Building a Type Theory from the Basis of Category Theory

Donald Pinckney

June 15, 2018

1 Introduction

A *type theory* is a formal system in which *terms* are classified by *types*. For example, in a type theory (and in the one developed here) one could express that x has the type of a natural number, written $x : \mathbb{N}$, and from this deduce that $+(x, 1)$ also has the type of a natural number. The goal of type theory is to provide a framework for performing these deductions, with the hope of the framework being able to be mechanically run by computers while being general enough to provide a foundation for mathematics.

2 Basic Syntax of Terms

Before going further, we should describe formally some of the basic items in type theory. Most importantly, we must discuss what exactly a term consists of. At the moment, we will develop a term to be only the necessary parts we need right now, and add to it later. We define a term to be one of three types: *constants*, *a free variable*, and *function application*. How these behave in the type theory will be discussed soon, but for now we provide a concrete syntax syntax for terms in Figure 1. Note that there is only one free variable allowed, x.

As an example, the term $+(x, 1)$ from above would be analyzed as a function application, with the function name the constant $+$, and the arguments consisting of the free variable x and the constant 1.

In addition, a type is syntactically represented by a term as well. In the non-dependent type theory developed here types will only consist of function applications and constants, but in a dependent type theory use of the free variable would be crucial. In addition, in this type theory only certain constants can be used in terms, and only other disjoint constants can be used in types.

⟨*term*⟩ ::= ⟨*free variable*⟩
  | ⟨*constant*⟩
  | ⟨*function application*⟩
  | ⟨*substitution*⟩

⟨*free variable*⟩ ::= '`x`'

⟨*constant*⟩ ::= **identifier** \ '`x`'

⟨*function application*⟩ ::= ⟨*term*⟩ '`(`' ⟨*term*⟩ (, ⟨*term*⟩)* '`)`'

Figure 1: First version of term grammar

3 Rules of Inference

The framework of deduction of type theory is specified by several *rules of inference*. A rule of inference consists of some number (possibly no) *premises*, and a *conclusion*, written as the form:

$$\frac{P_1 \qquad \cdots \qquad P_n}{C}$$

If all premises $P_1, \cdots, P_n$ can be derived, then C can be derived.

In type theory there are four types of statements that premises or a conclusion can consist of. They will be explained briefly here, and many examples will be seen later.

a) Type Existence Judgement: Since a type theory consists of a collection of types which are used to classify terms, we need a means to say if something is a type. This is represented formally as the statement "T Type", where T is a string which will name the type.

b) Type Equality Judgement: There may be multiple ways to syntactically represent a single given type. A type equality judgement is used to express that two strings name the same type, and is written formally as "$T = U$", where T and U are strings naming types.

c) Term Typing Judgement: The main purpose of type theory is to classify terms according to types. This is done by deriving a judgement that states that a given term is classified by a given type. Formally this is written as "$t : T$".

d) Term Equality Judgement: Finally, we also wish to tell when two terms are semantically equal, although syntactically they are not identical. This is done by deriving a statement that two terms are judgmentally equal in a given type. This is written as "$t_1 = t_2 : T$", where t_1 and t_2 are terms, and T is the type in which they are equal.

In addition, each statement also contains a *context*, which is written as "$[C]''$, where C is a type. This is used to keep track of the type of the free variable in an expression. Since only terms can contain free variables in this theory, only term typing judgements and term equality judgements will make meaningful use of the context. Contexts for type existence and type equality statements will be elided.

To construct our type theory and choose what rules of inference it should have, we will look at category theory. Specifically, the plan is to build up the inference rules from scratch based on the axioms and constructs of category theory, as applied to a specific category called the *syntactic category*, written as **Syn**. Category theory will serve two purposes: justification for why we choose certain inference rules, and since we can interpret our type theory as the category **Syn**, we can prove properties about it by appealing to results from category theory.

