that the Beck-Chevalley conditions are satisfied.

   Exercise 3.1.3. Show this.

3. For a related example, let $\mathbb{C}$ be any small index category and $\mathcal{C} = \widehat{\mathbb{C}}$, the category of presheaves on $\mathbb{C}$. An internal Heyting algebra H in $\mathcal{C}$, i.e. a functor $\mathbb{C}^{\mathsf{op}} \to \mathsf{Heyt}$, is said to be *internally complete* if, for every $I \in \mathcal{C}$, the transpose $\mathsf{H} \to \mathsf{H}^I$ of the projection $\mathsf{H} \times I \to \mathsf{H}$ has both left and right adjoints. Such an internally complete Heyting algebra determines a (representable) hyperdoctrine $P_{\mathsf{H}} : \mathcal{C} \to \mathsf{Set}$ just as for the case of $\mathcal{C} = \mathsf{Set}$, by setting $P_{\mathsf{H}}(C) = \mathcal{C}(C, \mathsf{H})$.

4. For any Heyting category $\mathcal{H}$ let $\mathsf{Sub}(C)$ be the Heyting algebra of all subobjects $S \rightarrowtail C$ of the object C. The presheaf $\mathsf{Sub} : \mathcal{H}^{\mathsf{op}} \to \mathsf{Heyt}$, with action by pullback, is then a hyperdoctrine, essentially by the definition of a Heyting category.

Remark 3.1.4 (Lawvere's Law)**.** In any hyperdoctrine $(\mathcal{C}, P)$, for each object $C \in \mathcal{C}$, we can determine an equality relation $=_C$ in each $P(C \times C)$, namely by setting

$$(x =_C y) \;=\; \exists_{\Delta_C}(\top)\,,$$

where $\Delta_C : C \to C \times C$ is the diagonal, $\exists_{\Delta_C} \dashv \Delta_C^*$, and $\top \in P(C)$. Displaying variables for clarity, if $\rho(x, y) \in P(C \times C)$ then $\Delta_C^* \rho(x, y) = \rho(x, x) \in PC$ is the contraction of the different variables, and the adjunction $\exists_{\Delta_C} \dashv \Delta_C^*$ can be formulated as the following two-way rule,

$$\frac{x : C \mid \top \vdash \rho(x, x)}{x : C, y : C \mid (x =_C y) \vdash \rho(x, y)} \tag{3.1}$$

which expresses that $(x =_C y)$ is the least reflexive relation on C. See [Law70] and Exercise **??** above.

Exercise 3.1.5. Prove the standard first-order laws of equality from the above hyperdoctrine formulation of Lawvere's Law (3.1).

Proper hyperdoctrines

Now let us consider some hyperdoctrines of a different kind. For any set I, let Set^I be the category of *families of sets* $(A_i)_{i \in I}$, with families of functions $(g_i : A_i \to B_i)_{i \in I}$ as arrows, and for $f : J \to I$ let us reindex along f by the precomposition functor $f^* : \mathsf{Set}^I \to \mathsf{Set}^J$, with

$$f^*((A_i)_{i \in I})_j = A_{f(j)}\,.$$

Thus we have a contravariant functor

$$P : \mathsf{Set}^{\mathsf{op}} \to \mathsf{Cat}$$

with $P(I) = \mathsf{Set}^I$ and $f^*(A : I \to \mathsf{Set}) = A \circ f : J \to \mathsf{Set}$.

Lemma 3.1.6. *The precomposition functors $f^* : \mathsf{Set}^I \to \mathsf{Set}^J$ have both left and right adjoints $f_! \dashv f^* \dashv f_*$ which can be computed by the formulas:*

$$\begin{aligned} f_!(A)_i &= \coprod_{j \in f^{-1}\{i\}} A_j \,, \\ f_*(A)_i &= \prod_{j \in f^{-1}\{i\}} A_j \,, \end{aligned} \tag{3.2}$$

for $A = (A_j)_{j \in J}$. Moreover, these functors satisfy the Beck-Chevally conditions.

A closely related example uses the familiar equivalence of categories $\mathsf{Set}^I \simeq \mathsf{Set}/_I$, where now the adjoints

$$f_! \dashv f^* \dashv f_* : \mathsf{Set}/_J \longrightarrow \mathsf{Set}/_I$$

to reindexing along $f : J \to I$ are (post-)composition, pullback, and "push-forward", respectively. In this case, the action of the pseudofunctor P is not strictly functorial, as it was for the case of $P(I) = \mathsf{Set}^I$. Note that the Beck-Chevalley conditions for such Cat-valued functors should now also be stated as (canonical) isomorphisms, rather than equalities as they were for poset-valued functors. In this way, when the individual categories $P(I)$ are proper, and not just posets, the entire hyperdoctrine structure may be weakened to include (coherent) isomorphisms, both in the functorial action of P, and in the B-C conditions. We will not spell out the required coherences here, but the interested reader may look up the corresponding notion of an *indexed-category*, which is a Cat-valued *pseudofunctor* (see [Joh03, B1.2]).

Example 3.1.7. Another example of a "proper" hyperdoctrine, with values in non-posetal (large!) categories, is the category of presheaves construction $\widehat{\mathbb{C}} = \mathsf{Set}^{\mathbb{C}^{\mathsf{op}}}$, where:

$$\begin{aligned} \mathcal{P} : \mathsf{Cat}^{\mathsf{op}} &\longrightarrow \mathsf{CAT} \,, \\ \mathbb{C} &\longmapsto \widehat{\mathbb{C}} \,. \end{aligned}$$

Here the action of $\mathcal{P}$ may be assumed to be *strictly* functorial, because it's given by precomposition. Nonetheless the B-C conditions must be stated as natural isos, because the adjoints $F_! \dashv F^* \dashv F_* : \widehat{\mathbb{D}} \longrightarrow \widehat{\mathbb{C}}$ for $F : \mathbb{D} \to \mathbb{C}$ are given by left and right Kan extensions, which need not be strictly functorial.

