λ calculus with applications, formalised

dr. James McKinna
J.McKinna@hw.ac.uk

SPLV 2022 Lecture Notes
July 11–12 2022

Prospectus

Lecture #1 historical remarks
λcalculus & combinatory logic
β-reduction and normal forms

Lecture #2 the Church Rosser theorem

Lecture #3 the Standardisation theorem

Lecture #4 Barendregt's Lemma ; Milner's 1st Context Lemma
(for CL) (for CL)

"What I know at sixty,
I knew as well at twenty.
Forty years of a long,
a superfluous,
labour of verification."

EM Cioran
'The Trouble with Being Born'

some history

1928–1935

Göttingen
Moscow
Princeton
Cambridge

Curry invents combinatory logic (CL) following Schönfinkel

Church invents λ calculus (LC)

Kolmogorov 'Aufgabe' interpretation of intuitionistic logic

Gödel defines primitive, general recursive functions

Turing invents his machines

" bliss it was in that dawn to be alive
but to be young was very heaven "

1967 – 1971

Stanford
Chicago
Utrecht
Oxford
Eindhoven
Swansea
London
MIT

Tait computability proofs / logical relations

Tait / Martin Löf parallel reduction in λcalculus

Scott model $D_\infty \simeq D_\infty \to D_\infty$ of the λcalculus

de Bruijn AUTOMATH ; "nameless dummies"

Landin "The Mechanical Evaluation of Expressions"

Strachey / Scott λcalculus models of programming languages ; continuations

Barendregt
Wadsworth PhD theses Morris contextual equivalence

Hindley principal type schemes

1971 – 1977

Stanford
Oxford
Utrecht
Edinburgh
Stockholm
Rome
Canterbury

<u>Milner</u>
LCF
principal type schemes
Hindley-Milner typechecking
Algorithm W
fully abstract models of λ calculi (1977) 'Ist Context Lemma'

<u>Barendregt</u> 'Barendregt's Lemma' (1972, unpublished)

<u>Hyland</u> PhD thesis; solvability & head normal forms (Wadsworth)

<u>Martin Löf</u> Intuitionistic Type Theory

<u>Barwise</u> Handbook of Mathematical Logic

λ Calculus

LC

What are we talking about?

- a language of expressions

  'variables' $x, y, \ldots$

  'abstractions' $\lambda x.e$

  'applications' $e \cdot a$

  together with a notion of substitution $e[a]$

- a theory of equality, based on a theory of reduction

  $$e = e' \qquad\qquad e \twoheadrightarrow e'$$

  (actually, many such relations)

- such that (at least)

  $$(\lambda x.e) \cdot a = e[a]$$

  (where did 'x' go?)

  "to apply a function to its argument is to evaluate the body of the function with its argument substituted for..."

informal, but necessary, considerations

• distinguish $\equiv$ syntactic equality, from $=_\alpha$ 'equality up to choice of bound variables'

$$\frac{x \text{ variable}}{x =_\alpha x} \qquad \frac{M =_\alpha M' \quad N =_\alpha N'}{M \cdot N =_\alpha M' \cdot N'}$$

congruence rules for variables + applications

together with

$$\frac{M[z/x] =_\alpha N[z/y]}{\lambda x. M =_\alpha \lambda y. N}$$

z a variable not occurring in M or N

• try to define everything 'up to $=_\alpha$' $\rightsquigarrow$ lots and lots of annoying well-formedness considerations + lots & lots of arguments

substitution (the bane of our lives)

idea in $\lambda x.e$, all the 'available' occurrences of x in e should be replaced by ... the supplied argument(s)

desire that we do so in such a way as to:—

- respect $=_\alpha$ (harder than it looks)
- enforce the above idea
- don't get confused about which occurrences of variables are 'available'

<u>substitution</u> (the bane of our lives)

<u>idea</u> in $\lambda x . e$, all the 'available' occurrences of x in e should be replaced by ... the supplied argument(s)

<u>desire</u> that we do so in such a way as to:—

- respect $=_\alpha$ (harder than it looks)
- enforce the above idea
- don't get confused about which occurrences of variables are 'available'

Combinatory Logic

CL

- variables get us into no end of trouble

- the only expressions worth considering are <u>closed</u>

- closed applications easier to understand than open ones

- need to start somewhere: $\underline{K}$, $\underline{S}$.

  together with 2 rules (plus congruence)

  $$(\underline{K} \cdot M) \cdot N \to_w M \qquad ((\underline{S} \cdot P) \cdot Q) \cdot R \to_w (P \cdot R) \cdot (Q \cdot R)$$

- it turns out that this (mostly) is sufficient ... (!)

we can relate CL and LC by

defining in LC $\quad K =_{def} \lambda x y . x$

$$S =_{def} \lambda x y z . (x \cdot z) \cdot (y \cdot z)$$

and on CL terms an operation $\lambda^*_x M$ s.t.

$$\lambda^* x . \, x = \underline{I} =_{def} (\underline{S} \cdot \underline{K}) \cdot \underline{K}$$

$$\lambda^* x . \, y = y \quad (x \neq y)$$

$$\lambda^* x . (M \cdot N) = S \cdot (\lambda^*_x M) \cdot (\lambda^*_x N)$$

there are pros & cons to both approaches!

