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
λ-calculus & combinatory logic
β-reduction and normal forms

Lecture #2 the Church Rosser theorem

Lecture #3 the Standardisation theorem

Lecture #4 Barendregt's Lemma (for CL) ; Milner's 1st Context Lemma (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 — Tait — computability proofs / logical relations

Chicago — Tait / Martin-Löf — parallel reduction in λ calculus

Utrecht — Scott — model $D_\infty \simeq D_\infty \to D_\infty$ of the λ calculus

Oxford — de Bruijn — AUTOMATH ; "nameless dummies"

Eindhoven — Landin — "The Mechanical Evaluation of Expressions"

Swansea — Strachey / λ calculus models of programming languages ; continuations

London — Scott

MIT — Barendregt — PhD theses — Morris — contextual equivalence

Wadsworth

Hindley — principal type schemes

1971 – 1977

Stanford

Oxford

Utrecht

Edinburgh

Stockholm

Rome

Canterbury

Milner — LCF
principal type schemes
Hindley-Milner typechecking
Algorithm W
fully abstract models of λ calculi (1977) — `1st Context Lemma'

Barendregt — `Barendregt's Lemma' (1972, unpublished)

Hyland — PhD thesis; solvability & head normal forms (Wadsworth)

Martin Löf — Intuitionistic Type Theory

Barwise — 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

  $e = e'$ $\qquad$ $e \longrightarrow\!\!> e'$

  (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'

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'

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 ... (∇_0)

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

then $Y \cdot F \to_\beta F \cdot (W_f \cdot W_f) \underset{\beta}{\longleftarrow} F \cdot (Y \cdot F)$

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 data ~ induction on relations considered as data

  (due thanks to Martin-Löf and successors: stop worrying about induction)

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

One methodology

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

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

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

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

de Bruijn 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
n free (available)
variable occurrences

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

- for $i < n$ — $\mathrm{var}\ i : \mathrm{Lam}\ n$
- for $n : \mathbb{N}$ — $\dfrac{e : \mathrm{Lam}\,(n+1)}{\lambda e : \mathrm{Lam}\ n}$

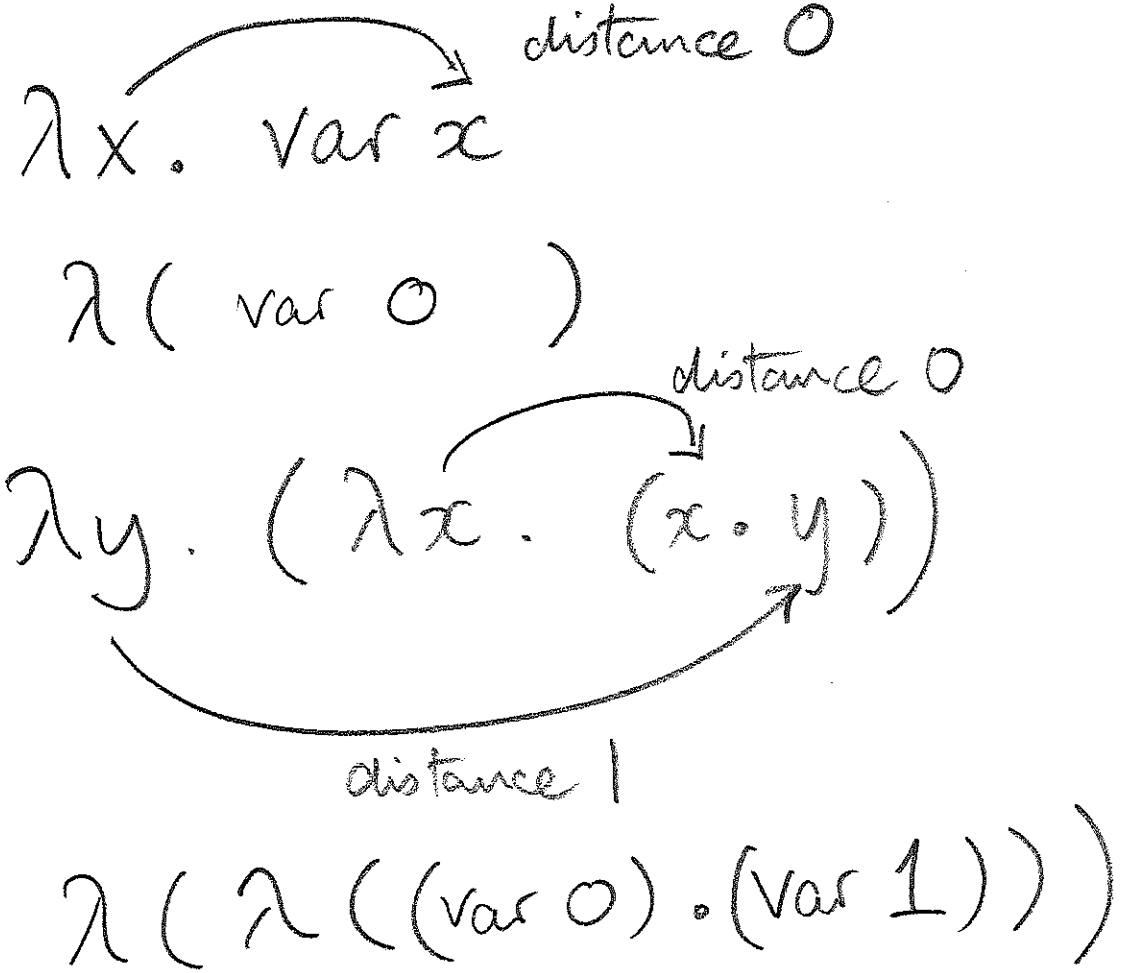
$=_\alpha$ becomes $=$

- (application stays the same) — $\dfrac{e : \mathrm{Lam}\ n \qquad a : \mathrm{Lam}\ n}{e \circ a : \mathrm{Lam}\ n}$
- renamings — $\dfrac{\text{for } j < m \text{ . } \xi_j \text{ is a variable} < n}{\xi = [\xi_0, \ldots, \xi_{m-1}] : \mathrm{Renaming}\ m\ n}$
- Substitutions — $\dfrac{e_j : \mathrm{Lam}\ n \quad [\, j < m \,]}{\sigma = [\sigma_0, \ldots \sigma_{m-1}] : \mathrm{Substitution}\ m\ n}$

if λe has n free slots

then e has one more such

NB '0' is the nearest slot to that λ


distance 0
λx. var x
λ(var 0)
distance 0
λy. (λx. (x . y))
distance 1
λ(λ((var 0) . (var 1)))

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

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

$e : \mathrm{Lam}(m+1)$

$a : \mathrm{Lam}\, m$

now we start again from

$$(\lambda e) \cdot a \longrightarrow_\beta e[a]$$

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

in general define $e[\xi]$ by

$$(\mathrm{var}\, j)[\xi] = \mathrm{var}(\xi j) \qquad j < m$$
$$(\lambda e)[\xi] = \lambda(e[\xi\uparrow])$$
$$(e \cdot a)[\xi] = e[\xi] \cdot a[\xi]$$

$e[\sigma]$ by

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

where

$$\frac{\varsigma : \text{Renaming } m\ n}{\varsigma\uparrow : \text{Renaming } (m+1)(n+1)}$$

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

$$(\varsigma\uparrow)_{j+1} = (\varsigma\, j) + 1$$

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

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

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

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

Whence the methodology of

prove/construct by closure under renaming

then by closure under substitution