We shall consider several more examples of proper hyperdoctrines below. The internal logic of such categories generalizes and "categorifies" first-order logic, and is better described as dependent type theory. Proper hyperdoctrines $P : \mathcal{C}^{\mathsf{op}} \to \mathsf{Cat}$ are roughly related to dependent type theory in the way that posetal ones $P : \mathcal{C}^{\mathsf{op}} \to \mathsf{Pos}$ are related to FOL. There are actually two distinct aspects of this generalization: (1) the individual categories of "predicates" $P(C)$ are proper categories rather than mere posets, (2) the variation over the index category $\mathcal{C}$ of contexts (and its adjoints) is weakened accordingly to pseudo-functoriality. Each of these aspects plays an important role in dependent type theory and its categorical semantics.

First-Order Logic	Dependent Type Theory
Propositional Logic	Simple Type Theory

3.2 Dependently-typed lambda-calculus.

We give a somewhat informal specification of the syntax of the *dependently-typed λ-calculus* (see [Hof95, AG] for a more detailed exposition).

Dependent type theories have four standard forms of judgement

$$A : \mathtt{type}\,, \quad A \equiv B : \mathtt{type}\,, \quad a : A\,, \quad a \equiv b : A\,.$$

We refer to the triple equality relation $\equiv$ in these judgements as *definitional (or judgemental) equality*. It should not be confused with the notions of *(extensional and intensional) propositional equality* to be introduced below. A judgement J of one of the four above kinds can also be made relative to a *context* Γ of variable declarations, a situation that we indicate by writing $\Gamma \vdash J$. When stating deduction rules for such judgements we make use of standard conventions to simplify the exposition, such as omitting the (part of the) context that is common to premisses and conclusions of the rule.

To formulate the rules, we revisit the rules of simple type theory from Section **??** and adjust them as follows.

Judgements: The basic kinds of judgements are:

$$\Gamma\ \mathtt{ctx}\,, \qquad \Gamma \vdash A\ \mathtt{type}\,, \qquad \Gamma \vdash a : A\,.$$

along with the judgemental equalities of each kind:

$$\Gamma \equiv \Delta\ \mathtt{ctx} \qquad \Gamma \vdash A \equiv B\ \mathtt{type} \qquad \Gamma \vdash a \equiv b : A\,,$$

each of which are assumed to satisfy the usual laws of equality.

Contexts: These are formed by the rules:

$$\frac{}{(\cdot)\ \mathtt{ctx}} \qquad\qquad \frac{\Gamma \vdash A\ \mathtt{type}}{\Gamma, x : A\ \mathtt{ctx}}$$

Here it is assumed that x is a fresh variable, not already occurring in Γ. Note that, unlike in the simple type theory of the previous chapter, the order of the types occurring in a context now matters, since types to the right may depend on ones to their left.

Types: In addition to the usual *simple types*, generated from *basic types* by formation of *products* and *function types*, we may also have some *basic types in context*,

$$\text{Basic dependent types} \quad \Gamma_1 \vdash \mathtt{B}_1,\ \Gamma_2 \vdash \mathtt{B}_2,\ \cdots$$

where the contexts Γ need not be basic. Further dependent types are formed from the basic ones by the *sum* Σ and *product* Π *type formers*, using the *formation rules*:

$$\frac{\Gamma, x : A \vdash B\ \mathtt{type}}{\Gamma \vdash \Sigma_{x:A} B\ \mathtt{type}} \qquad \frac{\Gamma, x : A \vdash B\ \mathtt{type}}{\Gamma \vdash \Pi_{x:A} B\ \mathtt{type}}$$

Terms: As for the simple types, we assume there is a countable set of *variables* $x, y, z, \ldots$. We are also given a set of *basic constants*. The set of *terms* is then generated from variables and basic constants by the following grammar, just as for simple types:

$$\begin{aligned} \text{Variables} \quad & v ::= x \mid y \mid z \mid \cdots \\ \text{Constants} \quad & c ::= \mathtt{c}_1 \mid \mathtt{c}_2 \mid \cdots \\ \text{Terms} \quad & t ::= v \mid c \mid * \mid \langle t_1, t_2 \rangle \mid \mathtt{fst}\, t \mid \mathtt{snd}\, t \mid t_1\, t_2 \mid \lambda x : A\,.\, t \end{aligned}$$

The rules for deriving typing judgments are much as for simple types. They are of course assumed to hold in any context Γ.

- Each basic constant $\mathtt{c}_i$ has a uniquely determined type C_i (not necessarily basic):

$$\frac{}{\mathtt{c}_i : C_i}$$

- The type of a variable is determined by the context:

$$\frac{}{x_1 : A_1, \ldots, x_i : A_i, \ldots, x_n : A_n \vdash x_i : A_i}\ (1 \leq i \leq n)$$

- The constant $*$ has type $\mathtt{1}$:

$$\frac{}{* : \mathtt{1}}$$

- The typing rules for pairs and projections now take the form:

$$\frac{a : A \qquad b : B(a)}{\langle a, b \rangle : \Sigma_{x:A} B} \qquad \frac{c : \Sigma_{x:A} B}{\mathtt{fst}\, c : A} \qquad \frac{c : \Sigma_{x:A} B}{\mathtt{snd}\, c : B(\mathtt{fst}\, c)}$$

  We write e.g. $B(a)$ rather than $B[a/x]$ to indicate a substitution of the term a for the variable x in the type B. Similarly, we may write $\Sigma_{x:A} B(x)$ to emphasize the possible occurence of the variable x in B. We treat $A \times B$ as another way of writing $\Sigma_{x:A} B$, when the variable $x : A$ does not occur in the type B.

