Simple Type Theory

Richard Southwell

November 21, 2020

1 Introduction

In this document we describe typed lambda calculus with sums, except that we explicitly keep track of contexts and all other parts of the theory using inference rules. This is the simple type theory described in [4]. This is a formalization of the type theory described in [1], but so that well formed contexts and types are generated following explicit inference rules, like in the appendix A2 of [2]. The terminology is mostly taken from [3].

2 Basics

A **term** is a value of a **type**. Some terms are **variables** (as we explain later). Each term t has a set $FV(t)$ of **free variables** (as we explain later).

There are six kinds of expressions:

1. A **typing declaration** $x : A$ says that x is a term of type A.
2. A **universal declaration** A_type says that A is a type.
3. An **equality declaration** $x \equiv y : A$ says values x and y of type A are equal.
4. A **context** Γ is a list, with each of its entries as a typing declaration or a universal declaration. We write $\Gamma :: \Delta$ to denote the concatenation of lists.
5. A **context declaration** $\Gamma \quad \mathrm{ctx}$ is a declaration that the context Γ is "well formed" (the meaning will be clear later from the rules).
6. A **judgment** is something of the form $\Gamma \vdash d$ where Γ is a context, and d is either a typing declaration or a universal declaration or an equality declaration. Sometimes we call d the **declaration** of the judgment $\Gamma \vdash d$.

A **rule** is something of the form

$$\frac{J_1 \quad J_2 \quad \dots \quad J_n}{K}$$

where $J_1, J_2, \ldots, J_n$ and K are all judgments. The meaning of the rule is that if each judgment in $J_1, J_2, \ldots, J_n$ can be derived in the type theory then judgment K may also be derived. Judgments can be stacked to make proof trees. An axiom is a rule

$$\frac{}{K}$$

with no prerequisites.

In addition to the assumed rules (which we name)

3 Forming base types

We write . to denote the empty context. The fact that the empty context is well formed is formalized by the rule:

$$\frac{}{.\quad \mathrm{ctx}}\ \text{ctx-EMP} \tag{1}$$

The next rule allows a well formed context to be extended by introducing a **base type** A**:**

$$\frac{\Gamma \quad \mathrm{ctx}}{\Gamma :: (A\_\mathrm{type}) \quad \mathrm{ctx}}\ \text{ctx-EXT1} \tag{2}$$

The base type A must not appear in the context Γ. Here we assume we have some list of base types [1]. If we are trying to model a particular system we may have specific base types ready, but for now let us just think of base types as variable types (although in this document we reserve the phrase "variable" for terms). So it is fine for A to be any type new to the context.

We can convert from well formed contexts to judgments about universal declarations using the following:

$$\frac{\Gamma :: (A\_\mathrm{type}) :: \Delta \quad \mathrm{ctx}}{\Gamma :: (A\_\mathrm{type}) :: \Delta \vdash A\_\mathrm{type}}\ \text{Vble1} \tag{3}$$

where Γ and Δ are contexts.

3.1 Example

Here is an example of how we derive the judgment $A\_\mathrm{type} \vdash A\_\mathrm{type}$.

$$\frac{\dfrac{\dfrac{}{.\quad \mathrm{ctx}}}{(A\_\mathrm{type}) \quad \mathrm{ctx}}}{(A\_\mathrm{type}) \vdash A\_\mathrm{type}} \tag{4}$$

Here we use ctx-EMP then ctx-EXT1 then Vble1.

4 Forming Other Types

This rule lets us form the unit type

$$\frac{\Gamma \quad \text{ctx}}{\Gamma \vdash 1\_\text{type}} \text{ Unit-Form} \tag{5}$$

Next, product types

$$\frac{\Gamma \vdash A\_\text{type} \quad \Gamma \vdash B\_\text{type}}{\Gamma \vdash A \times B\_\text{type}} \text{ Product-Form} \tag{6}$$

Next the empty type

$$\frac{\Gamma \quad \text{ctx}}{\Gamma \vdash 0\_\text{type}} \text{ Empty-Intro} \tag{7}$$

Next, sum types

$$\frac{\Gamma \vdash A\_\text{type} \quad \Gamma \vdash B\_\text{type}}{\Gamma \vdash A + B\_\text{type}} \text{ Sum-Form} \tag{8}$$

Next function types

$$\frac{\Gamma \vdash A\_\text{type} \quad \Gamma \vdash B\_\text{type}}{\Gamma \vdash A \to B\_\text{type}} \text{ Function-Form} \tag{9}$$

5 Forming Variables

We make a variable x of type A using the following rule

$$\frac{\Gamma \vdash A\_\text{type}}{\Gamma :: (x : A) \quad \text{ctx}} \text{ ctx-EXT2} \tag{10}$$

Note that x must be distinct from each term in the context Γ.

