Type Theory

Trey Plante

May 8, 2024

These notes are a work in progress; there are probably errors and incorrect statements.

1 Type Theory

Type theory is a deductive system that has one basic notions: types. The deductive system is constructed with a set of inference rules that can be combined to form derivations.

Definition 1.1. A type is ... A term is ...

Definition 1.2. An inference rule is...

Definition 1.3. A context is ...

Definition 1.4. A *judgment* is $\Gamma \vdash J$, where Γ is a context and J is some assertion that is given from Γ. The possibilities for J depend on the specific type theory being defined. We can think of the context Γ as a set of assumptions. Judgments that appear in all type theories are term and type formation ($\Gamma \vdash a : A$ and $\Gamma \vdash A$) and term and type equality ($\Gamma \vdash a \equiv b$ and $\Gamma \vdash A \equiv B$). Judgments form the premises and conclusions of an inference rule.

Definition 1.5. A derivation is ...

Definition 1.6. An *admissible rule* is an inference rule that can be derived from the basic rules of the type theory. In other words, an admissible rule is not a fundamental part of the type system itself but can be proved as a theorem using the existing rules.

2 Dependent Type Theory

2.1 Judgements

Definition 2.1 (Harper)**.** A *synthetic judgment* is a judgment that requires evidence. For example, saying A is true requires a proof of A; we need to search a for a proof to check the statement.

Definition 2.2 (Harper)**.** An *analytic judgment* is a judgment that requires no further evidence. For example, saying a is type A, $a : A$ is self-evident; we can check this statement from what is given.

Remark 2.3. Whenever we say "judgment" without a prefix we are referring to the type-theoretic judgment from Definition 1.4. The above two forms of "judgments" refer to the internal judgments that occur within a type-theoretic judgment.

Remark 2.4. Analytic judgments are the kind of assumptions that appear in the context of a type theory judgment in a context Γ, and they also can appear on the other side of the turnstile (as an internal conclusion). A synthetic judgment can only appear on the right hand side of the turnstile.

2.2 Equalities

Definition 2.5. Definitional (judgmental) equality is a syntactic notion of equality decided by the normalization or reduction rules of a type theory. If two terms are definitionally equal, they can be freely substituted for each other in any context without changing the meaning of the program. The notes denote definitional equality for two terms $a, b : A$ as $a \equiv b$.

Remark 2.6 (?). Definitional equality can be thought of as an equality of *sense* (a la Frege). Definitional equality is an analytic judgment.

Definition 2.7. Propositional equality is a semantic notion of equality that is a proposition or type in the type theory. Two terms are propositionally equal if there exists a proof or a witness inhabiting a type expressing propositional equality between the two terms. Propositional equality is denoted by $a =_A b$.

Remark 2.8 (?). Propositional equality can be thought of as an equality of *reference*. Propositional equality is a synthetic judgment.

Example 2.9 (?). In classical mathematics, there are two propositions: true and false. Every single mathematical theorem refers to true or false, but they have different information content. Thus, all true mathematical theorems are propositionally equal, but they are not definitionally equal.

Remark 2.10. Definitional equality is automated in Agda through the normalization of terms, while proving propositional equality is a task for the user. Definitional equality is a stronger notion than propositional equality and is usually decidable.The main distinction is that definitional equality is a syntactic notion built into the type theory, while propositional equality is a semantic notion expressed within the type theory itself.

Remark 2.11. The types in HoTT are treated intensionally rather than extensionally. Practically this means that we should think of types as being distinguished by the descriptions that define them, rather than by their contents. Thus, for example, the types 'positive integer less than 3' and 'integer exponent

n for which $a^n + b^n = c^n$ has a solution in the positive integers' are fundamentally distinct types, even though it can be proved that they are extensionally equal. This extends to empty types as well: the type 'even divisor of 9' is a distinct type from 'even divisor of 11', even though both types are uninhabited. This has the advantage that the basic elements of the theory are closer to the descriptions of mathematical entities and the mathematical concepts that are directly used in practice.

Definition 2.12. Equivalence ...

2.3 Dependent Type

Definition 2.13. A family of types ...

Definition 2.14. The dependent function type ...

Definition 2.15. The dependent pair type ...

3 Homotopy Type Theory

3.1 Brief Homotopy Theory Background

This subsection is provided to develop intuition for the subsequent discussions. Homotopy theory is the study of spaces up to continuous distortions; it focuses on the properties of spaces that are preserved by continuous deformations. Homotopy theory considers topological spaces and the continuous functions between them.

Definition 3.1. A continuous function ...

