# Notes on $\lambda$-calculus

Andrew W. Appel

## Syntax

$$
\begin{array}{rcl}
\mathit{exp} & \rightarrow & \mathit{var} \\
\mathit{exp} & \rightarrow & \lambda\ \mathit{var}\ .\ \mathit{exp} \\
\mathit{exp} & \rightarrow & \mathit{exp}\ \mathit{exp}
\end{array}
$$

Notation: $M, N$, etc. stand for *exp*s. $x, y, v$, etc. stand for *var*s.

$$
\mathit{exp} \rightarrow \underset{\text{variable}}{v} \quad | \quad \underset{\text{abstraction}}{\lambda v.M} \quad | \quad \underset{\text{application}}{M\ N}
$$

Application is (syntactically) left-associative, binds tighter than abstraction. Parentheses are used (in the obvious way) to disambiguate.

Sometimes we will introduce *constants* such as $+ \quad 1 \quad 2 \quad \ldots$

Examples:

$$
\begin{array}{rcl}
\mathit{add2} & = & \lambda x.\ +\ x\, 2 \\
\mathit{add2}' & = & \lambda x.\ +\ 2\, x \\
\mathit{add2}'' & = & +\, 2 \\
\mathit{revapp} & = & \lambda x.\lambda y.\, y\, x
\end{array}
$$

"Evaluation" is by $\beta$-*reduction* (for abstractions) and $\delta$-*reduction* (for constants).

$$
\begin{array}{ll}
 & (\lambda x.\lambda y.y\, x)(+\, 3\, 4)(\lambda x.+\ x\, 2) \\
\stackrel{\beta}{\rightarrow} & (\lambda x.+\ x\, 2)(+\, 3\, 4) \\
\stackrel{\beta}{\rightarrow} & +\, (+\, 3\, 4)\, 2 \\
\stackrel{\delta}{\rightarrow} & +\, 7\, 2 \\
\stackrel{\delta}{\rightarrow} & 9
\end{array}
$$

## Substitution

$$
\begin{array}{lcl}
[M/x]\,x & = & M \\
[M/x]\,y & = & y \\
[M/x]\,Y Z & = & ([M/x]\,Y)([M/x]\,Z) \\
[M/x]\,\lambda x.Y & = & \lambda x.Y \\
[M/x]\,\lambda y.Z & = & \lambda y.\,[M/x]\,Z \quad \text{if } x \notin \mathrm{fv}(Z) \text{ or } y \notin \mathrm{fv}(M) \\
[M/x]\,\lambda y.Z & = & \lambda w.\,[M/x]\,([w/y]\,Z) \quad w \notin \mathrm{fv}(ZM)
\end{array}
$$

## Conversion

$$
\begin{array}{llcll}
\alpha: & \lambda x.Z & \xrightarrow{\alpha} & \lambda y.\,[y/x]\,Z & \text{if } y \notin \mathrm{fv}(Z) \\
\beta: & (\lambda x.M)N & \xrightarrow{\beta} & [N/x]\,M & \\
\eta: & \lambda x.\,M\,x & \xrightarrow{\eta} & M & \text{if } x \notin \mathrm{fv}(M) \\
\delta: & +\,3\,4 & \xrightarrow{\delta} & 7 & \\
\text{general}: & M & \rightarrow & N & \text{if } M \xrightarrow{\alpha} N \text{ or } M \xrightarrow{\beta} N \text{ or } M \xrightarrow{\eta} N \text{ or } M \xrightarrow{\delta} N \\
\text{closure}: & M & \xrightarrow{*} & N & \text{if } M \equiv N + \text{or } M \rightarrow P \wedge P \rightarrow N
\end{array}
$$

## Church-Rosser Theorem

If $A \rightarrow B$ and $A \rightarrow C$ then $\exists D$ such that $B \xrightarrow{*} D$ and $C \xrightarrow{*} D$.

**Proof.** By residuals . . .

## Normal Form

$A$ is in normal form if $\nexists B \text{s.t.} A \rightarrow B$.

**Corollary of C-R theorem:** Each term has a unique normal form (if it has any normal form).

**Proof.** If $A \xrightarrow{*} B$, and $B$ is in normal form, and $A \xrightarrow{*} C$, and $C$ is in normal form, then $\exists D$ such that $B \xrightarrow{*} D$ and $C \xrightarrow{*} D$. But then it must be that $B \equiv D$ and $B \equiv D$, so $B \equiv C$.

So we say that "$B$ is <u>the</u> normal form of $A$," $B = \mathrm{nf}(A)$.

## Normalization Theorem:

To get to the NF (if there is one), always (or always eventually) take the Leftmost Outermost redex.

Different evaluation orders are like different parameter-passing mechanisms. Leftmost-outermost is like call-by-name. "Applicative order" is like call-by-value.