Note the ctx-EXT2 is the first time we have made a term (the variable x is a term of type A). The set of free variables of this newly made x is $FV(x) = \{x\}$.

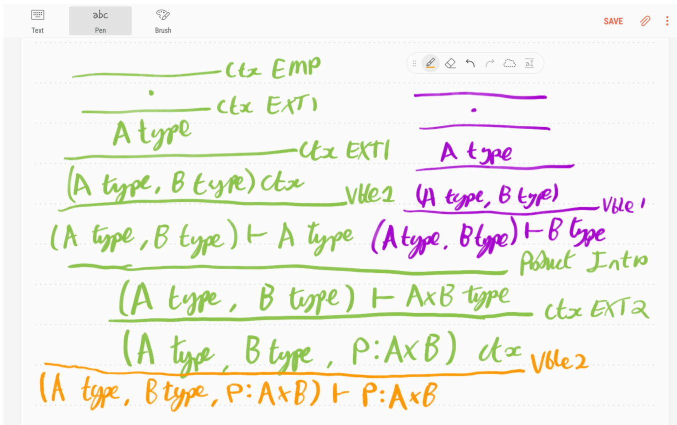
Judgments about variables can be formed with the following rule

$$\frac{\Gamma :: (x : A) :: \Delta \quad \text{ctx}}{\Gamma :: (x : A) :: \Delta \vdash x : A} \text{ Vble2} \tag{11}$$

5.1 Example

The following picture shows how we derive the rule

$$\frac{}{(A : \text{type}, B : \text{type}, p : A \times B) \vdash p : A \times B} \tag{12}$$



6 The Other Typing Rules

6.1 Products

We describe products as negative types [7].

$$\frac{\Gamma \quad \mathrm{ctx}}{\Gamma \vdash * : 1} \text{ Unit-Intro} \tag{13}$$

Here $*$ has the empty set $FV(*) = \{\}$ of free variables.

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash \langle a, b \rangle : A \times B} \text{ Product-Intro} \tag{14}$$

Here $FV(\langle a, b \rangle) = FV(a) \cup FV(b)$ where $\cup$ denotes the set theoretic union.

$$\frac{\Gamma \vdash p : A \times B}{\Gamma \vdash \mathrm{fst}(p) : A} \text{ Product-Elim1} \tag{15}$$

$FV(\mathrm{fst}(p)) = FV(p)$.

$$\frac{\Gamma \vdash p : A \times B}{\Gamma \vdash \mathrm{snd}(p) : B} \text{ Product-Elim2} \tag{16}$$

$FV(\mathrm{snd}(p)) = FV(p)$.

6.2 Sums

We describe sums as positive types.

The following elimination rule for the empty type is like that described in [4]

$$\frac{\Gamma \vdash A\_\mathrm{type} \quad \Gamma \vdash e : 0}{\Gamma \vdash \mathrm{abort}_A(e) : A} \text{ Empty-Elim} \tag{17}$$

$FV(\text{abort}_A(e)) = FV(e)$
Next, we have introduction rules for sum types

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash B\_\text{type}}{\Gamma \vdash \text{inl}_{A+B}(a) : A + B} \text{ Sum-Intro1} \tag{18}$$

$FV(\text{inl}_{A+B}(a)) = FV(a)$

$$\frac{\Gamma \vdash A\_\text{type} \quad \Gamma \vdash b : B}{\Gamma \vdash \text{inr}_{A+B}(b) : A + B} \text{ Sum-Intro2} \tag{19}$$

$FV(\text{inr}_{A+B}(b)) = FV(b)$

$$\frac{\Gamma \vdash s : A + B \quad \Gamma, x : A \vdash u : C \quad \Gamma, y : B \vdash v : C}{\Gamma \vdash \text{match}\,(s, x.u, y.v) : C} \text{ Sum-Elim} \tag{20}$$

$FV\,(\text{match}\,(s, x.u, y.v)) = FV(s) \cup FV(u) \cup FV(v) - [\{x\} \cup \{y\}]$ where $L - R$ denotes the set of members of set A that are not in set B.

Here $x.u$ denotes that variable x is bound to u in $\text{match}\,(s, x.u, y.v)$. And so $\text{match}\,(s, x.u, y.v)$ binds free occurrences of x and y.