Definition 3.2. Let $f, g : X \to Y$ be any two continuous functions. A *homotopy* between f and g is a continuous function $h : [0,1] \times X \to Y$ such that $\forall x \in X, h(0,x) = f(x)$ and $h(1,x) = g(x)$. We can interpret h as providing a continuous interpolation from f to g. If an h exists, then f and g are homotopic, denotes $f \sim g$.

Definition 3.3. Two spaces X and Y are *homotopy equivalent* if there are $f : X \to Y$ and $f' : Y \to X$ such that $f' \circ f \sim id_X$ and $f \circ f'$ id_Y. Homotopy equivalence is an equivalence relation, so we can define the equivalence class $[X]$ of all topological spaces homotopy equivalent to X. $[X]$ is the *homotopy type* of X.

HoTT describes types as spaces, terms as points in space (or, functions from the single point into the space), and identifications between terms as paths between points (functions out of [0,1] ti the space having those points as end-points).

3.2 Path Induction and Identity

For any two $a : A$ and $b : B$ we can state that a and b are equal; terms of different types cannot be equal. The type corresponding to the proposition that $a : A$ and $b : B$ are equal is written $Id_A(a, b)$ or $a =_A b$. A term $x : Id_A(a, b)$ is called an *identification* of a and b. The only identifications that are guaranteed to exist are the self-identifications.

Definition 3.4. Let A be a type and consider $a : A$. We define the identity type at a as an inductive family type $a =_A x$ indexed by $x : A$ with the constructor $refl_a a \equiv_A a$. The identity type is the HoTT version of propositional equality discussed in Definition 1.4.

The elimination rule for the identity type is called *path induction*. The idea is that to prove a property holds for all identifications between terms of some type A it suffices to show that the problem holds for all self-identifications $refl_a$. That is if, for all properties, if the base case satisfies some property, then all identifications do.

The *total identity type* is $\Sigma_{s,t:A} Id_A(s, t)$. The terms of the total identity type are triples (x, y, p) where $x : A$, $y : A$, and $p : Id_A(x, y)$. The type is guaranteed to be inhabited because of the term $(x, x, refl_x)$.

Consider a predicate Z on the total identity type. Path induction says that if Z is satisfied by $refl_x$ for every $x : A$ $(\Pi_{x:A} Z(x, x, refl_x))$, then we can produce a certificate of $Z(a, b, p)$ for every triple (a, b, p). Specifically, given $z : \Pi_{x:A} Z(x, x, refl)$ there is a function that takes $(a, b, p) : \Sigma_{s,t:A} Id_A(s, t)$ and returns a term of $Z(a, b, p)$, giving z when given $(a, a, refl_a)$.

To justify path induction we can argue that there is a structure among identity types that ensures that properties held by self-identification must also be shared by the other identifications. We can think of it being sufficient to define a functions behavior at a distinguished subset of elements from which the function is ensured to be well-defined for all other elements (like defining a linear function on the vector space by specifying its output on the basis vectors).

Homotopically we can argue that $Id_A(a, b)$ corresponds to the path space consisting of all paths from the point a to the point b in space A. Given a point a in space A $(a : A)$ the path corresponding to the self-identification $refl_A$ s the function $k_a : [0, 1] \to A$ that sends every point in the interval to a. Given any point b in A $(b : A)$ and a path p from a to b, there is a homotopy h between k_a and p via $h(t, x) = p(t \times x)$. This homotopically justifies path induction. Any properties that apply to k_a up to homotopy must apply to p up to homotopy. In short, if we are free to vary one or both ends of a path, then any path can be retracted to a constant path at some point.

Every term of a type is equal to the output of one of the constructors for that type. For example, for every token $c : A + B$ we have either a token of type $Id_{A+B}(c, inl(a))$ for some $a : A$ or a token of type $Id_{A+B}(c, inr(b))$ for some

$b : B$. Formally,

$$\Pi_{c:A+B}(\Sigma_{a:A} Id_{A+B}(c, inl(a))) + \sigma \Sigma_{b:B} Id_{A+B}(c, inr(b))$$

These are the *uniqueness principles* for the respective types. For example, we can say that there is one term up to identity in the Unit type with $\Pi Id_1(*, i)$.

These formal statements say that the term constructors output every token of a type up to identity. Thus, we can say that the constructor $refl$ gives us all the identifications up to identity.

Definition 3.5. Let A be a type and $a : A$. Then, we define $\Sigma_{x:A} Id_A(a, x)$ to be the *based identity type* whose terms are pairs (b, p) where $b : A$ and $p : Id_A(a, b)$. We denote the based identity type at $a : A$ as $\mathcal{E}_a$.

