# **λ-Calculus Cheat Sheet**

Overview of the Untyped Lambda Calculus

#### http://thesyntacticsugar.blogspot.com

# Definitions

- V is the set of variables
- Λ is the set of lambda terms

The  $\lambda$ -terms are defined as:

 $x \in V \Rightarrow x \in \Lambda$ 

If x is a variable, then x is a lambda term.

 $M, N \in \Lambda \Rightarrow (M N) \in \Lambda$ 

If M and N are lambda terms, then (M N) is a lambda term, called an application. The function M is applied to the input N.

 $M \in \Lambda, x \in V \Rightarrow (\lambda x.M) \in \Lambda$ 

If M is a lambda term, and x is a variable, then  $\lambda x.M$  is a lambda term; an abstraction of the anonymous function  $x \rightarrow M$ . A lambda abstraction binds the free variable x in M.

To summarize, a  $\lambda$ -term can be:

- a variable, e.g., x;
- an **application**, e.g., M N; or
- a function **abstraction**, e.g., λx.M.

Nothing else is a  $\lambda$ -term.



Alonzo Church (1903-1995)

# Specification in Backus-Naur Form

<term> ::= <variable> | ( <term> )  $|(\lambda < variable > . < term >)$ 

### **Syntax**

· Application associates to the left:

M P Q means ((M P) Q).

• λ-abstractions extend as far to the right as possible:

 $\lambda x.M P Q$  means  $\lambda x.(M P Q)$ .

### Shorthand notation

- $\lambda xyz.x$  is shorthand for  $\lambda x.\lambda y.\lambda z.x$ .
- · Outermost parentheses are usually omitted.
- λx(M) is λx.M.

# Free variables

1

S

The free variables (FV) of a  $\lambda$ -term is defined, inductively, as:

1. FV(v) = {v} 2. FV(M N) = FV(M) U FV(N)3.  $FV(\lambda x.M) = FV(M) - \{x\}$ 

M is a closed  $\lambda$ -term (also known as *combinator*) if FV(M) =  $\emptyset$ . The set of closed  $\lambda$ -terms is denoted by  $\Lambda^{\circ}$ .

Standard combinators	Y-Combinator
$I \equiv \lambda x. x$ $K \equiv \lambda x y. x$	The combinator <b>Y</b> provides a way to represent recursion:
$\mathbf{K}_{*} \equiv \lambda xy.y$ $\mathbf{S} \equiv \lambda xyz.xz(yz)$	$\mathbf{Y} \equiv \lambda f_{\cdot}(\lambda x.f(xx))(\lambda x.f(xx))$

#### Alpha-convertibility

Terms that differ only by alpha-conversion (renaming of bound variables) are called *α-equivalent*:

 $\lambda z.z =_{\alpha} \lambda x.x$ 

#### **Beta-reduction**

β-reduction formalizes the notion of a computation in the rule.

 $(\lambda x.M)N \rightarrow_{\beta} M[x := N]$ 

where x := N denotes the substitution of any free occurrence of x in M with the value N.

 $M \rightarrow N$  ... M reduces to N in exactly one step.

M —» N ... M reduces to N in zero or more steps.

#### Eta-convertibility

n-conversion expresses the idea of extensionality:

 $(\lambda x.M)x =_n M$ if x ∉ FV(M)

# Normal Form

A lambda expression is in normal form if no subexpression can be reduced. A term is said to have a normal form if it can be reduced to a term in normal form.

### **Head Normal Form**

An expression is in head normal form if the outermost application cannot be reduced, i.e., if there is no betaredex in head position.

Boolean logic			A
True	т	≡ λxy.x	
False	F	$\equiv \lambda xy.y$ (equal to numeral 0)	For a
And		≡ λρα ραρ	
Or		= λρα.ρρα	Chi
Not		≡ λpab.pba	Cill
lf-then-else		≡ λpab.pab	<b>c</b> <sub>0</sub> ≡ .
			<b>c</b> <sub>1</sub> ≡ .
			<b>c</b> <sub>2</sub> ≡ .
Arithmetic			<b>c</b> <sub>3</sub> ≡ .
Addition	A₊	$\equiv \lambda x y p q. x p (y p q)$	o =
Multiplication	A <sub>*</sub>	$\equiv \lambda x y z . x (y z)$	<b>C</b> <sub>n</sub> =
Exponentiation	A <sub>exp</sub>	≡ λxy.yx	
			Inde
Successor	S⁺	≡ λnfx.f(nfx)	
			Thei
			is de
Is-Zero Predicate			
ls zero		≡ λn.n (λx.False) True	Ω

indefinite value enoted

#### Substitution

The capture-avoiding substitution rules are defined as follows:

· For a variable, y;

$$y[x := N] \equiv \begin{cases} N & \text{if } x = y \\ y & \text{if } x \neq y \end{cases}$$

• For an application, (M N);

$$(M N)[x := P] \equiv M[x := P] N[x := P]$$

For an abstraction, λy.M;

$$(\lambda y.M)[x := N] \equiv \begin{cases} \lambda y.M & \text{if } x = y \\ \lambda y.(M[x := N]) & \text{if } x \neq y \text{ and } y \notin FV(N) \end{cases}$$

**Note:** M[x := N] is written M[N / x] in some books.

#### Freshness condition

Alpha-conversion is sometimes necessary to avoid changing the meaning of functions (name clashes). For example:

 $(\lambda x.y)[y := x] \equiv \lambda x.(y[y := x]) \equiv \lambda x.x \Rightarrow \lambda x.y \equiv \lambda x.x (!!)$ 

This substitution does not meet the condition;

 $(\lambda y.M)[x := N]$  requires that  $y \notin FV(N)$  (y is *fresh* for N).

After  $\alpha$ -conversion of  $\lambda x.y$  to  $\lambda z.y$ , we get:

 $(\lambda z.y)[y := x] \equiv \lambda z.(y[y := x]) \equiv \lambda z.x$ 

# **Fixed Point Theorem**

∀F ∃X FX = X where F.  $X \in \Lambda$ 

all  $\lambda$ -terms F, there exists a  $\lambda$ -term X such that  $\lambda \vdash FX = X$ .

#### urch numerals

λf.λx.x  $\lambda f.\lambda x.f x$  $\lambda f.\lambda x.f(f x)$  $\lambda f.\lambda x.f(f(f x))$ 

 $\lambda f.\lambda x.f^{n}(x)$ 

#### efinite value

≡ ωω

where  $\omega \equiv \lambda x.xx$ .

#### **Church-Rosser Theorem**

If a term M can be reduced to both N and P, then there must exist a term Q (possibly equal to N or P) to which both N and P can be reduced.