4 Categorical Motivations

4.1 Definition of a Category / Structural Rules

What exactly is a category? A *category* $\mathbf{C}$ consists of objects $A, B, C, \ldots$, and arrows $f, g, h, \ldots$, and two operations:

1. dom, which assigns to each arrow f an object in $\mathbf{C}$ (the domain of f).

2. cod, which assigns to each arrow f an object in $\mathbf{C}$ (the codomain of f).

We write an arrow f with $\text{dom}(f) = A$, $\text{cod}(f) = B$ as $A \xrightarrow{f} B$. So, the next step is to decide what the objects and arrows in the category **Syn** will be. We define the objects and arrows of **Syn** as follows:

Definition 4.1. *The category* **Syn** *consists of objects that are types which can be derived as a type existence judgement, with types that can be derived as equal by a type equality judgement identified as the same object. That is, if a judgement of the form "T Type" can be derived, then T is an object in* **Syn***, and if a judgement of the form "$T = U$" can be derived, then T and U name the same object.*

The arrows in **Syn** *are all the terms for which a judgement of the form $t : B\ [A]$ can be derived, with terms that can be derived as equal identified as the same arrow. If a judgement of the form $t_1 = t_2 : B\ [A]$ can be derived, then $t_1 : B\ [A]$ and $t_2 : B\ [A]$ are identified as the same arrow.*

The domain and codomain are given by:

$$\text{dom}(t : B\ [A]) = A$$
$$\text{cod}(t : B\ [A]) = B$$

As an example, $\mathbb{N}$ is an object in **Syn**, which is written formally in type theory as $\mathbb{N}$ Type. The term $+(x, 1)$ (where x is a free variable of type $\mathbb{N}$ and $+$ and 1 are appropriate constants) would be an arrow from $\mathbb{N}$ to $\mathbb{N}$.

Additionally, we require that the type equality relationship and the term equality relationship are equivalence relations. This yields the following six rules to guarantee this property:

$$\frac{A \text{ Type}}{A = A} \quad (1) \qquad \frac{A = B}{B = A} \quad (3) \qquad \frac{A = B \qquad B = C}{A = C} \quad (5)$$

$$\frac{f : B\ [A]}{f = f : B\ [A]} \quad (2) \qquad \frac{f = g : B\ [A]}{g = f : B\ [A]} \quad (4) \qquad \frac{f = g : B\ [A] \quad g = h : B\ [A]}{f = h : B\ [A]} \quad (6)$$

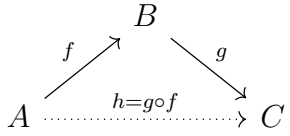
In an intuitive sense, an arrow behaves like a function. For example, the arrow $+(x, 1) : \mathbb{N}\ [\mathbb{N}]$ from $\mathbb{N}$ to $\mathbb{N}$ is like a function that takes a single input x of type $\mathbb{N}$ and returns a result of type $\mathbb{N}$. Formally, to evaluate a term f which contains the free variable x using the input term t, you replace all occurrences of x in f with t, which is notated as $f[t]$. However, the literal string $f[t]$ is not a term, it is only a meta-notation. As an example, if f is $+(x, 1) : \mathbb{N}\ [\mathbb{N}]$, and we wish to evaluate it with the input term $+(2, 3) : \mathbb{N}$, then we would get the result $f[+(2, 3)] \equiv +(+(2, 3), 1) : \mathbb{N}$.

There is also the case that an arrow contains no free variables x. For example, the term $5 : \mathbb{N}\ [\mathbb{N}]$. If we substitute anything into this term then we would simply get 5 back. Thus, an arrow with no free variables acts like a constant function. We will formalize these notions of substitution later.

