Types and Lambda Calculus Notes

by Tyler Wright

github.com/Fluxanoia fluxanoia.co.uk

These notes are not necessarily correct, consistent, representative of the course as it stands today, or rigorous. Any result of the above is not the author's fault.

These notes are in progress.

Contents

1 The Axioms of Lambda Calculus

We suppose that we have a countably infinite set of variables $\mathbb{V}$ (we usually refer to elements of this set as x, y, z, etc.), from this we define the alphabet of lambda calculus $\mathbb{V} + \{\lambda, ., (,)\}$.

The set of terms of lambda calculus Λ is defined inductively by the axioms. A string is a term if and only if there is a proof tree using the following axioms that concludes it is a term.

1.1 Axiom of Variables

For some x in $\mathbb{V}$, we have the axiom:

$$\frac{}{x \in \Lambda},$$

which says that each variable is a term.

1.2 Axiom of Application

We have the axiom:

$$\frac{M \in \Lambda \qquad N \in \Lambda}{(MN) \in \Lambda},$$

which says that for some terms M and N, (MN) is also a term.

1.3 Axiom of Abstraction

For some x in $\mathbb{V}$, we have the axiom:

$$\frac{M \in \Lambda}{(\lambda x.M) \in \Lambda},$$

which says that for some term M, $(\lambda x.M)$ is also a term.

1.4 Subterms

Subterms of a term M are substrings of M that are themselves terms and not captured by a λ (directly preceeded by).

1.5 Syntactical Conventions

Parentheses allow our lambda calculus to be unambigious, but for the sake of simplicity, we will construct conventions that will allow us to retain unique meaning with less parentheses:

- Omit outermost parentheses,
- Terms associate to the left, (MNP) parses as $((MN)P)$,
- Bodies of abstractions end at parentheses, $(\lambda x.MN)$ parses as $(\lambda x.(MN))$,
- Group repeated abstractions, $(\lambda xy.M)$ parses as $(\lambda x.(\lambda y.M))$.

2 Alpha

2.1 Free Variables

We define the function $FV : \Lambda \to \mathcal{P}(\mathbb{V})$, which returns the set of variables contained within a term M that are not bound. We define it recursively on the structure of terms:

$$\begin{aligned} FV(x) &= \{x\}, \\ FV(MN) &= FV(M) \cup FV(N), \\ FV(\lambda x.M) &= FV(M) \backslash \{x\}. \end{aligned}$$

If a term has no free variables we say it is closed, and if a term has at least one free variable then we say it is open. The set of all closed terms is denoted by Λ^0.

2.2 Substitution

We define 'capture-avoiding' substitution of a term M for a variable x recursively on the structure of terms:

$$\begin{array}{rcll} y[M/x] & = & y & \text{if } y \neq x, \\ y[M/x] & = & M & \text{if } y = x, \\ (PQ)[M/x] & = & P[M/x] \cup Q[M/x], & \\ (\lambda y.P)[M/x] & = & \lambda y.P & \text{if } y = x, \\ (\lambda y.P)[M/x] & = & \lambda y.P[M/x] & \text{if } y \neq x \text{ and } y \notin FV(M). \end{array}$$

On the final case, we stipulate that y cannot be a free variable of M because otherwise free variables in the substitution could be captured by the lambda.

2.3 Alpha Equivalence

Suppose we have a term $\lambda x.M$ and y in $\mathbb{V} \setminus FV(M)$, then substituting y for x is a change of bound variable name. If two terms can be made identical through changes of bound variable name, they are α-equivalent. The set of λ-terms is the set Λ under α-equivalence.

This equivalence is much more useful to us than string comparision, so for the remainder of the notes we will always be referring to λ-terms as terms.

2.4 The Variable Convention

For $M_1, \ldots, M_k$ terms occuring in the same scope, we assume each term has distinct bound variables. We can make this assumption as otherwise, we can use changes of bound variable names to make it so.

3 Beta

A term of the form $(\lambda x.M)N$ is called a β-redex and $M[N/x]$ is the contraction of the redex.

3.1 Beta Reductions

The one-step β-reduction relation, denoted by $\to_\beta$, is inductively defined by the redex rule:

$$\frac{}{(\lambda x.M)N \to_\beta M[N/x]},$$

the left and right application rules:

$$\frac{M \to_\beta M'}{MN \to_\beta M'N} \qquad \frac{N \to_\beta N'}{MN \to_\beta MN'},$$

and the abstraction rule:

$$\frac{M \to_\beta N}{\lambda x.M \to_\beta \lambda x.N}.$$

A term M is said to be in β-normal form if there is no term N such that $M \to_\beta N$.