6.3 Functions

We describe functions as negative types.

$$\frac{\Gamma :: (x : A) \vdash b : B}{\Gamma \vdash (\lambda x : A)\,.b : A \to B} \text{ Function-Intro} \tag{21}$$

$FV\,((\lambda x : A)\,.b) = FV(b) - \{x\}$.
Here the variable x is to bound in $(\lambda x : A)\,.b$.

$$\frac{\Gamma \vdash f : A \to B \quad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B} \text{ Function-Elim} \tag{22}$$

$FV(f(a)) = FV(f) \cup FV(a)$.

6.4 Extra Optional Rules

It is possible that we may wish to also include Wkg1 and Subst1 from page 554 of [2], but at the moment I don't think they are required.

7 Equational Theory

In this section we give the rules for making equality declarations

7.1 Equivalence Relation

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash a \equiv a : A} \text{ Reflexive} \tag{23}$$

$$\frac{\Gamma \vdash a \equiv a' : A}{\Gamma \vdash a' \equiv a : A} \text{ Symmetric} \tag{24}$$

$$\frac{\Gamma \vdash a \equiv a' : A \quad a' \equiv a'' : A}{\Gamma \vdash a \equiv a'' : A} \text{ Transitive} \tag{25}$$

7.2 Products

The uniqueness principle (η conversion rule) for the unit type is

$$\frac{\Gamma \vdash v : 1}{\Gamma \vdash v \equiv * : 1} \text{ Unit-Uniqueness} \tag{26}$$

The computation rules (β reduction rules) for the product type are

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash \mathrm{fst}\,(\langle a, b \rangle) \equiv a} \text{ Product-Computation1} \tag{27}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash \mathrm{snd}\,(\langle a, b \rangle) \equiv b} \text{ Product-Computation2} \tag{28}$$

The uniqueness principle (η conversion rule) for the product types is

$$\frac{\Gamma \vdash p : A \times B}{\Gamma \vdash p \equiv \langle \mathrm{fst}(p), \mathrm{snd}(p) \rangle : A \times B} \text{ Product-Uniqueness} \tag{29}$$

7.3 About Substitution

For a term u and a variable x and a term a of the same type as x we write $u\,[a/x]$ to denote the result of taking term u and replacing all free occurrences of x with term a. In such a case [5], if no free variables of a become bound in $u\,[a/x]$ then we say a is free for x in u.

7.4 Sums

The uniqueness principle (η conversion rule) of the empty type [6] is

$$\frac{\Gamma \vdash e : 0 \quad \Gamma \vdash A\_\mathrm{type} \quad \Gamma \vdash x : A}{\Gamma \vdash \mathrm{abort}_A(e) \equiv x : A} \text{ Empty-Uniqueness} \tag{30}$$

The computation rules (β reduction rules) for sum types are

$$\frac{\Gamma \vdash a : A \quad \Gamma :: (x : A) \vdash u : C \quad \Gamma :: (y : B) \vdash v : C}{\Gamma \vdash \mathrm{match}\,(\mathrm{inl}_{A+B}(a), x.u, y.v) \equiv u\,[a/x] : C} \text{ Sum-Computation1} \tag{31}$$

provided that a is free for x in u.

$$\frac{\Gamma \vdash b : B \quad \Gamma :: (x : A) \vdash u : C \quad \Gamma :: (y : B) \vdash v : C}{\Gamma \vdash \text{match}\left(\text{inr}_{A+B}(b), x.u, y.v\right) \equiv v\,[b/y] : C} \text{ Sum-Computation2} \quad (32)$$

provided that b is free for y in v.

(I am guessing these "freeness" conditions are required in each rule involving substitution, just like as in the internal language of a topos, described in [5]. Different variables can be used within the substitution if there are problems, as discussed in Bell's book on Local Set Theory).

The uniqueness principle (η conversion rule) for the sum types is

$$\frac{\Gamma \vdash s : A + B \quad \Gamma, h : A + B \vdash m : C}{\Gamma \vdash \text{match}\left(s, x.m\left[\text{inl}_{A+B}(x)/h\right], y.m\left[\text{inr}_{A+B}(y)/h\right]\right) \equiv m\,[s/h] : C} \text{ Sum-Uniqueness} \quad (33)$$

provided that $s, \text{inl}_{A+B}(x)$ and $\text{inr}_{A+B}(y)$ all each free for h in m.

