A calculus with applications, formalised

dr. James McKinna
J. McKinna @ huw.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
- Milner's 1st Context Lemma
  (for CL)
"What I knew at sixty,  
I knew as well at twenty.  
Forty years of a long,  
a superfluous,  
labour of verification."

E M 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 (CC)

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 \( \lambda \) calculus

Scott model \( D_\omega = D_\omega \to D_\omega \) of the \( \lambda \) calculus
de Bruijn AUTOMATH; "nameless dummies"

Landin "The Mechanical Evaluation of Expressions"

Strachey/\( \lambda \) calculus models of programming

Swiss languages; continuations

Barendregt PhD theses Morris contextual equivalence

Wadsworth principal type schemes Hindley
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) Lemma

Barendregt
Barendregt's Lemma
(1972, unpublished)

Hyland
PhD thesis; Solvability & head normal forms

Martin Løf

Intuitionistic Type Theory

Barwise

Handbook of Mathematical Logic
A 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 \( \equiv \)

(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 \( x \) variable \( \vdash x =_x x \) syntactic equality, from \( =_x \) equality up to choice, of bound variables

\[ M =_x M' \quad N =_x N' \]
\[ M \cdot N =_x M' \cdot N' \]
congruence rules for variables + applications

- try to define everything 'up to \( =_x \)' \( \rightarrow \) lots and lots of annoying well-formedness considerations + lots & lots of arguments
idea in \( \lambda \text{e} \), all the ‘available’ occurrences of \( x \) in \( \text{e} \) should be replaced by \( \ldots \) 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’
idea in A = e, all the available arguments should be replaced by •••. the supplied arguments. For

Substitution (the bane of our lives)

desire. That are so in similar way as to:

x

enforce the above idea

x = x (don't get confused about which occurrences

if variables are available).
Combinatory Logic

CL
• variables get us into no end of trouble

• the only expressions worth considering are closed

• closed applications easier to understand than open ones

• need to start somewhere: $K, S$

together with 2 rules (plus congruence)

$$(K \cdot M) \cdot N \rightarrow^M ((S \cdot P) \cdot Q) \cdot R \rightarrow^\omega (P \cdot R) \cdot (Q \cdot R)$$

• it turns out that this (mostly) is sufficient ... $(\nabla)$
we can relate CL and LC by

defining in LC

\[ K = \text{def } \lambda x y. x \]

\[ S = \text{def } \lambda x y z. (x z) \cdot (y z) \]

and on CL terms

\[ \lambda x M \quad \text{s.t.} \quad \lambda x. x = I = \text{def } (S \cdot K) \cdot K \]

\[ \lambda x. y = y \quad (x \neq y) \]

\[ \lambda x. (M \cdot N) = S \cdot (\lambda x M) \cdot (\lambda x N) \]

can be operation

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

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

so take $Y = \lambda F. (W_F \cdot W_F)$ i.e. $Y = \lambda F. (\lambda x. F \circ (x \circ x))$

then $Y \cdot F \to \beta F \cdot (W_F \cdot W_F) \iff F \cdot (Y \cdot F)$

so $Y$ produces fixed points of $F$ for every $F$

Exercise (Turing) show $\Theta = T \cdot T$ with $T = \lambda x f. f \cdot (x \circ (x \circ f))$

is also a fixed point combinator
On Formalisation
Two themes

- induction on data \rightarrow \text{induction on relations considered as data}

(dues thanks to Martin-Loef 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: map indices to indices
  (analogous to variable renaming)

  substitution: map indices to expressions, together with suitable renaming
De Bruijn indices

- Variables become indices $0, 1, 2, \ldots$ (eh?)
  (approximating the same)
- Binding becomes nameless $\lambda e$ (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

$e : \text{Lam} \ n \ [n : \text{N}]$
for \( i < n \)

\[
\text{var } i : \text{Lam } n
\]

for \( n : \text{N} \)

\[
\text{e : Lam}(n+1)
\]

\[
\lambda e : \text{Lam } n
\]

(application stays the same)

renamings

\[
\text{e : Lam } n \quad a : \text{Lam } n
\]

\[
e \circ a : \text{Lam } n
\]

for \( j < m \), \( g_j \) is a variable \(< n \)

\[
g = [g_0, \ldots, g_{m-1}] : \text{Renaming } m \text{ n}
\]

\[
e_j : \text{Lam } n \quad [j < m]
\]

\[
\sigma = [\sigma_0, \ldots, \sigma_{m-1}] : \text{Substitution } m \text{ n}
\]
if \( \lambda \in E \) has \( n \) free slots then \( e \in E \) has one more such 

NB '0' is the nearest slot to that \( \lambda \)
\[ \lambda x. \text{var} \ x \]

\[ \lambda (\text{var} \ 0) \]

\[ \lambda y. (\lambda x. (x, y)) \]

\[ \lambda (\lambda (\text{var} \ 0) \cdot (\text{var} \ 1)) \]
\( \lambda x y \cdot (x \cdot (\lambda z. y) \cdot y) \)

\( \lambda \lambda \cdot (\text{var 1}) \cdot (\lambda (\text{var 1})) \cdot (\text{var 0}) \)
Now we start again
from

where \( e[a] = \text{def} e[\sigma] \) \( \sigma = \{ 0 \mapsto a \\
\phantom{0} \; j+1 \mapsto \text{var} j \} \)

in general define

\( e[g] \) by \((\text{var} j)[g] = \text{var}(g_j) \; j < m \)
\( (\lambda e)[g] = \lambda(e[g \uparrow]) \)
\( (e \cdot a)[g] = e[g] \cdot a[g] \)

\( e[\sigma] \) by \((\text{var} j)[\sigma] = \sigma_j \)
\( (\lambda e)[\sigma] = \lambda(e[\sigma \uparrow]) \)
\( (e \cdot a)[\sigma] = e[\sigma] \cdot a[\sigma] \)
\( e : \text{Lamb}(m+1) \)
\( a : \text{Lamb} m \)
\[ f : \text{Renaming } m \leftrightarrow n \]
\[ g : \text{Renaming } (m+1)(n+1) \]
\[ \Delta : \text{Substitution } m \leftrightarrow n \]
\[ \sigma : \text{Substitution } (m+1)(n+1) \]

\[ (g^\uparrow)_0 = 0 \]
\[ (g^\uparrow)_{j+1} = (g_j) + 1 \]
\[ (\Delta^\uparrow)_0 = \varnothing \quad \text{var } \varnothing : \text{Lamb}(n+1) \]
\[ (\sigma^\uparrow)_{j+1} = (\sigma_j)[w_k_n] \]

where \[ w_k_n : \text{Renaming } n(m+1) \]
\[ (w_k_n)_i = i + 1 \]

Whence the prove/construct by closure under renaming

then by closure under substitution