- The typing rules for application and λ-abstraction are now:

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

  We treat $A \to B$ as another way of writing $\Pi_{x:A} B$ when the variable $x : A$ does not occur in the type B.

The (β and η) equations between these terms are just as they were for simple types:

- Equations for unit type:

$$\frac{}{t \equiv * : \mathtt{1}}$$

- Equations for sum types:

$$\frac{u \equiv v : A \qquad s \equiv t : B(a)}{\langle u, s\rangle \equiv \langle v, t\rangle : \Sigma_{x:A} B}$$

$$\frac{s \equiv t : \Sigma_{x:A} B}{\mathtt{fst}\,s \equiv \mathtt{fst}\,t : A} \qquad\qquad \frac{s \equiv t : \Sigma_{x:A} B}{\mathtt{snd}\,s \equiv \mathtt{snd}\,t : A}$$

$$\frac{}{t \equiv \langle \mathtt{fst}\,t, \mathtt{snd}\,t\rangle : \Sigma_{x:A} B}$$

$$\frac{}{\mathtt{fst}\,\langle s, t\rangle \equiv s : A} \qquad\qquad \frac{}{\mathtt{snd}\,\langle s, t\rangle \equiv t : A}$$

- Equations for product types:

$$\frac{s \equiv t : \Pi_{x:A} B \qquad u \equiv v : A}{s\,u \equiv t\,v : B}$$

$$\frac{x : A \vdash t \equiv u : B}{(\lambda x : A\,.\,t) \equiv (\lambda x : A\,.\,u) : \Pi_{x:A} B}$$

$$\frac{}{(\lambda x : A\,.\,t)u \equiv t[u/x] : A} \qquad (\beta\text{-rule})$$

$$\frac{}{\lambda x : A\,.\,(t\,x) \equiv t : \Pi_{x:A} B} \text{ if } x \notin \mathsf{FV}(t) \qquad (\eta\text{-rule})$$

Equality types: Just as for first-order logic, for each type A we have a primitive *equality type*:

$$x, y : A \vdash \mathtt{Eq}_A(x, y)\ \mathtt{type}\,.$$

This is called *propositional equality*. For convenience, we may sometimes also write $x =_A y$ for $\mathtt{Eq}_A(x, y)$. Although they will turn out to be logically equivalent, the reader is warned not to confuse propositional and judgemental equality $x \equiv y : A$.

The formation, introduction, elimination, and computation rules for equality types are as follows:

$$\frac{s : A \qquad t : A}{s =_A t\ \mathtt{type}} \qquad\qquad \frac{a : A}{\mathtt{refl}_a : (a =_A a)}$$

$$\frac{p : s =_A t}{s \equiv t : A} \qquad\qquad \frac{p : s =_A t}{p \equiv \mathtt{refl}_s : (s =_A s)}$$

The elimination rule is known as *equality reflection*. We may say that two elements $s, t : A$ are *propositionally equal* if the type $s =_A t$ is inhabited. Thus the equality reflection rule says that if two terms are propositionally equal then they are judgementally equal.

Exercise 3.2.1. Show that two terms are propositionally equal if, and only if, they are judgementally equal.

Remark 3.2.2 (Identity types)**.** The formulation of the rules for equality just given is known as the *extensional* theory. There is also an *intensional* version, with different elimination (and computation) rules, to be considered in the next chapter. To help maintain the distinction between these three (!) different relations, the intensional version is sometimes called the *identity type* and written $\mathtt{Id}_A(s, t)$ instead. See [AG] for details.

Remark 3.2.3 (Variant rules for sum types)**.** Another formulation of the rules for Σ-types using a single dependent elimination rule is as follows:

$$\frac{z : \Sigma_{x:A} B \vdash C\ \mathtt{type} \qquad x : A, y : B(x) \vdash c(x, y) : C(\langle x, y\rangle)}{z : \Sigma_{x:A} B \vdash \mathtt{split}(z, c) : \Sigma_{x:A} B}$$

with the associated computation rule:

$$\frac{z : \Sigma_{x:A} B \vdash C\ \mathtt{type} \qquad x : A, y : B(x) \vdash c(x, y) : C(\langle x, y\rangle)}{x : A, y : B(x) \vdash \mathtt{split}(\langle x, y\rangle, c) \equiv c(x, y) : C(\langle x, y\rangle)}$$

These rules permit one to derive the simple elimination terms $\mathtt{fst}\, c$ and $\mathtt{snd}\, c$, and to prove the above computation rules for them. The η-rule is derived using a dependent elimination involving the $\mathtt{Eq}$-type.

Exercise 3.2.4. Prove the simple elimination rules for sum-types (involving $\mathtt{fst}\, c$ and $\mathtt{snd}\, c$) from the dependent ones (involving $\mathtt{split}$).

Remark 3.2.5 (The type-theoretic axiom of choice)**.** One of the oldest problems in the foundations of mathematics is the logical status of the Axiom of Choice. Is it a "Law of Logic"? A mathematical fact about sets? A falsehood with paradoxical consequences?

Per Martin-Löf discovered that the rules of constructive type theory that we have just presented actually suffice to *decide* this question in favor of "Law of Logic" in a certain sense [ML84] (see also [Tai68]). Since the statement of the type theoretic axiom of choice goes (slightly) beyond standard first-order logic, this arguably provides a resolution that also clarifies why the problem remained open for so long in conventional mathematics.

Under propositions as types, reading Σ as "there exists" and Π as "for all", a type such a $\Pi_{x:A}\Sigma_{y:B}R(x,y)$ can be regarded as a stating a proposition—in this case, "for all $x : A$ there is a $y : B$ such that $R(x, y)$". By Curry-Howard, such a "proposition" is then provable if it has a closed term $t : \Pi_{x:A}\Sigma_{y:B}R(x,y)$, which then corresponds to a proof, by unwinding the rules that constructed the term, and observing that they correspond to the usual natural deduction rules for first-order logic.

