Simple Type Theory (Simplified)

Richard Southwell

November 25, 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 of typing declarations. We write $\Gamma :: \Delta$ to denote the concatenation (joining) of lists.
5. A **context declaration** $\Gamma \quad \text{ctx}$ 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:

$$\frac{}{. \quad \text{ctx}} \text{ Empty Context} \tag{1}$$

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

$$\frac{\Gamma \quad \text{ctx}}{\Gamma \vdash A\_\text{type}} \text{ Base Type Formation} \tag{2}$$

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.

4 Forming Other Types

This rule lets us form the unit type

$$\frac{\Gamma \quad \text{ctx}}{\Gamma \vdash 1\_\text{type}} \text{ Unit Formation} \tag{3}$$

Next, product types

$$\frac{\Gamma \vdash A\_\text{type} \quad \Gamma \vdash B\_\text{type}}{\Gamma \vdash A \times B\_\text{type}} \text{ Product Formation} \tag{4}$$

Next the empty type

$$\frac{\Gamma \quad \text{ctx}}{\Gamma \vdash 0\_\text{type}} \text{ Empty Formation} \tag{5}$$

Next, sum types

$$\frac{\Gamma \vdash A\_\text{type} \quad \Gamma \vdash B\_\text{type}}{\Gamma \vdash A + B\_\text{type}} \text{ Sum Formation} \tag{6}$$

Next function types

$$\frac{\Gamma \vdash A\_\text{type} \quad \Gamma \vdash B\_\text{type}}{\Gamma \vdash A \to B\_\text{type}} \text{ Function Formation} \tag{7}$$

5 Forming Variables

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

$$\frac{\Gamma \vdash A\_\text{type}}{\Gamma :: (x : A) \quad \text{ctx}} \text{ Context Extension} \tag{8}$$

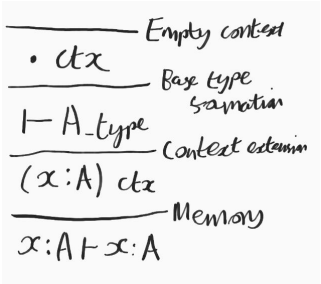
Here the variable x must not appear in the context Γ.

The set of free variables of this newly made x is $FV(x) = \{x\}$. Also $BV(x) = \{\}$ where $BV(t)$ denotes the set of bound variables of a term t, and $\{\}$ denotes the empty set.

Judgments about variables can be formed with the following rule

$$\frac{\Gamma :: (x : A) :: \Delta \quad \text{ctx}}{\Gamma :: (x : A) :: \Delta \vdash x : A} \text{ Memory} \tag{9}$$

The Memory rule appears as Vble1 on page 554 of [2]. The rule also appears in [1]. However [10] uses a different rule which they call the identity rule instead. However, we may derive said identity rule as follows:



In category theory Memory could be thought of as projection.

5.1 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], we say that a is free for x in u if and only if no free variables of a become bound in $u\,[a/x]$ (in other words, if the intersection of $FV(a)$ and $BV(u\,[a/x])$ is empty). Recall that the possible bindings that happen in our system just consist of three cases (1) in $(\lambda x : A).t$ we have that each variable x in term t binds to λ, and (2) in match $(s, x.u, y.v)$ we have that

each variable x in u becomes bound, and (3) we have that in match $(s, x.u, y.v)$ we we have that each variable y in v becomes bound.

To keep track of what is required within a computer, rather than just doing $(FV(m), FV(n)) \mapsto FV(m) \cup FV(n)$ etc. when terms are combined, we can form an expression tree for a term. We can also form corresponding expression trees for the free and bound variables, and when we construct $u\,[a/x]$ we replace the leaf x in u with the tree for a. Also, when we have binding operators, they 'color' the corresponding variables which might appear within the appropriate places deeper within the expression tree, and and we should make sure that after substitution, so we want to gaurd against previously free variables of the appropriate names, from a, being there , when we replace x with a in u so that the newly added subtree ends up lying within a colored region, involving new bindings of said variables.

Maybe we should use De-Brujin Notation. !!!

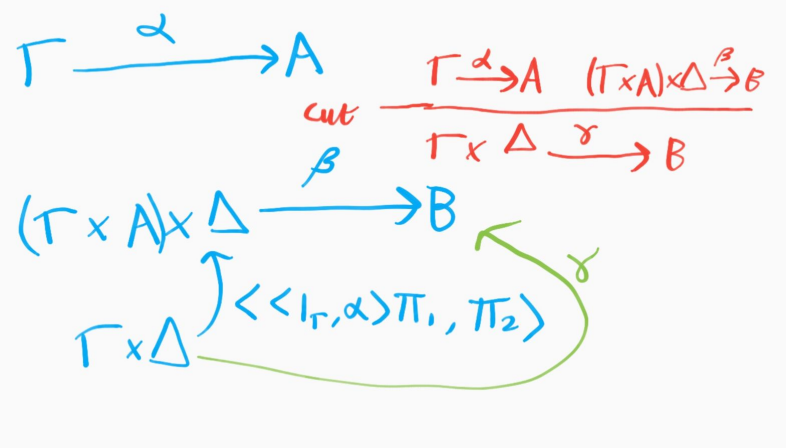
6 Other Structural Rules

Our next rules are taken from [10].

$$\frac{\Gamma \vdash a : A \quad \Gamma :: (x : A) :: \Delta \vdash b : B}{\Gamma :: \Delta \vdash b[a/x] : B}\ \text{Cut} \tag{10}$$

where x is free for a in b (is this required ? need to check about variables of chapter 1 of [10]).