Furthermore, a category has the following additional structure:

1. Identity arrows: for each object A there is an arrow $A \circlearrowright id_A$.

2. Composition: for any arrows $A \xrightarrow{f} B$ and $B \xrightarrow{g} C$ there exists an arrow $h = g \circ f$ from A to C. This is represented by the diagram:



We must equip our type theory and similarly our category **Syn** with each of these properties. We introduce the following definitions:

Definition 4.2. *The identity arrow on a type A is the term $x : A\ [A]$. This yields the inference rule:*

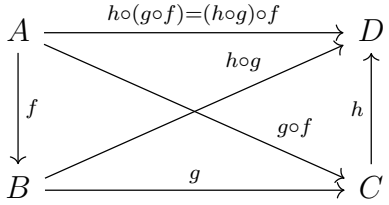
$$\frac{A\ \ Type}{x : A\ [A]} \tag{7}$$

For composition, we assume that there are arrows $f : B\ [A]$, and $g : C\ [B]$. We define the composition $g \circ f$ as the term $g[f] : C\ [A]$. As an inference rule:

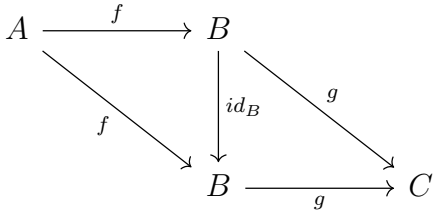
$$\frac{f : B\ [A] \qquad g : C\ [B]}{g[f] : C\ [A]} \tag{8}$$

Lastly, a category has two properties to constrain the behavior of arrows.:

1. Associativity: If we have the situation $A \xrightarrow{f} B \xrightarrow{g} C \xrightarrow{h} D$, then $h \circ (g \circ f) = (h \circ g) \circ f$. Pictorially the diagram below must commute[1]:



2. Unit Law: In the situation $A \xrightarrow{f} B \xrightarrow{g} C$, composition with the identity arrow id_B gives $id_B \circ f = f$ and $g \circ id_B = g$. In a commutative diagram:



We now interpret these properties as inference rules. For associativity we suppose there are arrows $f : B\ [A]$, $g : C\ [B]$, $h : D\ [C]$. We compose $g \circ f = g[f] : C\ [A]$ and $h \circ g = h[g] : D\ [B]$. We then compose again to obtain the two possible compositions:

$$h \circ (g \circ f) = h[g[f]] : D\ [A]$$
$$(h \circ g) \circ f = h[g][f] : D\ [A]$$

The associativity property states that these substitutions are equivalent. As an inference rule:

$$\frac{f : B\ [A] \qquad g : C\ [B] \qquad h : D\ [C]}{h[g[f]] = h[g][f] : D\ [A]} \tag{9}$$

[1]A diagram is *commutative* if for any vertices A and A' and for any two paths p_1 and p_2 from A to A', p_1 and p_2 form equal arrows by composition along their separate paths.

For the unit law, we suppose that if there is an arrow $f : B\ [A]$, then $id_B \circ f = f$, and similarly for composition with identity on the right side of $g : C\ [B]$. We write id_B as $x : B\ [B]$, and using this we get two inference rules:

$$\frac{f : B\ [A]}{x[f] = f : B\ [A]} \qquad (10) \qquad \frac{g : C\ [B]}{g[x] = g : C\ [B]}$$

The associativity and unit laws did not provide much new insight in how to judge the types of terms, like the composition property did. However, they instead provide simple but important properties about how substitution works. Rule (9) describes the associativity of substitution, while rule (10) provides a base case for substituting into a single free variable. The second inference rule does not yield anything particularly useful in this formulation of type theory, since we only allow for one free variable x.

4.2 A Terminal Object

We currently have no means to describe constants of a certain type. Suppose the type $\mathbb{N}$ exists in our type theory (it will later). Then we would want to introduce a constant 0 of type $\mathbb{N}$. We might want to write this as $0 : \mathbb{N}$, but to fit more naturally with the framework of category theory, we need this term to be an arrow, and since it has no context, it is not an arrow. Instead we want to write something of the form $0 : \mathbb{N}\ [A]$. Note that this function acts as a constant 0, since it is a constant function from A to $\mathbb{N}$. In effect we want to associate any *closed term* (a term with no free variables) of type $\mathbb{N}$ with arrows from A to $\mathbb{N}$.