Of course, the rules of construction for terms correspond to provability only under a certain "constructive" conception of validity (see [Sco70]). Stated as follows,

$$\Pi_{x:A}\Sigma_{y:B}R(x,y) \to \Sigma_{f:A\to B}\Pi_{x:A}R(x,fx)\,, \tag{3.3}$$

the "type theoretic axiom of choice" may sound like the classical axiom of choice under the propositions as types interpretation, but this type is actually *provable* in (constructive) type theory, rather than being an axiom!

Exercise 3.2.6. Prove the type theoretic axiom of choice (3.3) from the rules for sum and product types given here.

Interaction of Eq with Σ and Π

The type theoretic axiom of choice Example 3.2.5, can be seen as a distributivity law for Σ and Π. It is in fact an *isomorphism of types*: there are terms going both ways, the composites of which are propositionally (and therefore definitionally!) equal to the identity maps (i.e. $\lambda x : X.\, x : X \to X$). It is natural to ask, how do the other type formers interact?

Consider first the result of combining Eq-types with Σ. We can show that for $s, t : A \times B$ there is always a term,

$$\mathtt{Eq}_{A\times B}(s,t) \to \big(\mathtt{Eq}_A(\mathtt{fst}\, s, \mathtt{fst}\, t) \times \mathtt{Eq}_B(\mathtt{snd}\, s, \mathtt{snd}\, t)\big)\,,$$

Moreover, there is a term in the other direction as well, and the composites are propositionally equal to the identity. By equality reflection, it therefore follows that these types are also *syntactically isomorphic*, in the sense just described. The same is true for dependent sums, although this is a bit more awkward to state, owing to the fact that $\mathtt{snd}\,(s) : B(\mathtt{fst}\, s)$ and $\mathtt{snd}\, t : B(\mathtt{fst}\,(t)$. However, since the first projection gives a term $p : \mathtt{Eq}_A(\mathtt{fst}\, s, \mathtt{fst}\, t)$ we have $\mathtt{fst}\, s \equiv \mathtt{fst}\, t$ and therefore $B(\mathtt{fst}\, s) \equiv B(\mathtt{fst}\, t)$, so that $\mathtt{Eq}_{B(\mathtt{fst}\, s)}(\mathtt{snd}\, s, \mathtt{snd}\, t)$ makes sense, and is in fact judgementally equal to $\mathtt{Eq}_{B(\mathtt{fst}\, t)}(\mathtt{snd}\, s, \mathtt{snd}\, t)$, so we can write:

$$\mathtt{Eq}_{\Sigma_{x:A}B}(s,t) \to \Sigma_{p:\mathtt{Eq}_A(\mathtt{fst}\, s, \mathtt{fst}\, t)}\mathtt{Eq}_{B(\mathtt{fst}\, s)}(\mathtt{snd}\, s, \mathtt{snd}\, t)\,,$$

Moreover, these types are again isomorphic.

For Π-types, given terms $f, g : A \to B$, we can form a term of type

$$\mathtt{Eq}_{A\to B}(f,g) \to \Pi_{x:A}\mathtt{Eq}_B(fx,gx)\,.$$

and again, this is an isomorphism of types. The corresponding law for dependent functions $f, g : \Pi_{x:A}B$ takes the form

$$\mathtt{Eq}_{\Pi_{x:A}B}(f,g) \to \Pi_{x:A}\mathtt{Eq}_{B(x)}(fx,gx)\,,$$

which of course is also an iso. Note that these last two isomorphisms say that two functions are equal just if they are so "pointwise". This principle is called *Function Extensionality*.

Finally, let us consider equality of equality types. Given any terms $a, b : C$ and $p, q : \mathtt{Eq}_C(a,b)$, what more can be said? The principle called *Uniqueness of Identity Proofs* (UIP) asserts that there is always a term of type

$$\mathtt{Eq}_{\mathtt{Eq}_C(a,b)}(p,q)\,.$$

Is there an argument for this principle, analogous to those for the equalities of terms of types Σ and Π? We shall return to this question in the setting of intensional type theory in the next chapter.

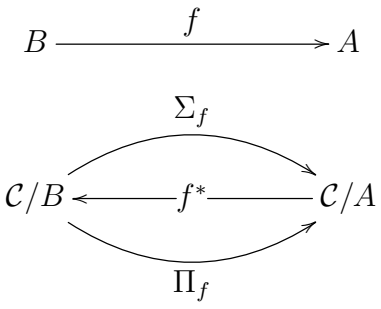
Exercise 3.2.7. Prove that extensional type theory satisfies (UIP).

3.3 Locally cartesian closed categories

Recall the following proposition from **??**.

Proposition 3.3.1. *The following conditions on a category $\mathcal{C}$ with a terminal object 1 are equivalent:*

1. *Every slice category $\mathcal{C}/A$ is cartesian closed.*
2. *For every arrow $f : B \to A$ the (post-) composition functor $\Sigma_f : \mathcal{C}/B \to \mathcal{C}/A$ has a right adjoint f^*, which in turn has a right adjoint Π_f.*



Such a category is called locally cartesian closed.

The notation of course anticipates the interpretation of DTT.

