PROOF101, Spring 2026, Lecture Slides 01

American University of Beirut

Untyped Lambda-Calculus

Assaf Kfoury

March 2026

What Is the Lambda Calculus?

- first developed in the 1930's to formalize methods by which mathematical functions manipulate inputs to compute outputs
- as such, the lambda calculus is an an alternative to ***Turing machines***, ***partial μ-recursive functions***, ***Markov algorithms***, and several other (equally powerful) formal models of computation

closer to our concerns in this course, the λ-calculus is:

- succint way to write and apply ***anonymous*** computable functions, *i.e.*, without naming them
- most convenient to study ***evaluation strategies*** (*termination conditions, confluence, normal forms, recursion, tail recursion, fixed points, . . .*)
- most convenient to study ***typing systems*** for programming languages
- most convenient to study ***formal semantics*** of programming languages
- linchpin of a profound connection between logic (namely, *proof theory*) and computer programming, the so-called ***Curry-Howard Isomorphism***

Outline of Lecture Slides 01

- most details in these lecture slides are from the book:

  Lectures on the Curry-Howard Isomorphim

  by Sorensen and Urzyczyn – henceforth denoted [LCHI]

- headings in this set of slides (a little different from [LCHI, Chapt 1]):
  - syntax of the λ-calculus [LCHI, Sect 1.2]
  - free and bound variables [LCHI, Sect 1.2]
  - substitution and α-conversion [LCHI, Sect 1.2]
  - β-reduction and η-reduction [LCHI, Sect 1.3]
  - Church-Rosser Theorem [LCHI, Sect 1.4]
  - more on reduction [LCHI, Sect 1.5]
  - $\lambda\mathbf{I}$-calculus and Conservation Theorem [LCHI, Sect 1.6]
  - expressibility and undecidability [LCHI, Sect 1.7]

Outline of Lecture Slides 01

important things to remember from [LCHI, Chapter 1] :

- *Church-Rosser/Confluence Theorem*, slide 13
- *Leftmost Reduction Is Normalizing*, slide 17
- *Conservation Theorem*, slide 18
- *λ-Definability of Partial Recursive Functions*, slide 20

care to know more about ***untyped lambda-calculus*** beyond [LCHI] ?

- an excellent reference is Barendregt's book:

  The Lambda Calculus, Its Syntax and Semantics

  which covers far more than we need for this course
- another excellent reference is Hindley's book:

  Basic Simple Type Theory

  although its coverage is far more limited than Barendregt's book

Syntax of the Lambda Calculus

- a ***λ-term*** is a *variable* x (also called a *λ-variable* or an *object variable*), or an *application* $(M\,N)$, or an *abstraction* $(\lambda x\,M)$
- with Λ denoting the set of λ-terms, an extended-BNF definition of Λ:

$$M, N \in \Lambda ::= x \mid (M\,N) \mid (\lambda x\,M)$$

  where x ranges over a countably infinite set of variables
- conventions for omitting matching parentheses in [LCHI, 1.2.2, p 3]
- **WARNING**: contrary to [LCHI], we do not distinguish between *pre-terms* (in [LCHI, 1.2.1, p 3]) and *λ-terms* (in [LCHI, 1.2.14, p 7])
- instead, we identify λ-terms up to *α-conversion*,[1] defined in [LCHI, 1.2.8, p 6] and summarized on slides 8 and 10

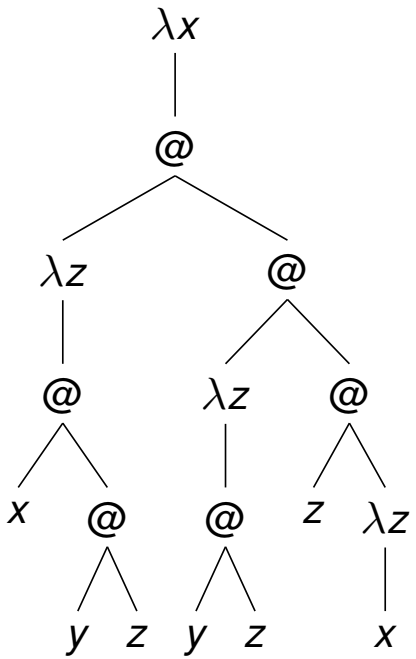
[1] this is another name for a time-honored practice of seasoned programmers: *"consistent renaming of bound variables does not change the meaning of programs."*

Syntax of the Lambda Calculus

below is an example of a fully parenthesized λ-term M – the indeces on matching parentheses are not part of M, only added to help with the tedious parsing of M:

$$M \triangleq \underset{1}{\Big(}\lambda x \underset{2}{\Big(}\underset{3}{\Big(}\lambda z \underset{4}{\big(} x \underset{5}{\big(} y\ z \underset{5}{\big)}\big)\underset{4}{\Big)}\Big)\underset{3}{\Big(}\underset{3}{\Big(}\underset{4}{\big(}\underset{5}{\big(}\lambda z \underset{6}{\big(} y\ z\underset{6}{\big)}\big)\underset{5}{\big)}\underset{4}{\big(}\underset{4}{\big(} z \underset{5}{\big(}\lambda z\ x\underset{5}{\big)}\big)\underset{4}{\big)}\Big)\underset{3}{\Big)}\Big)\underset{2}{\Big)}\underset{1}{\Big)}$$

the *syntax tree* (or *tree representation*) of M is much easier to read:



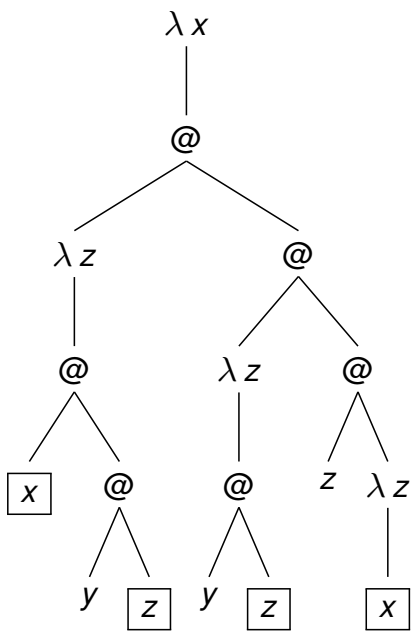
- a node labelled with '@' denotes an application
- *syntax trees* ignore all matching parentheses!