The most natural way to do this is for the type A to be the *terminal object* 1 in the category **Syn**. We introduce this type using the rule:

$$\frac{}{1\ \text{Type}} \qquad (11)$$

For 1 to be a terminal object requires that for any type T there is a unique arrow from T to 1. To write this we require a constant of type 1, which is written $\overline{1}$. The existence and uniqueness of this arrow is given by the rules:

$$\frac{T\ \text{Type}}{\overline{1} : 1\ [T]} \qquad (12) \qquad \frac{T\ \text{Type} \qquad f : 1\ [T]}{\overline{1} = f : 1\ [T]} \qquad (13)$$

Now, suppose we want to write a closed term t of type T. We simply need to write it as the constant function $t : T\ [1]$. We can prove that this corresponds to a closed term using (13):

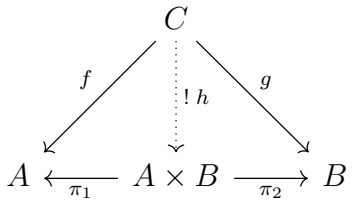
$$\frac{\dfrac{T\ \text{Type} \qquad f : 1\ [T]}{\overline{1} = f : 1\ [T]} \qquad t : T\ [1]}{t[\overline{1}] = t[f] : T\ [T]}$$

This means that our closed term t is the same regardless of what we substitute into it, which is precisely what it means to be a closed term. This justifies writing terms of the form $t : T\ [1]$ for closed terms in T.

4.3 Products

Our type theory so far is extremely limited. We have only introduced two types, $\mathbb{N}$ and 1. Currently it is impossible to create new types from these types. In addition, we can only consider arrows of a single variable, since the only free variable is x. For example, we simply have no way to express an arrow *and* which would take in two natural numbers and output their sum.

To address these limitations we introduce *product types*, which correspond to *products* in our category. Given objects A and B, the binary product $A \times B$ is an object, along with two arrows $A \times B \xrightarrow{\pi_1} A$ and $A \times B \xrightarrow{\pi_2} B$. These arrows satisfy a certain universal property: if there are arrows $C \xrightarrow{f} A$ and $C \xrightarrow{g} B$ then there exists a unique arrow $C \xrightarrow{h} A \times B$. In addition, $\pi_1 \circ h = f$ and $\pi_2 \circ h = g$. This is represented by the commutative diagram:



For the category **Syn** to have products, we need to provide a mechanism to construct the product type $A \times B$, π_1, π_2 and the unique arrow h. The product type $A \times B$ is given by the concrete syntax $Product(A, B)$, and the existence of the type $Product(A, B)$ is given by the following rule:

$$\frac{A \text{ Type} \qquad B \text{ Type}}{Product(A, B) \text{ Type}} \tag{14}$$

We can now specify the existence of π_1 and π_2, which use the concrete syntax $fst(x)$ and $snd(x)$ respectively. The existence of these arrows is given by the two rules:

$$\frac{}{fst(x) : A\ [Product(A, B)]} \tag{15}$$

$$\frac{}{snd(x) : B\ [Product(A, B)]} \tag{16}$$

The last existence we need to specify is the existence of the arrow h, if such arrows f and g exists. The concrete syntax for this is $pair(f, g)$. Formally, the rule is:

$$\frac{f : A\ [C] \qquad g : B\ [C]}{pair(f, g) : Product(A, B)\ [C]} \tag{17}$$

We have constructed the existence of all the aspects of product types. Now we must describe the uniqueness of $pair(f, g)$. Specifically, if there is another arrow h from C to $Product(A, B)$ satisfying the commutative diagram above, then it must be the same as $pair(f, g)$. This is given by the rule:

$$\frac{f : A\ [C] \quad g : B\ [C] \quad h : Product(A, B)\ [C] \quad fst(h) = f : A\ [C] \quad snd(h) = g : B\ [C]}{h = pair(f, g) : Product(A, B)\ [C]} \tag{18}$$