Proof. Suppose every slice of $\mathcal{C}$ is cartesian closed. It suffices to consider the case of (2) with $A = 1$, and to show that the functor $B^* : \mathcal{C} \to \mathcal{C}/_B$ with $B^*(X) = (\pi_2 : X \times B \to B)$ has both left and right adjoints $\Sigma_B \dashv B^* \dashv \Pi_B : \mathcal{C}/_B \to \mathcal{C}$. For Σ_B we can just take the forgetful functor. For $\Pi_B(p : X \to B)$ we use the CCC structure in $\mathcal{C}$ to form the map $p^B : X^B \to B^B$, which we then pull back along the point $'1_B' : 1 \to B^B$ that is the transpose of the identity map $1_B : B \to B$.

$$\begin{array}{ccc} \Pi_B(X) & \longrightarrow & X^B \\ \downarrow & \lrcorner & \downarrow{\scriptstyle p^B} \\ 1 & \xrightarrow['1_B']{} & B^B \end{array}$$

Conversely, if $\mathcal{C}$ is LCC, then in every slice $\mathcal{C}/_X$ we can define the product of $A \to X$ and $B \to X$ as $A \times_X B = \Sigma_A A^* B$ and the exponential as $(B^A)_X = \Pi_A A^* B$, and the universal properties are easily checked. □

Exercise 3.3.2. Verify the details of the proof just sketched for Proposition 3.3.1.

Basic examples of LCCCs

1. We have already seen the hyperdoctrine $\mathcal{C} = \mathsf{Set}$ and $P : \mathsf{Set}^{\mathsf{op}} \to \mathsf{Cat}$ where $P(I) = \mathsf{Set}^I$, with action of $f : J \to I$ on $A : I \to \mathsf{Set}$ by precomposition $f^*A = A \circ f : J \to \mathsf{Set}$, which is strictly functorial. There is an equivalent hyperdoctrine with the slice category $\mathsf{Set}/_I$ as the "category of predicates" and action by pullback $f^* : \mathsf{Set}/_I \to \mathsf{Set}/_J$. The equivalence of categories

$$\mathsf{Set}^I \simeq \mathsf{Set}/_I$$

allows us to use post-composition as the left adjoint $f_! : \mathsf{Set}/_J \to \mathsf{Set}/_I$, rather than the coproduct formula in (3.2). Indeed, this hyperdoctrine structure arises immediately from the locally cartesian closed character of Set. We have the same for any other LCC $\mathcal{E}$, namely the pair $(\mathcal{E}, \mathcal{E}/_{(-)})$ determines a hyperdoctrine, with the action of $\mathcal{E}/_{(-)}$ by pullback, and the left and right adjoints coming from the LCC structure.

2. Another familiar example of a hyperdoctrine arising from LCC structure is presheaves on a small category $\mathbb{C}$, where for the slice category $\widehat{\mathbb{C}}/_X$ we have another category of presheaves, namely

$$\widehat{\mathbb{C}}/_X \cong \widehat{\textstyle\int_{\mathbb{C}} X}\,,$$

on the category of elements $\int_{\mathbb{C}} X$. For a natural transformation $f : Y \to X$ we have a functor $\int f : \int Y \to \int X$, which induces a triple of adjoints

$$(\textstyle\int f)_! \dashv (\int f)^* \dashv (\int f)_* : \widehat{\int Y} \longrightarrow \widehat{\int X}\,.$$

These satisfy the Beck-Chevalley conditions up to isomorphism, because this indexed category is equivalent to the one coming from the LCC structure,

$$\widehat{\textstyle\int X} \;\simeq\; \widehat{\mathbb{C}}/_X\,,$$

which we know satisfies them.

Note that each of the categories $\widehat{\mathbb{C}}/_X$ is therefore also Cartesian closed, and moreover has coproducts $0, X+Y$, so it is a "categorified" Heyting algebra—although we don't make that part of the definition of a hyperdoctrine.

3. An instructive example of a hyperdoctrine that is not an LCC is the subcategory of Pos of posets and monotone maps, which we already met in Section **??**, with the "predicates" being the discrete fibrations. For each poset K, let us take as the category of predicates $P(K)$ the full subcategory $\mathsf{dFib}\_K \hookrightarrow \mathsf{Pos}/_K$ consisting of the *discrete fibrations*: monotone maps $p : X \to K$ with the "unique lifting property": for any x and $k \leq p(x)$ there is a unique $x' \leq x$ with $p(x') = k$. Since each category $\mathsf{dFib}/_K$ is equivalent to a category of presheaves $\mathsf{Set}^{K^{\mathsf{op}}}$, and pullback along any monotone $f : J \to K$ preserves discrete fibrations, and moreover commutes with the equivalences to the presheaf categories and the precomposition functor $f^* : \hat{K} \to \hat{J}$, we have a hyperdoctrine if only the Beck-Chevalley conditions hold. We leave this as an exercise for the reader. Finally, observe that dFib cannot be an LCC, simply because it does not have a terminal object; however, every slice of course does one, and so every slice $\mathsf{dFib}/_K$ is a CCC, and therefore an LCC.

4. An example formally similar to the foregoing is the non-full subcategory $\mathsf{LocHom} \hookrightarrow \mathsf{Top}$ of topological spaces and local homeomorphisms between them, which also lacks a terminal object, but each slice of which $\mathsf{LocHom}/X \simeq \mathsf{Sh}(X)$ is equivalent to the topos of *sheaves* on the space X, and is therefore CCC (and so LCCC).

5. Fibrations of groupoids. Another, similar, example of a hyperdoctrine not arising simply from an LCCC is the category Grpd of groupoids and homomorphisms, which is not LCC (cf. [Pal03]). We can however take as the category of predicates $P(G)$ the full subcategory $\mathsf{Fib}(G) \hookrightarrow \mathsf{Grpd}/_G$ consisting of the *fibrations* into G: homomorphisms $p : H \to G$ with the "iso lifting property": for any $h \in H$ and $\gamma : g \cong p(h)$ there is some $\vartheta : h' \cong h$ with $p(\vartheta) = \gamma$. Now each category $\mathsf{Fib}(G)$ is biequivalent to a category of presheaves of groupoids $\mathsf{Fib}(G) \simeq \mathsf{Grpd}^{G^{\mathsf{op}}}$. It is not so easy to show that this is a (bicategorical) hyperdoctrine; see [HS98]. This example will be important in the next chapter as a model of *intensional* dependent type theory.