In general, β-reductions are sequences of consecutive one-step β-reductions:

$$M_0 \to_\beta M_1 \to_\beta \cdots \to_\beta M_k,$$

for some k in $\mathbb{N}_0$. We say that M_0 β-reduces to M_k, denoted by $M_0 \twoheadrightarrow_\beta M_k$.

If $M \twoheadrightarrow_\beta N$, we say that N is a reduct of M, and is a proper reduct if $N \neq M$. If we can choose some β-normal N such that $M \twoheadrightarrow_\beta N$ then M is normalisable. If a term admits no infinite β-reductions then we say that it is strongly normalisable.

3.2 Standard Combinators

We have some interesting programs described below:

$$\begin{array}{rcl} \mathbf{I} & = & \lambda x.x \\ \mathbf{K} & = & \lambda xy.x \\ \mathbf{S} & = & \lambda xyz.xz(yz) \\ \omega & = & \lambda x.xx \\ \Omega & = & \omega\omega \\ \Theta & = & (\lambda xy.y(xxy))(\lambda xy.y(xxy)) \end{array} \qquad \begin{array}{c} \mathbf{I}M \twoheadrightarrow_\beta M \\ \mathbf{K}MN \twoheadrightarrow_\beta M \\ \mathbf{S}MNP \twoheadrightarrow_\beta MP(NP) \\ \omega M \twoheadrightarrow_\beta MM \\ \Omega \to_\beta \Omega \\ \Theta M \twoheadrightarrow_\beta M(\Theta M) \end{array}$$

3.3 Confluence of Beta

For a term M, if $M \twoheadrightarrow_\beta P$ and $M \twoheadrightarrow_\beta Q$ then there exists some term N such that $P \twoheadrightarrow_\beta N$ and $Q \twoheadrightarrow_\beta N$.

3.4 Beta Convertibility

For M, N terms, if M and N have a common reduct then we say that M and N are β-convertible, denoted by $M =_\beta N$.

4 Definability

4.1 Church Numerals

The Church numeral for the number n in $\mathbb{N}_0$, denoted by $\ulcorner n \urcorner$, is:

$$\lambda f x.\, \underbrace{f(\cdots(f\, x)\cdots)}_{n \text{ times}}.$$

Each Church numeral is already in normal form.

4.2 Lambda Definability

A function $f : \mathbb{N} \times \cdots \times \mathbb{N} \to \mathbb{N}$ is λ-definable if there exists a λ-term F that satifies:

$$F \ulcorner n_1 \urcorner \cdots \ulcorner n_k \urcorner =_\beta \ulcorner f(n_1, \ldots, n_k) \urcorner.$$

4.3 Basic Functions

We have addition, predecessor, subtraction, and the zero conditional defined here:

$$\begin{array}{ccll}
\textbf{Add} & = & \lambda yz.\lambda fx.yf(zfx) & \textbf{Add}\ulcorner m \urcorner\ulcorner n \urcorner =_\beta \ulcorner m+n \urcorner \\
\textbf{Pred} & = & \lambda z.\lambda fx.z(\lambda gh.h(gf))(\lambda u.x)(\lambda u.u) & \begin{array}{l}\textbf{Pred}\ulcorner 0 \urcorner =_\beta \ulcorner 0 \urcorner \\ \textbf{Pred}\ulcorner n+1 \urcorner =_\beta \ulcorner n \urcorner\end{array} \\
\textbf{Sub} & = & \lambda mn.n\,\textbf{Pred}\,m & \begin{array}{ll}\textbf{Sub}\ulcorner m \urcorner\ulcorner n \urcorner =_\beta \ulcorner 0 \urcorner & \text{if } m-n<0 \\ \textbf{Sub}\ulcorner m \urcorner\ulcorner n \urcorner =_\beta \ulcorner m-n \urcorner & \text{otherwise}\end{array} \\
\textbf{IfZero} & = & \lambda\lambda xyz.x(\mathbf{K}\,z)y & \begin{array}{l}\textbf{IfZero}\ulcorner 0 \urcorner\ulcorner p \urcorner\ulcorner q \urcorner =_\beta \ulcorner p \urcorner \\ \textbf{IfZero}\ulcorner n+1 \urcorner\ulcorner p \urcorner\ulcorner q \urcorner =_\beta \ulcorner q \urcorner\end{array}
\end{array}$$

4.4 Church Lists

The Church encoding of a list xs is the term $\ulcorner xs \urcorner$ defined recursively by:

$$\begin{aligned}
\ulcorner \textbf{Nil} \urcorner &= \lambda cn.n, \\
\ulcorner x : xs \urcorner &= \lambda cn.c\ulcorner x \urcorner(\ulcorner xs \urcorner\, cn).
\end{aligned}$$

This gives us the definition of **Cons**:

$$\textbf{Cons} = \lambda xy.\lambda cn.cx(ycn).$$

5 Recursion

5.1 The Factorial

We have that fact is the unique function on the natural numbers that satistfies the equation:

$$F(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot F(n-1) & \text{otherwise.} \end{cases}$$

5.2 Fixed Points

We say that a term N is a fixed point of another term M if $MN =_\beta N$.