Lastly, we need to specify that $fst(x) \circ pair(f, g) = f$, and likewise for snd. This is given by the rules:

$$\frac{h = pair(f, g) : Product(A, B)\ [C]}{fst(h) = f : A\ [C]} \quad (19) \qquad \frac{h = pair(f, g) : Product(A, B)\ [C]}{snd(h) = g : A\ [C]} \quad (20)$$

4.4 An Initial Object

We wish to also equip our category with an *initial object*. An initial object is an object I such that for every object A in the category, there is a unique arrow from I to A. We will choose to call this object in the category **Syn** as *Empty*. We first have to state that it is a type:

$$\frac{}{Empty\ \text{Type}} \quad (21)$$

Now we need only satisfy that there exists an arrow between *Empty* and any other type, and that this arrow is unique. We define an arrow between *Empty* and a type A as:

$$\frac{A\ \text{Type}}{abort_A(x) : A\ [Empty]} \quad (22)$$

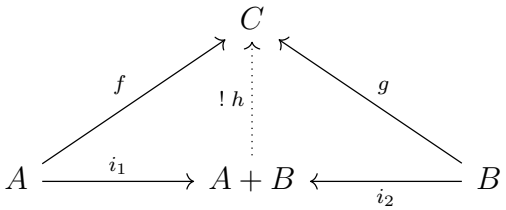
Proposition 4.1. *The Empty type as specified above is the initial object of the category* **Syn**.

Proof. Clearly, *Empty* is an object of **Syn**, and for any object A of **Syn** there exists an arrow from *Empty* to **Syn**, as given by (22). To show that the arrow is unique, consider that there are no constructors for the type *Empty*, that is, there is no possible way to construct an arrow from some type B to *Empty*. ■

4.5 Coproducts

We implement sum types similarly to product types: they correspond to coproducts in the category **Syn**.

Definition 4.3. *Let A and B be objects in a category. An object $A + B$ is called the coproduct of A and B if there exist arrows $A \xrightarrow{i_1} A + B$ and $B \xrightarrow{i_2} A + B$ such that for any object C and any arrows $A \xrightarrow{f} C$ and $B \xrightarrow{g} C$, there exists a unique arrow $A + B \xrightarrow{h} C$ such that $f = h \circ i_1$ and $g = h \circ i_2$. That is, the following diagram commutes:*



We add the necessary inference rules to the type theory so that **Syn** has coproducts. First, we need a new type $A + B$ for any two types A and B. We call this type $Sum(A, B)$. The rule of inference is given as:

$$\frac{A\ \text{Type} \qquad B\ \text{Type}}{Sum(A, B)\ \text{Type}} \quad (23)$$

Now we create the existence of the arrows i_1 and i_2. These are given by the terms $inl(x)$ and $inr(x)$ (include left/right). The existence is given by the rules:

$$\frac{}{inl(x) : Sum(A, B)\ [A]} \quad (24) \qquad \frac{}{inr(x) : Sum(A, B)\ [B]} \quad (25)$$

The last arrow to construct is the unique arrow h. Suppose f and g are arrows $A \xrightarrow{f} C$ and $B \xrightarrow{f} C$. Then we express the unique arrow h as $match(x, f, g)$. Intuitively, *match* can use the first parameter x to tell if it is the *inl* or *inr* case, and then appropriately apply either f or g. The existence of this arrow is given by the rule:

$$\frac{f : C\ [A] \qquad g : C\ [B]}{match(x, f, g) : C\ [Sum(A, B)]} \quad (26)$$

However, *match* must commute in the diagram above. Specifically, we must have that $match(x, f, g) \circ inl(x) = f$, and likewise for g and inr. This is given by the following two rules:

$$\frac{f : C\ [A]}{match(inl(x), f, g) = f : C\ [A]} \quad (27)$$

$$\frac{g : C\ [B]}{match(inr(x), f, g) = g : C\ [B]} \quad (28)$$

Note that this is **not** the typical substitution, since we do not substitute $inl(x)$ and $inr(x)$ into f and g, which might contain free variables. Essentially the free variables in f and g were bound under *match*, and we must keep track of that. This is thus a special case of the composition operation for arrows, in which substitution must behave differently.