Exercise 3.3.3. 1. Verify that the pullback of a discrete fibration $X \to K$ along a monotone map $f : J \to K$ exists in Pos, and is again a discrete fibration.

2. Verify the equivalence of categories $\mathsf{dFib}(K) \simeq \mathsf{Set}^{K^{\mathsf{op}}}$.

3. Show the Beck-Chavelley conditions for the indexed category of discrete fibrations of posets.

Exercise 3.3.4. Let $P : \mathcal{C}^{\mathsf{op}} \to \mathsf{Cat}$ be a hyperdoctrine for which there are equivalences $PC \simeq \mathcal{C}/C$, naturally in C, with respect to the left adjoints $\Sigma_f : \mathcal{C}/A \to \mathcal{C}/B$ for all $f : A \to B$ in $\mathcal{C}$. Show that $\mathcal{C}$ is then LCC.

Exercise 3.3.5. Show that any LCCC $\mathcal{C}$, regarded as a hyperdoctrine, has equality in the sense of Remark 3.1.4.

3.4 Functorial semantics of DTT in LCCCs

We begin by describing a "naive" interpretation of dependent type theory in a locally cartesian closed category which, although not strictly sound, is nonetheless useful and intuitive. In particular, it extends the functorial semantics of simple type theory in CCCs that we developed in the last chapter. In a subsequent section, we shall "strictify" the interpretation to one that is fully correct, but technically somewhat more complicated. See Remark 3.4.2 below.

Contexts Γ are interpreted as objects $[\![\Gamma]\!]$, and dependent types $\Gamma \vdash A$ as morphisms into the context $[\![\Gamma]\!]$. To begin, let $\mathcal{C}$ be an LCCC, and interpret the empty context as the terminal object, $[\![\cdot]\!] = 1$. Then for each closed basic type $\cdot \vdash \mathtt{B}$, interpret $[\![\cdot \vdash \mathtt{B}]\!] : [\![\mathtt{B}]\!] \to 1$. Proceeding by recursion, given any type in context $\Gamma \vdash A$, we shall have

$$[\![\Gamma \vdash A]\!] : [\![\Gamma, A]\!] \longrightarrow [\![\Gamma]\!]\,,$$

abbreviating $\Gamma, x : A$ to Γ, A. Note that in this way, we also interpret the operation of *context extension*, by taking the domain of the interpretation of a type in context. Weakening a type in context $\Gamma \vdash C$ to one $\Gamma, A \vdash C$ is interpreted as

$$[\![\Gamma, A \vdash C]\!] = \pi^*[\![\Gamma \vdash C]\!],$$

that is, the lefthand vertical map in the following pullback square, where the substitution $\pi : [\![\Gamma, A]\!] \to [\![\Gamma]\!]$ is the product projection.

$$\begin{array}{ccc} [\![\Gamma, A, C]\!] & \longrightarrow & [\![\Gamma, C]\!] \\ {\scriptstyle [\![\Gamma, A \vdash C]\!]}\downarrow & & \downarrow{\scriptstyle [\![\Gamma \vdash C]\!]} \\ [\![\Gamma, A]\!] & \xrightarrow[\pi]{} & [\![\Gamma]\!] \end{array} \tag{3.4}$$

Then, given $\Gamma, A \vdash B$, we may assume that we already have maps

$$[\![\Gamma, A, B]\!] \overset{[\![\Gamma, A \vdash B]\!]}{\longrightarrow} [\![\Gamma, A]\!] \overset{[\![\Gamma \vdash A]\!]}{\longrightarrow} [\![\Gamma]\!]\,,$$

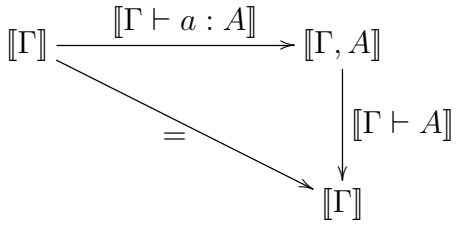
and we use the left and right adjoints to the pullback functor

$$[\![\Gamma \vdash A]\!]^* : \mathcal{C}/_{[\![\Gamma]\!]} \to \mathcal{C}/_{[\![\Gamma, A]\!]}$$

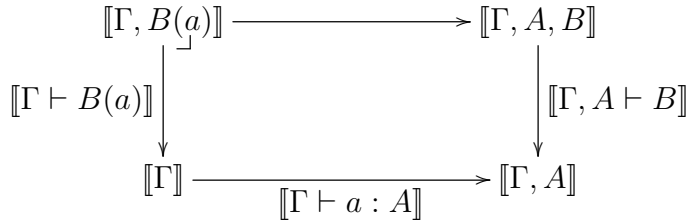
to interpret the eponymous type-forming operations:

$$\begin{aligned} [\![\Gamma \vdash \Sigma_{x:A} B]\!] &= \Sigma_{[\![\Gamma \vdash A]\!]}([\![\Gamma, A \vdash B]\!])\,, \\ [\![\Gamma \vdash \Pi_{x:A} B]\!] &= \Pi_{[\![\Gamma \vdash A]\!]}([\![\Gamma, A \vdash B]\!])\,. \end{aligned}$$

A term $\Gamma \vdash a : A$ is interpreted as a section:

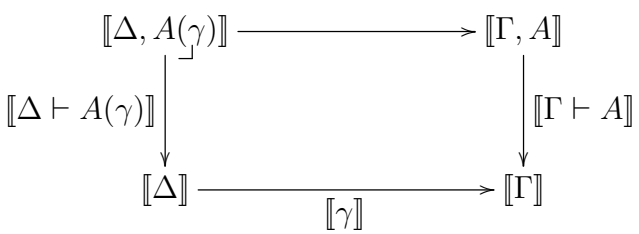


Finally, as in first-order logic, substitution of a term $\Gamma \vdash a : A$ for a variable $\Gamma, x : A$ in a dependent type $\Gamma, A \vdash B$ is interpreted by taking a pullback,



and similarly for substitution into terms.