5.3 The First Recursion Theorem

Every term posesses a fixed point.

Proof. Let M be a term. We define:

$$\mathbf{Y} = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx)),$$

called Curry's Paradoxical Combinator. We will show that $M(\mathbf{Y}M) =_\beta \mathbf{Y}M$ (so $\mathbf{Y}M$ is a fixed point of M):

$$\begin{aligned} \mathbf{Y}M &\to_\beta (\lambda x.M(xx))(\lambda x.M(xx)) \\ &\to_\beta M((\lambda x.M(xx))(\lambda x.M(xx))) \end{aligned}$$

$$M(\mathbf{Y}M) \to_\beta M((\lambda x.M(xx))(\lambda x.M(xx)))$$

$$M(\mathbf{Y}M) =_\beta \mathbf{Y}M,$$

as required. □

5.3.1 Solving Recursive Definitions

We want to solve for some term M:

$$\begin{aligned} &Mx_1 \cdots x_n =_\beta N[M/y] \\ &\iff M =_\beta \lambda x_1 \cdots x_n.N[M/y] \\ &\iff M =_\beta (\lambda y.\lambda x_1 \cdots x_n.N)M \end{aligned}$$

so if we find a fixed point of $(\lambda y.\lambda x_1 \cdots x_n.N)$ then we have an M which satisfies the equation. But, by the first recursion theorem, we can always find such M. Thus, we can just set $M = \mathbf{Y}(\lambda y.\lambda x_1 \cdots x_n.N)$.

6 Induction

We let Φ be some property of terms. We have that if the following conditions are met then it follows that Φ holds for all terms:

1. For all variables x, $\Phi(x)$ holds,
2. For all terms P, Q, if $\Phi(P)$ and $\Phi(Q)$ hold then $\Phi(PQ)$ holds,
3. For all terms P and variables x, if $\Phi(P)$ holds then $\Phi(\lambda x.P)$ holds.

6.1 The Induction Metaprinciple

For some set S with an inductive definition defined by rules $R_1, \dots, R_k$. The induction principle for proving that for all s in S, $\Phi(s)$ holds has k clauses corresponding to the rules of S.

If a rule R_i has m premises and a side condition ψ:

$$\psi \, \frac{s_1 \in S \cdots s_m \in S}{s \in S}, \, (R_i)$$

then the corresponding clause in the induction principle requires that if $\Phi(s_1), \dots, \Phi(s_m), \Phi(\psi)$ hold then $\Phi(s)$ holds.

7 Types

7.1 Monotypes

We assume a countable set of type variables $\mathbb{A}$ (usually denoted by a, b, c, etc.). The monotypes $\mathbb{T}$ are a set of strings defined inductively by the type variable rule:

$$\frac{}{a \in \mathbb{T}},$$

for some $a \in \mathbb{A}$ and the arrow rule:

$$\frac{A \in T \qquad B \in T}{(A \to B) \in \mathbb{T}}.$$

7.2 Type Schemes

Types schemes are pairs consisting of a finite set of type variables $a_1, \ldots, a_m$ and a monotype A which we write as $\forall\, a_1 \cdots a_m.A$.

7.3 Free Type Variables

We define the set of free type variables for a type scheme $\forall\, \bar{a}.A$, denoted by $FTV(\forall\, \bar{a}.A)$, recursively by the following rules:

$$\begin{aligned} FTV(a) &= \{a\}, \\ FTV(A \to B) &= FTV(A) \cup FTV(B), \\ FTV(\forall\, a_1 \cdots a_m.A) &= FTV(A) \backslash \{a_1, \ldots, a_m\}. \end{aligned}$$

We consider type schemes that only differ by choice of bound variable names to be equivalent.

7.4 Type Substitution

A type substitution is a total map σ, τ, θ from $\mathbb{A}$ to $\mathbb{T}$ with the property that $\sigma(a) \neq a$ for only finitely many $a \in \mathbb{A}$. We define the map as follows:

$$\begin{aligned} a\sigma &= \sigma(a) \\ (A \to B)\sigma &= A\sigma \to B\sigma. \end{aligned}$$

7.4.1 Composition of Substitutions

We write $\sigma_1\sigma_2$ for the substitution obtained by composing σ_2 after σ_1, defined as by $(\sigma_1\sigma_2)(a) = (\sigma_1(a))\sigma_2$.