LC has the 'better' theory; CL has an 'easier' theory

The Y combinator (Curry)

for every F there is an X such that $F \cdot X =_\beta X$

take $W_F = \lambda x.\, F \cdot (x \cdot x)$ and $X = W_F \cdot W_F$

then $X \to_\beta F \cdot (W_F \cdot W_F) = F \cdot X$

so take $Y = \lambda f. (W_f \cdot W_f)$ ie. $Y = \lambda f. (\lambda x. f \cdot (x \cdot x)) \cdot (\lambda x. f \cdot (x \cdot x))$

then $Y.F \to_\beta F \cdot (W_F \cdot W_F) \twoheadleftarrow_\beta F \cdot (Y \cdot F)$

so Y produces fixed points of F for every F

Exercise (Turing) show $\Theta \equiv T \cdot T$ with $T = \lambda x f. f \cdot (x \cdot (x \cdot f))$
is also a fixed point combinator

On Formalisation

Two themes

- induction on <u>data</u> ~ induction on <u>relations</u> con<u>sidered</u> as data

(due thanks to Martin-Löf and successors: stop <u>worrying</u> about <u>induction</u>)

- the 'Inventor's Paradox' – state a harder / more general result
  - from which intended result follows (relatively) easily
  - more general result is <u>easier</u> to prove

One methodology

- systematic use of de Bruijn's (1972) 'nameless dummies'

- two distinct actions on variables,
  expressions
  relations (plus pointwise extension)

renaming : map indices to indices
ρ (analogous to variable renaming)

substitution : map indices to expressions,
σ together with suitable renaming

deBruijn indices

- variables become indices $0, 1, 2, \ldots$ (eh?)
  (application stays the same)
- binding becomes nameless λe (how?)
- substitution, similarly $e[a]$, more generally $e[\sigma]$
  $\sigma = [e_0, \ldots, e_{n-1}]$

idea keep track of the
$\underline{n}$ free (available)
variable occurrences

$$e : \mathrm{Lam}\ n \quad [n : \mathbb{N}]$$

$\boxed{=_\alpha \text{ becomes } \equiv}$

- for $i < n$ $\quad$ $\text{var } i : \text{Lam } n$

- for $n : \mathbb{N}$ $\quad$ $\dfrac{e : \text{Lam}(n+1)}{\lambda e : \text{Lam } n}$

- (application stays the same) $\quad$ $\dfrac{e : \text{Lam } n \quad a : \text{Lam } n}{e \circ a : \text{Lam } n}$

- renamings $\quad$ $\dfrac{\text{for } j < m \,.\ \rho_j \text{ is a variable } < n}{\rho = [\rho_0, \dots, \rho_{m-1}] : \text{Renaming } m\ n}$

- substitutions $\quad$ $\dfrac{e_j : \text{Lam } n \quad [j < m]}{\sigma = [\sigma_0, \dots \sigma_{m-1}] : \text{Substitution } m\ n}$

if $\lambda_{\bar{z}} e$ has n free slots

then e has one more such

<u>NB</u> '0' is the nearest slot to that λ

distance 0

$$\lambda x.\ \mathrm{var}\ x$$

$$\lambda(\ \mathrm{var}\ 0\ )$$

distance 0

$$\lambda y.\ (\lambda x.\ (x \cdot y))$$

distance 1

$$\lambda(\lambda((\mathrm{var}\ 0) \cdot (\mathrm{var}\ 1)))$$

$$\lambda x\, y.\ \big(x \cdot (\lambda z.\, y) \cdot y\big)$$

$$\lambda\, \lambda\, \big((\mathrm{var}\ 1) \cdot (\lambda (\mathrm{var}\ 1)) \cdot (\mathrm{var}\ 0)\big)$$

now we start again from $(\lambda e) \cdot a \to_\beta e[a]$ $\quad e : \mathrm{Lam}(m+1)$ $\quad a : \mathrm{Lam}\, m$

where $e[a] =_{def} e[\sigma]$ $\quad \sigma = \begin{cases} 0 \mapsto a \\ j+1 \mapsto \mathrm{var}\, j \end{cases}$

in general define $e[\rho]$ by
$$(\mathrm{var}\, j)[\rho] = \mathrm{var}(\rho_j) \qquad j < m$$
$$(\lambda e)[\rho] = \lambda(e[\rho\uparrow])$$
$$(e \cdot a)[\rho] = e[\rho] \cdot a[\rho]$$

$e[\sigma]$ by
$$(\mathrm{var}\, j)[\sigma] = \sigma_j$$
$$(\lambda e)[\sigma] = \lambda(e[\sigma\uparrow])$$
$$(e \cdot a)[\sigma] = e[\sigma] \cdot a[\sigma]$$

where

$$\frac{\rho : \mathrm{Renaming}\ m\ n}{\rho\uparrow : \mathrm{Renaming}\,(m+1)\,(n+1)}$$

$$(\rho\uparrow)_0 = 0$$

$$(\rho\uparrow)_{j+1} = (\rho_j) + 1$$

$$\frac{\sigma : \mathrm{Substitution}\ m\ n}{\sigma\uparrow : \mathrm{Substitution}\,(m+1)\,(n+1)}$$

$$(\sigma\uparrow)_0 = \mathrm{var}\,0 : \mathrm{Lam}\,(n+1)$$

$$(\sigma\uparrow)_{j+1} = (\sigma_j)[wk_n]$$

where $wk_n : \mathrm{Renaming}\ n\,(n+1)$ $(wk_n)_i = i+1$

Whence the methodology of prove/construct by closure under renaming

then by closure under substitution