More generally, given any substitution $\gamma : \Delta \to \Gamma$ (a tuple of terms $c_1, \ldots, c_n$ in context Δ of types those in $\Gamma = (x_1 : C_1, \ldots, x_n : C_n)$), we have a morphism $[\![\gamma]\!] : [\![\Delta]\!] \to [\![\Gamma]\!]$. Then, as in the substitution of a single term $\Gamma \vdash c : C$ for a variable $\Gamma, x : C$, we can obtain a pullback diagram along $[\![\gamma]\!]$:



The lefthand vertical map is then by definition the interpretation of the substituted type $\Delta \vdash A(\gamma)$. The interpretation of a substitution into a term $\Delta \vdash a(\gamma) : A(\sigma)$ is similarly induced by pullback.

Finally, we interpret an equality type $x : A, y : A \vdash \mathtt{Eq}_A(x, y)$ as the diagonal of the interpretation of A,

$$[\![x, y : A \vdash \mathtt{Eq}_A(x, y)]\!] = \Delta_{[\![A]\!]} : [\![A]\!] \longrightarrow [\![A]\!] \times [\![A]\!] \,.$$

As a map of contexts we then have

$$[\![A, A, \mathtt{Eq}_A(x, y)]\!] = [\![A]\!] \longrightarrow [\![A]\!] \times [\![A]\!] = [\![A, A]\!] \,.$$

For $\Gamma, A, A \vdash \mathtt{Eq}_A(x, y)$, with a context Γ, we take the diagonal of the map $[\![\Gamma \vdash A]\!] : [\![\Gamma, A]\!] \longrightarrow [\![\Gamma]\!]$ as an object in the slice category over $[\![\Gamma]\!]$.

Proposition 3.4.1 ([See84])**.** *The rules of dependent type theory are sound with respect to the naive interpretation in any LCCC, modulo the following Remark 3.4.2.*

Remark 3.4.2 (Coherence)**.** Although it is correct according to the intuition of slice categories in an LCCC, the naive interpretation of substitution as pullback leads to a well-known *coherence problem* [Hof95] for the interpretation of dependent type theory, arising from the fact that pullbacks are only determined up to (canonical) isomorphism. For example, in the foregoing situation, given another substitution $\Phi \vdash \delta : \Delta$, we obtain the two-pullback diagram:

$$\begin{array}{ccccc}
[\![\Phi, A(\gamma)(\delta)]\!] & \longrightarrow & [\![\Delta, A(\gamma)]\!] & \longrightarrow & [\![\Gamma, A]\!] \\
\Big\downarrow{\scriptstyle [\![\Phi \vdash A(\gamma)(\delta)]\!]} & & \Big\downarrow{\scriptstyle [\![\Delta \vdash A(\gamma)]\!]} & & \Big\downarrow{\scriptstyle [\![\Gamma \vdash A]\!]} \\
[\![\Phi]\!] & \xrightarrow[[\![\Phi \vdash \delta : \Delta]\!]]{} & [\![\Delta]\!] & \xrightarrow[[\![\Delta \vdash \gamma : \Gamma]\!]]{} & [\![\Gamma]\!]
\end{array}$$

However, the composition of substitutions $[\![\Delta \vdash \gamma : \Gamma]\!] \circ [\![\Phi \vdash \delta : \Delta]\!]$ across the bottom is the map $[\![\Phi \vdash \gamma(\delta) : \Gamma]\!]$, and so there is another option for the vertical map on the left, namely the single pullback:

$$\begin{array}{ccc}
[\![\Phi, A(\gamma(\delta))]\!] & \longrightarrow & [\![\Gamma, A]\!] \\
\Big\downarrow{\scriptstyle [\![\Phi \vdash A(\gamma(\delta))]\!]} & & \Big\downarrow{\scriptstyle [\![\Gamma \vdash A]\!]} \\
[\![\Phi]\!] & \xrightarrow[[\![\Phi \vdash \delta(\gamma) : \Gamma]\!]]{} & [\![\Gamma]\!]
\end{array}$$

Since pullbacks are unique up to iso, there is of course a canonical isomorphism

$$[\![\Phi, A(\gamma)(\delta)]\!] \;\cong\; [\![\Phi, A(\gamma(\delta))]\!]$$

over the base $[\![\Phi]\!]$, but there is no reason for these two objects (and their associated projections) to be the same. To put the matter succinctly, the action of substitution between contexts on dependent types is *strictly* functorial, but the action of pullback on slice categories is only a *pseudo*functor.

The naive LCCC interpretation, with substitution as pullback, is thus only sound "up to (canonical) isomorphism". The problem becomes more acute in the case of intensional type theory, where the interpretation of certain type formers is not even determined up to isomorphism. We shall consider several solutions to this problem in detail in Section **??** below. For the remainder of this chapter on extensional type theory, however, we can continue to work "up to (canonical) isomorphism" without worrying about coherence.

As was done for simple type theory in Section **??**, we can also again develop the relationship between the type theory and its models using the framework of *functorial semantics*. This is now a common generalization of λ-theories, modeled in CCCs, and first-order logic, modeled in Heyting categories. The first step is to build a syntactic *classifying category* $\mathcal{C}_{\mathbb{T}}$ from a theory $\mathbb{T}$ in dependent type theory, which we then show classifies $\mathbb{T}$-models in LCCCs. We omit the now essentially routine details (given the analogous cases already considered), and merely state the main result, the proof of which is also analogous to the previous cases. A detailed treatment can be found in the seminal paper [See84].