Finally, we need uniqueness of *match*. Suppose there is some other arrow $h : C\ [Sum(A, B)]$ such that $h \circ inl(x) = f$ and $h \circ inr(x) = g$. Then, it must be that $h = match$. This is specified in the rule:

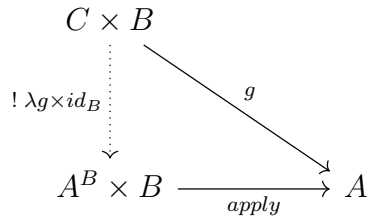
$$\frac{h : C\ [Sum(A, B)] \qquad h[inl(x)] = f : C\ [A] \qquad h[inr(x)] = g : C\ [B]}{match(x, f, g) = h : C\ [Sum(A, B)]} \quad (29)$$

4.6 Exponent Objects / Function Types

To make our type theory increasingly expressive, we add the notion of *function types*, which are types that represent functions themselves. We start with the following category theory definition, and then translate it for the category **Syn**.

Definition 4.4. *Let A and B be objects in a category that is equipped with binary products. Then the exponent object of A and B is an object A^B with a counit arrow $A^B \times B \xrightarrow{apply} A$ such that*

for any object C and any arrow $C \times B \xrightarrow{g} A$ there exists a unique arrow $C \xrightarrow{\lambda g} A^B$ such that this diagram commutes:



We now interpret this definition for the category **Syn**, and add the necessary inferences rules to the type theory so that **Syn** has exponent objects. First, we need to be able to construct exponent objects, which are known as function types in type theory. The rule is simple:

$$\frac{A \text{ Type} \qquad B \text{ Type}}{Func(B, A) \text{ Type}} \tag{30}$$

Now, A^B is an object which exists. We now define the existence of the counit arrow *apply*:

$$\frac{A \text{ Type} \qquad B \text{ Type}}{apply(x) : A\ [Product(Func(B, A), B)]} \tag{31}$$

We now need to complete the universal property. Suppose that C is any type, and let $g : A\ [Product(C, B)]$ be an arrow. Then there must exist a unique arrow from C to $Func(B, A)$, indexed by g. We write this arrow as $curry(g)$, and the existence of it is given by the rule:

$$\frac{g : A\ [Product(C, B)]}{curry(g) : Func(B, A)\ [C]} \tag{32}$$

We now need to express the commutativity of the diagram above. If we again suppose that $g : A\ [Product(C, B)]$ is an arrow, then we must have that:

$$apply(x) \circ pair(curry(g)[fst(x)], snd(x)) \equiv apply(pair(curry(g)[fst(x)], snd(x))) = g$$

Written as a rule of inference:

$$\frac{g : A\ [Product(C, B)]}{apply(pair(curry(g)[fst(x)], snd(x))) = g : A\ [Product(C, B)]} \tag{33}$$

Finally, we must guarantee the uniqueness of $curry(g)$. We suppose there is some h that makes the above diagram commute. Then this means that $h = curry(g)$. As a rule of inference:

$$\frac{\begin{array}{c} g : A\ [Product(C, B)] \\ h : Func(B, A)\ [C] \\ apply(pair(h[fst(x)], snd(x))) = g : A\ [Product(C, B)] \end{array}}{h = curry(g) : Func(B, A)\ [C]} \tag{34}$$

Proposition 4.2. *With the rules given in this section, the category* **Syn** *has an exponent object for any two given objects.*

A proof is not done explicitly, as it should be clear from the constructions above.