## Representing things in pure $\lambda$-calculus.

$$
\begin{array}{lll}
\mathit{pair} & = & \lambda x \lambda y \lambda f . f x y \\
\mathit{fst} & = & \lambda p . p(\lambda x . \lambda y . x) \\
\mathit{snd} & = & \lambda p . p(\lambda x . \lambda y . y) \\
\multicolumn{3}{l}{\text{“datatype}(\alpha, \beta)t = \text{this of } \alpha \;\mid\; \text{that of } \beta\text{ ”}} \\
\mathit{this} & = & \lambda a . \lambda f \lambda g . f\, a \\
\mathit{that} & = & \lambda b . \lambda f \lambda g . g\, b \\
\multicolumn{3}{l}{\text{“casefn } (\lambda a.M)(\lambda b.N)\text{”}} \\
\mathit{casefn} & = & \lambda \mathit{getthis} . \lambda \mathit{getthat} . \lambda x . x\; \mathit{getthis}\; \mathit{getthat}
\end{array}
$$

## Church numerals

$$
\begin{aligned}
\underline{0} &= \lambda x . \lambda f . x \\
\underline{1} &= \lambda x . \lambda f . f\, x \\
\underline{2} &= \lambda x . \lambda f . f\, (f\, x) \\
\underline{n} &= \lambda x . \lambda f . f^n(x) \\
\mathit{succ} &= \lambda n . \lambda f . \lambda x . f(n\, f\, x)
\end{aligned}
$$

## Fixed points

Consider the function $G \equiv \lambda f . \lambda n.$ if $n = 0$ then 1 else $n \times f(n-1)$

and the function $\Omega \equiv (\lambda x.\, x\, x)(\lambda x.\, x\, x)$

Then

$$
\begin{aligned}
\Omega &= \{\} \qquad \text{i.e., undefined on all inputs} \\
G(\Omega) &= \{0 \mapsto 1\} \\
G(G(\Omega)) &= \{0 \mapsto 1, 1 \mapsto 1\} \\
G(G(G(\Omega))) &= \{0 \mapsto 1, 1 \mapsto 1, 2 \mapsto 2\} \\
G(G(G(G(\Omega)))) &= \{0 \mapsto 1, 1 \mapsto 1, 2 \mapsto 2, 3 \mapsto 6\}
\end{aligned}
$$

We say that $f \sqsubset g$ ("$f$ approximates $g$") if $\forall x,\ f(x)$ is undefined or $f(x) = g(x)$.

Then $\Omega \sqsubset G(\Omega) \sqsubset G(G(\Omega)) \sqsubset G(G(G(\Omega))) \sqsubset \ldots$

How can we implement the factorial function?

$$fact = \lambda n.(succ(n))g\,\Omega\,n$$

(where $n$ is a church numeral). $(n+1)g\Omega$ is a good enough approximation to the true factorial function to evaluation factorial($n$).

Is there a function $\phi$ such that $G(\phi) = \phi$ ?

If so, then $\phi$ is the factorial function.

**Proof.** By induction:

$$\begin{aligned}
G(\phi)(0) &= 1 \\
G(\phi)(n) &= \text{if } n = 0 \text{ then } 1 \text{ else } n \times \phi(n-1) \\
&= n \times \phi(n-1) \\
&= n \times (n-1)! \qquad \text{by induction hypothesis} \\
&= n!
\end{aligned}$$

How about: $\phi \equiv (\lambda x.G(x\,x))(\lambda x.G(x\,x))$

This expands by $\beta$-conversion to

$\phi = G((\lambda x.G(x\,x))(\lambda x.G(x\,x))) = G(\phi)$

So $\phi$ is the factorial function!

More generally, we define the *fixed-point combinator* $Y \equiv \lambda f.(\lambda x.f(x\,x))(\lambda x.f(x\,x))$

And for any $H$, $YH = H(YH)$, that is, $YH$ is a fixed point of $H$.

## Syntactic sugar for $\lambda$-calculus

$$\begin{aligned}
(M, N) &\equiv \lambda f.f\,M\,N \\
\lambda(x, y).M &\equiv \lambda p.p(\lambda x.\lambda y.M) \\
\text{let } v = M \text{ in } N &\equiv (\lambda v.N)M \\
\text{let rec } v = M \text{ in } N &\equiv (\lambda v.N)(Y(\lambda v.M)) \\
0, 1, 2\ldots &\equiv \lambda f.\lambda x.x,\ \lambda f.\lambda x.f\,x,\ \lambda f.\lambda x.f\,(f\,x), \ldots \\
M + N &\equiv (\lambda x\lambda y.x\ succ\ y)MN \\
true &\equiv \lambda x\lambda y.x \\
false &\equiv \lambda x\lambda y.y \\
\text{if } M \text{ then } N \text{ else } P &\equiv M\,N\,P \\
\text{datatype } \ldots &\equiv \text{as previously described}
\end{aligned}$$