The way the cut rule can be vizualized in terms of category theory as pictured below



Our next rule is weakening

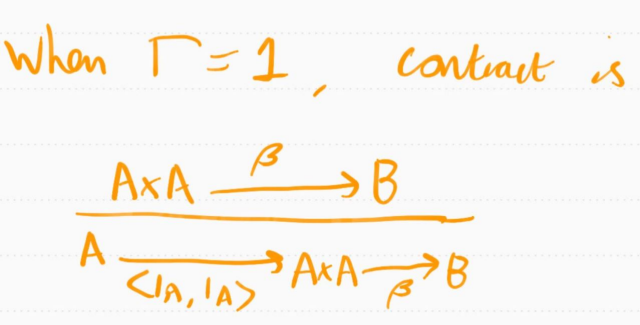
$$\frac{\Gamma :: \Delta \vdash b : B \quad \Gamma \vdash A\_\text{type}}{\Gamma :: (x : A) :: \Delta \vdash b : B}\ \text{Weakening} \tag{11}$$

In terms of category theory, this just corresponds to doing the product of the source arrow with some other arrow, and then doing a projection before

composing with the arrow. Note after the weakening rule has been applied, some may consider the variable x to occur within b invisably.

$$\frac{\Gamma :: (x : A, y : A) :: \Delta \vdash b : B}{\Gamma :: (x : A) :: \Delta \vdash b[x/y] : B} \text{ Contract} \tag{12}$$

The picture below should how to think of contract categorically (discounting Γ and Δ).



Our final structural rule is

$$\frac{\Gamma :: (x : A, y : A) :: \Delta \vdash b : B}{\Gamma :: (y : A, x : A) :: \Delta \vdash b : B} \text{ Exchange} \tag{13}$$

This allows us to think of the context as a set. Categorically exchange can be thought of as corresponding to permuting the order of objects in a product.

I wonder if cut and exchange are required.

7 The Other Typing Rules

We write $FV(t)$ to denote the set of free variables of a term t. We also write $BV(t)$ to denote the set of bound variables of a term t (the meaning of which will become clear later).

7.1 Products

We describe products as negative types [7].

$$\frac{\Gamma \quad \text{ctx}}{\Gamma \vdash * : 1} \text{ Unit-Intro} \tag{14}$$

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

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash \langle a, b \rangle : A \times B} \text{ Product-Intro} \tag{15}$$

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

$$\frac{\Gamma \vdash p : A \times B}{\Gamma \vdash \mathrm{fst}(p) : A} \text{ Product-Elim1} \tag{16}$$

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

$$\frac{\Gamma \vdash p : A \times B}{\Gamma \vdash \mathrm{snd}(p) : B} \text{ Product-Elim2} \tag{17}$$

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

7.2 Sums

We describe sums as positive types.

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

$$\frac{\Gamma \vdash A\_\mathrm{type} \quad \Gamma \vdash e : 0}{\Gamma \vdash \mathrm{abort}_A(e) : A} \text{ Empty-Elim} \tag{18}$$

$FV(\mathrm{abort}_A(e)) = FV(e)$ Also $BV(\mathrm{abort}_A(e)) = BV(e)$

Next, we have introduction rules for sum types

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash B\_\mathrm{type}}{\Gamma \vdash \mathrm{inl}_{A+B}(a) : A + B} \text{ Sum-Intro1} \tag{19}$$

$FV(\mathrm{inl}_{A+B}(a)) = FV(a)$ Also $BV(\mathrm{inl}_{A+B}(a)) = BV(a)$

$$\frac{\Gamma \vdash A\_\mathrm{type} \quad \Gamma \vdash b : B}{\Gamma \vdash \mathrm{inr}_{A+B}(b) : A + B} \text{ Sum-Intro2} \tag{20}$$

$FV(\mathrm{inr}_{A+B}(b)) = FV(b)$ Also $BV(\mathrm{inr}_{A+B}(b)) = BV(b)$

$$\frac{\Gamma \vdash s : A + B \quad \Gamma, x : A \vdash u : C \quad \Gamma, y : B \vdash v : C}{\Gamma \vdash \mathrm{match}\,(s, x.u, y.v) : C} \text{ Sum-Elim} \tag{21}$$

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

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

7.3 Functions

We describe functions as negative types.

$$\frac{\Gamma :: (x : A) \vdash b : B}{\Gamma \vdash (\lambda x : A).b : A \to B} \text{ Function-Intro} \tag{22}$$

$FV\left((\lambda x : A).b\right) = FV(b) - \{x\}$. Also $BV\left((\lambda x : A).b\right) = BV(b) \cup \{x\}$.