4.7 A Natural Numbers Object

With both product types and function types introduced we can now introduce a natural numbers object. To do so we claim the following:

Proposition 4.3. *The category* **Syn** *is a Cartesian closed category.*

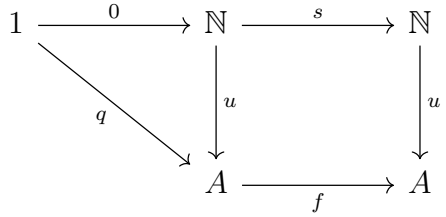
The proof of this is simple: the requirements of a Cartesian closed category are that the category has a terminal object, and for any objects A and B there exists the product $A \times B$ and the exponent object A^B. The terminal object in **Syn** is 1, the product $A \times B$ is given by $Product(A, B)$, and the exponent object A^B is given by $Func(B, A)$. We have shown previously that these three constructions satisfy the required properties. Thus, **Syn** matches the requirements of being a Cartesian closed category.

In a Cartesian closed category, and thus in **Syn**, a natural numbers object is defined as so:

Definition 4.5. *A natural numbers object in a Cartesian closed category is given by:*

- *An object* $\mathbb{N}$
- *An arrow* $1 \xrightarrow{0} \mathbb{N}$ *, where 1 is the terminal object*
- *An arrow* $\mathbb{N} \xrightarrow{s} \mathbb{N}$ *, called the successor*

such that for any diagram $1 \xrightarrow{q} A \xrightarrow{f} A$ *there is a unique arrow* $\mathbb{N} \xrightarrow{u} A$ *such that the following diagram commutes:*



We now build the necessary inference rules of our type theory to equip the category **Syn** with a natural numbers object. We first define $\mathbb{N}$ as a type:

$$\frac{}{\mathbb{N}\ \text{Type}} \tag{35}$$

We also introduce the arrow $1 \xrightarrow{0} \mathbb{N}$:

$$\frac{}{0 : \mathbb{N}\ [1]} \tag{36}$$

And we add the successor arrow from $\mathbb{N}$ to $\mathbb{N}$ to represent adding 1 to a natural number:

$$\frac{}{s(x) : \mathbb{N}\ [\mathbb{N}]} \tag{37}$$

Now, we need to satisfy the universal property of the definition. Suppose there are two arrows q and f like so: $1 \xrightarrow{q} A \xrightarrow{f} A$. That is, $q : A\ [1]$, $f : A\ [A]$. Then there must be a unique arrow $\mathbb{N} \xrightarrow{u} A$ indexed by q and f. We write this arrow as $natrec(x, q, f)$. We state the existence of it as:

$$\frac{q : A\ [1] \qquad f : A\ [A]}{natrec(x, q, f) : A\ [\mathbb{N}]} \tag{38}$$

In addition we need to specify that the diagram above commutes. Working with the left triangle, we have that $q = u \circ 0$. This gives the rule:

$$\frac{q : A\ [1]}{natrec(0, q, f) = q : A\ [1]} \tag{39}$$

Commutativity of the right square gives $u \circ s = f \circ u$. This gives the rule:

$$\frac{f : A\ [A]}{natrec(s(x), q, f) = f[natrec(x, q, f)] : A\ [\mathbb{N}]} \tag{40}$$

Finally, we must specify the uniqueness of the arrow *natrec*. That is, if there is another arrow $\mathbb{N} \xrightarrow{h} A$ which fits the universal property, then it must be the same as *natrec*. This is written in a rule of inference as:

$$\frac{q : A\ [1] \qquad f : A\ [A] \qquad h : A\ [\mathbb{N}] \qquad h[0] = q : A\ [1] \qquad h[s(x)] = f[h] : A\ [\mathbb{N}]}{natrec(x, q, f) = h : A\ [\mathbb{N}]} \tag{41}$$

5 List of All Inference Rules

As a formal reference, all of the inference rules of this type theory are enumerated below.

