1 Dependent Type Theory

1.1 Equalities

Definition 1.1. Definitional 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 eachother in any context without changing the meaning of the program.

Definition 1.2. 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.

Remark 1.3. 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 1.4. 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.

1.2 Dependent Type

Definition 1.5. A family of types ...

Definition 1.6. The dependent function type ...

Definition 1.7. The dependent pair type ...

2 Homotopy Type Theory

2.1 Path Induction

2.2 Contractible Types

Definition 2.1. 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 2.2 (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 2.3 (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 2.4 (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 $x : A$.

[TODO: prove all contractible types are equvalent to unit type in agda]