Notes from: Type Theory and Formal Proof, Chapter 2: Simply Typed Lambda Calculus, $\lambda\!\!\rightarrow$

Notes by Donald Pinckney

May 29, 2019

Untyped lambda calculus offers a concise and elegant formal system for expressing the behavior of functions. However, it is a bit "too liberal" with what it allows as terms, giving rise to some rather non-intuitive constructions. The goal is now to add types so as to prevent these strange constructions. In this chapter we add *simple types* which will actually be a bit too restrictive, and then in later chapters we will add more sophisticated types.

1 Simple Types

We define the set of *simple types* $\mathbb{T}$ by a grammar. Let $\mathbb{V} = \{\alpha, \beta, \gamma, \cdots\}$ be an infinite set of *type variables*. Note that from here on *variable* refers to ordinary variables $x, y, z, \cdots$. while *type variable* refers to $\mathbb{V}$.

Definition 2.2.1 (Set $\mathbb{T}$ of simple types)

1. (Type variable) If $\alpha \in \mathbb{V}$ then $\alpha \in \mathbb{T}$.
2. (Arrow type) If $\sigma, \tau \in \mathbb{T}$, then $(\sigma \to \tau) \in \mathbb{T}$.

As an abstract syntax, $\mathbb{T} = \mathbb{V} \mid \mathbb{T} \to \mathbb{T}$.

Notation 2.2.2

1. *We use $\alpha, \beta, \cdots$ for type variables.*
2. *We use $\sigma, \tau, \cdots$ (occasionally $A, B, \cdots$) for arbitrary simple types.*
3. *Arrow types are right associative.*

Examples include $\gamma, \beta \to \gamma, (\gamma \to \alpha) \to \alpha \to \beta \to \gamma$, etc. The intended intuitive meaning of simple types is that type variables are abstract representations of basic types such as nat, bool, etc. Arrow types represent function types.

2 Derivation Rules

We now want to decorate our λ-terms with types. We revise the syntax of λ-terms by indicating explicitly the types of binding variables:

Definition 2.4.1 (Pre-typed λ-terms, $\Lambda_{\mathbb{T}}$)

$$\Lambda_{\mathbb{T}} = V \mid (\Lambda_{\mathbb{T}} \Lambda_{\mathbb{T}}) \mid (\lambda V : \mathbb{T}.\Lambda_{\mathbb{T}})$$

To express things like "λ-term M has type σ" in a context Γ, we define a *judgment*:

Definition 2.4.2 (Statement, declaration, context, judgment)

1. A *statement* is of the form $M : \sigma$, where $M \in \Lambda_{\mathbb{T}}$ and $\sigma \in \mathbb{T}$. In such a statement M is called the *subject* and σ is called the *type*.

2. A *declaration* is a statement with subject being some *variable*.

3. A *context* is a list of declarations with *different subjects each*.

4. A *judgment* has the form $\Gamma \vdash M : \sigma$, with Γ a context and $M : \sigma$ a statement. We say that such a judgment is *derivable* if M has type σ in context Γ.

Note that we consider the variables declared in a context to be binding variables. Next we have the derivation rules for $\lambda\!\!\rightarrow$:

Definition 2.4.5 (Derivation rules for $\lambda\!\!\rightarrow$)

1. (var) $\Gamma \vdash x : \sigma$ if $x : \sigma \in \Gamma$
2. (appl) $\dfrac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}$
3. (abst) $\dfrac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x : \sigma . M : \sigma \to \tau}$

Remark *There is a correspondence between (appl) and (abst) rules defined above and implication elimination (modus ponens) and introduction in logic. See book, page 43 for more details.*

Definition 2.4.10 (Legal $\lambda\!\!\rightarrow$ terms) A pre-typed term M in $\lambda\!\!\rightarrow$ is *legal* if there exist context Γ and type ρ such that $\Gamma \vdash M : \rho$.

Examples The term xx is not legal. In order for xx to have a type, it must be through the application rule, so it must be that $x : \sigma \to \tau$ for some σ and τ (via the left x). However, in order to use the application rule we must also have that $x : \sigma$ (via the right x). Therefore, in order to have $\Gamma \vdash xx : \rho$ we must have that $x : \sigma \in \Gamma$ and $x : \sigma \to \tau \in \Gamma$. But this violates the definition of a context.

3 Flag Format of Proofs

A standard way of writing proofs inside a derivation system such as Definition 2.4.5 is in a tree-style. However, this quickly becomes unwieldy with larger proofs. Instead, we use the *flag format* of derivations. One displays each *declaration* (i.e. element of the context) in a "flag", with the "flag pole" indicating for what portion of the proof the declaration is in the context. In some situations we also omit uses of the (var) rule from the written proof, though they are still part of the proof.

Examples Consider the following tree-style proof:

$$\dfrac{\dfrac{\dfrac{y : \alpha \to \beta, z : \alpha \vdash y : \alpha \to \beta \qquad y : \alpha \to \beta, z : \alpha \vdash z : \alpha}{y : \alpha \to \beta, z : \alpha \vdash yz : \beta}}{y : \alpha \to \beta \vdash \lambda z : \alpha . yz : \alpha \to \beta}}{\varnothing \vdash \lambda y : \alpha \to \beta . \lambda z : \alpha . yz : (\alpha \to \beta) \to \alpha \to \beta}$$

If we rewrite this proof in flag-style and including uses of (var) rule, we get:

(a) $\boxed{y : \alpha \to \beta}$

(b) $\quad\boxed{z : \alpha}$

(1) $\quad y : \alpha \to \beta$ (var) on (a)

(2)	$z : \alpha$	(var) on (b)
(3)	$yz : \beta$	(appl) on (1) and (2)
(4)	$\lambda z : \alpha.yz : \alpha \to \beta$	(abst) on (3)
(5)	$\lambda y : \alpha \to \beta.\lambda z : \alpha.yz : (\alpha \to \beta) \to \alpha \to \beta$	(abst) on (4)

If we choose to exclude uses of (var) we would write:

(a)	$\boxed{y : \alpha \to \beta}$	
(b)	$\boxed{z : \alpha}$	
$(1')$	$yz : \beta$	(appl) on (a) and (b)
$(2')$	$\lambda z : \alpha.yz : \alpha \to \beta$	(abst) on $(1')$
$(3')$	$\lambda y : \alpha \to \beta.\lambda z : \alpha.yz : (\alpha \to \beta) \to \alpha \to \beta$	(abst) on $(2')$

4 Kinds of problems in type theory

In general there are 3 kinds of problems to be solved in type theory:

1. *Well-typedness / typeability.*

   In this problem, we are given a term M and wish to find an appropriate context and type:

   $$? \vdash M : ?$$

   In other words, show that M is legal. If the term is not legal, then explain where it goes wrong. A variant of this is *type assignment* where a context Γ is given, and want to find an appropriate type: $\Gamma \vdash M : ?$.

2. *Type Checking.*

   Here, the context Γ, term M, and type σ are all given to us, but we want to check (verify) if $\Gamma \stackrel{?}{\vdash} M : \sigma$.

3. *Term Finding / Term Construction / Inhabitation.*

   In this situation a context Γ and type σ are given, and we want to determine if there exists a term of that type, given the context: $\Gamma \vdash ? : \sigma$. A common specific case is for $\Gamma = \varnothing$.

I recommend looking at sections 2.7, 2.8 and 2.9 in the textbook for guided examples of all 3 of these problems.

5 General properties of $\lambda\!\to$