$$\frac{A \text{ Type}}{A = A}$$

$$\frac{f : B\ [A]}{f = f : B\ [A]}$$

$$\frac{A = B}{B = A}$$

$$\frac{f = g : B\ [A]}{g = f : B\ [A]}$$

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

$$\frac{f = g : B\ [A] \quad g = h : B\ [A]}{f = h : B\ [A]}$$

$$\frac{A\ \text{Type}}{x : A\ [A]}$$

$$\frac{f : B\ [A] \qquad g : C\ [B]}{g[f] : C\ [A]}$$

$$\frac{f : B\ [A] \qquad g : C\ [B] \qquad h : D\ [C]}{h[g[f]] = h[g][f] : D\ [A]}$$

$$\frac{}{1\ \text{Type}}$$

$$\frac{T\ \text{Type}}{\overline{1} : 1\ [T]}$$

$$\frac{T\ \text{Type} \qquad f : 1\ [T]}{\overline{1} = f : 1\ [T]}$$

$$\frac{A\ \text{Type} \qquad B\ \text{Type}}{Product(A, B)\ \text{Type}}$$

$$\frac{}{fst(x) : A\ [Product(A, B)]}$$

$$\frac{}{snd(x) : B\ [Product(A, B)]}$$

$$\frac{f : A\ [C] \qquad g : B\ [C]}{pair(f, g) : Product(A, B)\ [C]}$$

$$\frac{h = pair(f, g) : Product(A, B)\ [C]}{fst(h) = f : A\ [C]}$$

$$\frac{h = pair(f, g) : Product(A, B)\ [C]}{snd(h) = g : A\ [C]}$$

$$\frac{}{Empty\ \text{Type}}$$

$$\frac{A\ \text{Type}}{abort_A(x) : A\ [Empty]}$$

$$\frac{A\ \text{Type} \qquad B\ \text{Type}}{Sum(A, B)\ \text{Type}}$$

$$\frac{}{inl(x) : Sum(A, B)\ [A]}$$

$$\frac{}{inr(x) : Sum(A, B)\ [B]}$$

$$\frac{f : C\ [A] \qquad g : C\ [B]}{match(x, f, g) : C\ [Sum(A, B)]}$$

$$\frac{f : C\ [A]}{match(inl(x), f, g) = f : C\ [A]}$$

$$\frac{g : C\ [B]}{match(inr(x), f, g) = g : C\ [B]}$$

$$\frac{h : C\ [Sum(A, B)] \qquad h[inl(x)] = f : C\ [A] \qquad h[inr(x)] = g : C\ [B]}{match(x, f, g) = h : C\ [Sum(A, B)]}$$

$$\frac{A \text{ Type} \qquad B \text{ Type}}{Func(B, A) \text{ Type}}$$

$$\frac{A \text{ Type} \qquad B \text{ Type}}{apply(x) : A\ [Product(Func(B, A), B)]}$$

$$\frac{g : A\ [Product(C, B)]}{curry(g) : Func(B, A)\ [C]}$$

$$\frac{g : A\ [Product(C, B)]}{apply(pair(curry(g)[fst(x)], snd(x))) = g : A\ [Product(C, B)]}$$

$$\frac{}{\mathbb{N} \text{ Type}}$$

$$\frac{}{0 : \mathbb{N}\ [1]}$$

$$\frac{}{s(x) : \mathbb{N}\ [\mathbb{N}]}$$

$$\frac{q : A\ [1] \qquad f : A\ [A]}{natrec(x, q, f) : A\ [\mathbb{N}]}$$

$$\frac{q : A\ [1]}{natrec(0, q, f) = q : A\ [1]}$$

$$\frac{f : A\ [A]}{natrec(s(x), q, f) = f[natrec(x, q, f)] : A\ [\mathbb{N}]}$$

$$\frac{q : A\ [1] \qquad f : A\ [A] \qquad h : A\ [\mathbb{N}] \qquad h[0] = q : A\ [1] \qquad h[s(x)] = f[h] : A\ [\mathbb{N}]}{natrec(x, q, f) = h : A\ [\mathbb{N}]}$$