Theorem 3.4.3. *For any theory $\mathbb{T}$ in dependent type theory, the locally cartesian closed syntactic category $\mathcal{C}_{\mathbb{T}}$ classifies $\mathbb{T}$-models, in the sense that for any locally cartesian closed category $\mathcal{C}$ there is an equivalence of categories*

$$\mathsf{Mod}\big(\mathbb{T}, \mathcal{C}\big) \;\simeq\; \mathsf{LCCC}\big(\mathcal{C}_{\mathbb{T}}\,, \mathcal{C}\big)\,, \tag{3.5}$$

naturally in $\mathcal{C}$. The morphisms of $\mathbb{T}$-models on the left are the isomorphisms of the underlying structures, and on the right we take the natural isomorphisms of LCCC functors.

As a corollary, again as before, we have that dependent type theory is *complete* with respect to the semantics in locally cartesian closed categories, in virtue of the syntactic construction of the classifying category $\mathcal{C}_{\mathbb{T}}$. Specifically, any theory $\mathbb{T}$ has a canonical interpretation $[-]$ in the syntactic category $\mathcal{C}_{\mathbb{T}}$ which is *logically generic* in the sense that, for any terms $\Gamma \vdash s : A$ and $\Gamma \vdash t : A$, we have

$$\begin{aligned} \mathbb{T} \vdash (\Gamma \vdash u \equiv t : A) \;&\iff\; [\Gamma \vdash u : A] = [\Gamma \vdash t : A] \\ &\iff\; [-] \models (\Gamma \vdash s \equiv t : A)\,. \end{aligned}$$

Thus, for the record, we have:

Proposition 3.4.4. *For any dependently typed theory $\mathbb{T}$,*

$$\mathbb{T} \vdash (\Gamma \vdash u \equiv t : A) \quad \textit{if, and only if,} \quad \mathcal{C}_{\mathbb{T}} \models (\Gamma \vdash u \equiv t : A)\,.$$

Of course, the syntactic model $[-]$ in $\mathcal{C}_{\mathbb{T}}$ is the one associated under (3.5) to the identity functor $\mathcal{C}_{\mathbb{T}} \to \mathcal{C}_{\mathbb{T}}$, *i.e.* it is the *universal* one. It therefore satisfies an equation just in case the equation holds in *all* models, by the classifying property of $\mathcal{C}_{\mathbb{T}}$, and the preservation of satisfaction of equations by LCCC functors (as in Proposition **??**).

Corollary 3.4.5. *For any dependently typed theory* $\mathbb{T}$*,*

$$\mathbb{T} \vdash (\Gamma \vdash u \equiv t : A) \quad \textit{if, and only if,} \quad M \models (\Gamma \vdash u \equiv t : A) \textit{ for every LCCC model } M.$$

Moreover, a closed type A *is inhabited* $\vdash a : A$ *if, and only if, there is a point* $1 \to [\![A]\!]^M$ *in every model* M*.*

The embedding and completeness theorems of the previous chapter, with respect to general presheaf models, Kripke models, and topological and sheaf semantics can also be extended to dependently typed theories. See [AR11, Awo00] for details.

Exercise 3.4.6. In the internal logic of an LCCC $\mathcal{E}$, show that the category of types in context $\Gamma \in \mathcal{E}$ is equivalent to the slice category $\mathcal{E}/_{\Gamma}$. (*Hint*: use `Eq`-types.)

Bibliography

[AG] C. Angiuli and D. Gratzer. Principles of dependent type theory. Online at `https://carloangiuli.com/courses/b619-sp24/notes.pdf`. Version 2024-11-26.

[AR11] S. Awodey and F. Rabe. Kripke semantics for Martin-Löf's extensional type theory. *Logical Methods in Computer Science*, 7(3):1–25, 2011.

[Awo00] Steve Awodey. Topological representation of the λ-calculus. *Mathematical Structures in Computer Science*, 10:81–96, 2000.

[Hof95] Martin Hofmann. Syntax and semantics of dependent types. In *Semantics and logics of computation*, volume 14 of *Publ. Newton Inst.*, pages 79–130. Cambridge University Press, Cambridge, 1995.

[How80] William A. Howard. The formulae-as-types notion of construction. In Jonathan P. Seldin and J. Roger Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. 1980. Reprinted from 1969 manuscript.

[HS98] Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In *Twenty-five years of constructive type theory (Venice, 1995)*, volume 36 of *Oxford Logic Guides*, pages 83–111. Oxford Univ. Press, New York, 1998.

[Joh03] P.T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium, 2 vol.s*. Number 43 in Oxford Logic Guides. Oxford University Press, 2003.

[Law70] F.W. Lawvere. Equality in hyperdoctrines and comprehension schema as an adjoint functor. *Proceedings of the AMS Symposium on Pure Mathematics XVII*, pages 1–14, 1970.

[ML84] Per Martin-Löf. *Intuitionistic type theory*, volume 1 of *Studies in Proof Theory*. Bibliopolis, 1984.

[Pal03] Erik Palmgren. Groupoids and local cartesian closure. 08 2003. unpublished.

[Sco70] Dana S. Scott. Constructive validity. In M. Laudet, D. Lacombe, L. Nolin, and M. Schützenberger, editors, *Symposium on Automatic Demonstration*, volume 125, pages 237–275. Springer-Verlag, 1970.

[See84] R. A. G. Seely. Locally cartesian closed categories and type theory. *Mathematical Proceedings of the Cambridge Philosophical Society*, 95(1):33, 1984.

[Tai68] William W. Tait. Constructive reasoning. In *Logic, Methodology and Philos. Sci. III (Proc. Third Internat. Congr., Amsterdam, 1967)*, pages 185–199. North-Holland, Amsterdam, 1968.