Here the variable x is to bound in $(\lambda x : A).b$.

$$\frac{\Gamma \vdash f : A \to B \quad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B} \text{ Function-Elim} \tag{23}$$

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

8 Equational Theory

In this section we give the rules for making equality declarations

8.1 Equivalence Relation

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

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

$$\frac{\Gamma \vdash a \equiv a' : A \quad a' \equiv a'' : A}{\Gamma \vdash a \equiv a'' : A} \text{ Transitive} \tag{26}$$

8.2 Products

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

$$\frac{\Gamma \vdash v : 1}{\Gamma \vdash v \equiv * : 1} \text{ Unit-Uniqueness} \tag{27}$$

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

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash \text{fst}\,(\langle a, b\rangle) \equiv a} \text{ Product-Computation1} \tag{28}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash \text{snd}\,(\langle a, b\rangle) \equiv b} \text{ Product-Computation2} \tag{29}$$

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

$$\frac{\Gamma \vdash p : A \times B}{\Gamma \vdash p \equiv \langle \text{fst}(p), \text{snd}(p)\rangle : A \times B} \text{ Product-Uniqueness} \tag{30}$$

8.3 Sums

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

$$\frac{\Gamma \vdash e : 0 \quad \Gamma \vdash A\_\text{type} \quad \Gamma \vdash x : A}{\Gamma \vdash \text{abort}_A(e) \equiv x : A} \text{ Empty-Uniqueness} \tag{31}$$

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

$$\frac{\Gamma \vdash a : A \quad \Gamma :: (x : A) \vdash u : C \quad \Gamma :: (y : B) \vdash v : C}{\Gamma \vdash \text{match}\,(\text{inl}_{A+B}(a), x.u, y.v) \equiv u\,[a/x] : C} \text{ Sum-Computation1} \tag{32}$$

provided that a is free for x in u.

$$\frac{\Gamma \vdash b : B \quad \Gamma :: (x : A) \vdash u : C \quad \Gamma :: (y : B) \vdash v : C}{\Gamma \vdash \text{match}\,(\text{inr}_{A+B}(b), x.u, y.v) \equiv v\,[b/y] : C} \text{ Sum-Computation2} \tag{33}$$

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

$$\frac{\Gamma \vdash s : A + B \quad \Gamma, h : A + B \vdash m : C}{\Gamma \vdash \text{match}\,(s, x.m\,[\text{inl}_{A+B}(x)/h]\,, y.m\,[\text{inr}_{A+B}(y)/h]) \equiv m\,[s/h] : C} \text{ Sum-Uniqueness} \tag{34}$$

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

8.4 Functions

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

$$\frac{\Gamma :: (x : A) \vdash m : C \quad \Gamma \vdash a : A}{\Gamma \vdash ((\lambda x : A)\,.m)\,(a) \equiv m\,[a/x] : C} \text{ Function-Computation} \tag{35}$$

provided that a is free for x in u.

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

$$\frac{\Gamma \vdash f : A \to B}{\Gamma \vdash f \equiv (\lambda x : A)\,.\,(f(x)) : A \to B} \text{ Function-Uniqueness} \tag{36}$$

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

$$\frac{\Gamma \vdash f : A \to B \quad \Gamma \vdash a \equiv a' : A}{\Gamma \vdash f(a) \equiv f(a') : B} \text{ Function-Similar Inputs} \tag{37}$$

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

$$\frac{\Gamma :: (x : A) \vdash b \equiv b' : B}{\Gamma \vdash (\lambda x : A) .b \equiv (\lambda x : A) .b' : A \to B} \text{ Function-Similar Functions} \tag{38}$$

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

$$\frac{\Gamma :: (x : A) \vdash b : B \quad \Gamma :: (y : A) \vdash b' : B \quad \Gamma \vdash b\,[y/x] \equiv b' : B}{\Gamma \vdash (\lambda x : A) .b \equiv (\lambda y : A) .b' : A \to B} \tag{39}$$

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

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

provided x' is free for x in u.

$$\frac{\Gamma \vdash s : A + B \quad \Gamma :: (x : A) \vdash u : C \quad \Gamma :: (y : B) \vdash v : C \quad \Gamma :: (y' : B) \vdash v' : C \quad \Gamma \vdash v\,[y'/y] \equiv v' : C}{\Gamma \vdash \text{match}\,(s, x.u, y.v) \equiv \text{match}\,(s, x.u, y'.v') : C} \tag{41}$$

provided y' is free for y in v.

9 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].

Do example of making an element of 1 times A
read how substitution is treated
prove exponential properties
Maybe we should use De-Brujin Notation. !!!

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

[10] Practical Foundations of Mathematics, Chapter 2, Paul Taylor