7.5 Functions

The computation rule (β reduction rule) for function types is

$$\frac{\Gamma :: (x : A) \vdash m : C \quad \Gamma \vdash a : A}{\Gamma \vdash \left((\lambda x : A).m\right)(a) \equiv m\,[a/x] : C} \text{ Function-Computation} \quad (34)$$

provided that a is free for x in u.

The uniqueness rule (η conversion rule) for function types is

$$\frac{\Gamma \vdash f : A \to B}{\Gamma \vdash f \equiv (\lambda x : A).(f(x)) : A \to B} \text{ Function-Uniqueness} \quad (35)$$

Provided that x is not a free variable of f (that is, provided $x \notin FV(f)$).

We also require that the same function operating on equal inputs gives equal outputs, that is

$$\frac{\Gamma \vdash f : A \to B \quad \Gamma \vdash a \equiv a' : A}{\Gamma \vdash f(a) \equiv f(a') : B} \text{ Function-Similar Inputs} \quad (36)$$

We also require that operating equal functions on the same input give equal outputs, that is

$$\frac{\Gamma :: (x : A) \vdash b \equiv b' : B}{\Gamma \vdash (\lambda x : A).b \equiv (\lambda x : A).b' : A \to B} \text{ Function-Similar Functions} \quad (37)$$

Finally, if there is an α conversion from $(\lambda x : A).b$ to $(\lambda y : A).b'$ then we consider $(\lambda x : A).b \equiv (\lambda y : A).b'$, however α conversion is somewhat technical to implement, and I presume there is no need for α conversion if the variable names are chosen well. I guess the official rule for α conversion is

$$\frac{\Gamma :: (x : A) \vdash b : B \quad \Gamma :: (y : A) \vdash b' : B \quad \Gamma \vdash b\,[y/x] \equiv b' : B}{\Gamma \vdash (\lambda x : A)\,.b \equiv (\lambda y : A)\,.b' : A \to B} \tag{38}$$

provided that y is free for x in b. But I am not sure this rule properly defines alpha conversion (see [8]).

I guess the α conversion rules corresponding to the sum type are

$$\frac{\Gamma \vdash s : A + B \quad \Gamma :: (x : A) \vdash u : C \quad \Gamma :: (y : B) \vdash v : C \quad \Gamma :: (x' : A) \vdash u' : C \quad \Gamma \vdash u\,[x'/x] \equiv u' : C}{\Gamma \vdash \mathrm{match}\,(s, x.u, y.v) \equiv \mathrm{match}\,(s, x'.u', y.v) : C} \tag{39}$$

provided x' is free for x in u.

$$\frac{\Gamma \vdash s : A + B \quad \Gamma :: (x : A) \vdash u : C \quad \Gamma :: (y : B) \vdash v : C \quad \Gamma :: (y' : B) \vdash v' : C \quad \Gamma \vdash v\,[y'/y] \equiv v' : C}{\Gamma \vdash \mathrm{match}\,(s, x.u, y.v) \equiv \mathrm{match}\,(s, x.u, y'.v') : C} \tag{40}$$

provided y' is free for y in v.

8 Further Directions

1. It would be good to show that we can prove all the categorical machinery (terminal object, products, initial object, coproducts, exponential objects) can be produced, and that rules expressing their universal properties can be derived.

2. If we code up the above theory, we just seem to have to keep track of dependencies (like how $\langle a, b \rangle$ depends on a and b), and some label, and the lists of free and bound variables, and which type a term has. We could imagine all this as part of the data structure of the term, and then terms start to look a lot more like objects in a category of presheaves, or like algebras (W types). This suggests we could abstract this theory, and open up the possibility of categorical analysis of the meta theory.

3. We may want to add natural number types, like in [9].

4. We may want to add equalizers, subobject classifiers and reference to monomorphisms, so we can do topos theory.

5. The theory above allows us to model any bicartesian closed category [1]. We can model **Cat** by adding appropriate axioms, as described in [5].

References

[1] *Extensional Normalisation and Type-Directed Partial Evaluation for Typed Lambda Calculus with Sums* Vincent Balat, Roberto Di Cosmo, Marcelo Fiore

[2] Homotopy Type Theory Book

[3] ncatlab type theory page ncatlab type theory

[4] Video https://www.youtube.com/watch?v=KHteAK7GSRY

[5] *Elementary categories, elementary toposes* Colin McLarty

[6] stack exchange website math stack exchange

[7] ncatlab products product types

[8] FUNCTIONAL PEARLS alpha-conversion is easy THORSTEN ALTENKIRCH

[9] *Introduction to higher order categorical logic* Lambek and Scott