Then, we can make the statement formal by saying that

$$\Pi_{(b,p):\mathcal{E}_a} Id_{\mathcal{E}_a}((a, refl_a), (b, p))$$

This is the uniqueness principle for identity types, which says that the only token up to identity in $\mathcal{E}_a$ is $(a, refl_a)$.

3.3 Contractible Types

Definition 3.6. The type A is *contractible* if it comes equipped with an element of the type

$$\text{is-Contr}(A) \coloneqq \Sigma_{(c:A)} \Pi_{(x:A)} a = x$$

Given $(c, C) : \text{is-Contr}(A)$, $c : A$ is the *center of contraction* for A and $C : \Pi_{(x:A)}$ is the *contraction of* A.

Remark 3.7 (Intuition). A type is contractible if there exists $a : A$ such that for every term $x : A$, there is a path $p : a = x$. We can say that a type is contractible if it is equivalent to the unit type, $\mathbf{1}$. A contractible type can be thought of as a type that contains only one unique element up to homotopy [?].

Remark 3.8 (Properties). A few points about contractible types:

- All contractible types are equivalent to each other and the unit type.
- Contractible types are a terminal object in the category of types and functions.
- The path space between any two elements in a contractible type is itself contractible.

Remark 3.9 (Examples). A few examples of contractible types.

- The unit type, $\mathbf{1}$, is contractible.
- The type $\Sigma_{(a:A)} A$ is contractible for any type A. This type represents the idea of a pointed type, where $a : A$ serves as the distinguished point [?].

- The type of paths between two equal terms $x = x$ is contractible for any $x : A$.

4 Truncation Levels and Homotopy Types

Remark 4.1 (Truncation Levels). Truncation levels are a way to classify types by their equality structures. They are defined inductively, with each level being a property a type may or may not satisfy. This means the truncation levels form a cumulative hierarchy, meaning that every type at a given truncation level also belongs to all higher truncation levels.

Remark 4.2 (Examples). The first four n-types (n is a truncation level) are as follows.

- The -2-types are the contractible types. A type is a -2-type if it has one inhabitant up to equality. Thus, all inhabitants of a -2-type are equal and the equality type of a -2-type is always inhabited (contractible). The unit type is an exmaple.
- The -1-types are the proposition types, or mere propositions. A type is a -1-type if any two of its inhabitants are equal. Propositions are either inhabited or uninhabited, and equality between inhabitants is always provable. The unit and empty types are examples. The equality type of a proposition is always either inhabited or uninhabited.
- The 0-types are sets. A type is a set if equality between any two inhabitants is a proposition. Thus, any two inhabitants of a set are equal or unequal, and the equality is unique. The natural numbers and Booleans are examples. The equality type of a set is always a -1-type.
- The 1-types are groupoids. A type is a groupoid if equality between any two inhabitants is a set. Thus, groupoids allow for multiple equalities between sets. The type of sets and the type of paths are examples. The equality type of a 1-type is always a 0-type.

Remark 4.3 (Homotopy Types). Homotopy types are a way of describing types as spaces with a homotopy structure. Every type is a homotopy type because a type is space with a particular shape determined by its equalities. Homotopy types are not directly classified by integers, but rather by their homotopy-theoretic properties, such as contractibility, connectedness, or homotopy groups. The concept of homotopy types allows for the interpretation of types as topological spaces and the use of homotopy-theoretic reasoning in type theory.

Remark 4.4 (Relationship). Truncation levels can be seen as a way to describe the homotopy type of a space in terms of its homotopy n-types. A type is an n-type if all of its homotopy groups above dimension n are trivial. For example, a (-1)-type (proposition) corresponds to a contractible space, a 0-type (set) corresponds to a discrete space, and a 1-type (groupoid) corresponds to a space

with only non-trivial fundamental groupoid. Truncation levels provide a way to express the complexity of a homotopy type in terms of the highest non-trivial homotopy dimension.

5 Cubical Type Theory

The following section comes from [[CCHM16],[ABC$^+$21],[CMS20]].

6 Scratch and Todos

[TODO: prove all contractible types are equvalent to unit type in agda] [TODO: notes on truncation] [TODO: notes on apd, tpt, ap, decidable equality] [TODO prove Z = N + N]

References

[ABC$^+$21] Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Robert Harper, Kuen-Bang Hou (Favonia), and Daniel R. Licata, *Syntax and models of cartesian cubical type theory*, Mathematical Structures in Computer Science **31** (2021), no. 4, 424–468.

[CCHM16] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg, *Cubical type theory: a constructive interpretation of the univalence axiom*, 2016.

[CMS20] Evan Cavallo, Anders Mörtberg, and Andrew W Swan, *Unifying Cubical Models of Univalent Type Theory*, 28th eacsl annual conference on computer science logic (csl 2020), 2020, pp. 14:1–14:17.