Free and Bound Variables

- the set of free variables $\mathrm{FV}(M)$ of λ-term M [LCHI, 1.2.3, p 4] :

$$\begin{aligned} \mathrm{FV}(x) &\triangleq \{x\} \\ \mathrm{FV}(\lambda\, x\, P) &\triangleq \mathrm{FV}(P) - \{x\} \\ \mathrm{FV}(P\, Q) &\triangleq \mathrm{FV}(P) \cup \mathrm{FV}(Q) \end{aligned}$$

- for the λ-term M on slide 6 :



- bound variable-occurrences are boxed
- all occurrences of variable x are *bound*
- all occurrences of variable y are *free*
- variable z occurs both *bound* and *free*
- there are three binding occurrences of z, the rightmost binding of z is *dummy*

Substitution and α-Conversion

- *the substitution of N for x in M*, written $M[x := N]$, may or may not be defined[2]
  - if $x \notin \mathrm{FV}(M)$ then $M[x := N] = M$, else
    if $x \in \mathrm{FV}(M)$ then $M[x := N]$ is defined provided:
    there is no subterm $(\lambda y\, P)$ of M such that $x \in FV(\lambda y\, P)$ and $y \in FV(N)$
  - this proviso is necessary and sufficient to avoid *variable-name capture* and is enforced by the formal definition in [LCHI, 1.2.4, p 4]
- *α-conversion*, denoted $=_\alpha$, is the least transitive and reflexive relation such that:
  - if $y \notin \mathrm{FV}(M)$ and $M[x := y]$ is defined, then $(\lambda x\, M) =_\alpha (\lambda y\, M[x := y])$
  - if $M =_\alpha N$, then
    1. $(\lambda x\, M) =_\alpha (\lambda x\, N)$ for all variables x
    2. $(M\, P) =_\alpha (N\, P)$ for all λ-terms P
    3. $(P\, M) =_\alpha (P\, N)$ for all λ-terms P

  further details on α-conversion are in [LCHI, Sect 1.2, pp 6-9]

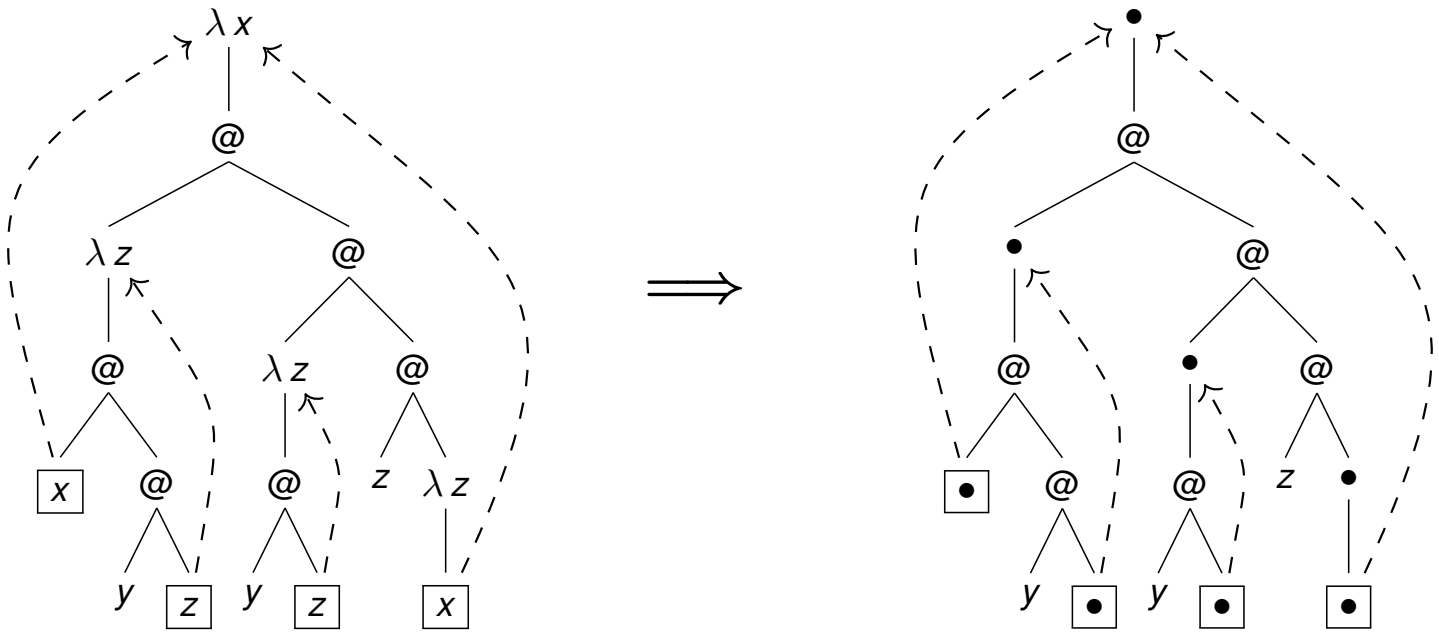
[2] The expression '$M[x := N]$' is not a λ-term, since the suffix part '$[x := N]$' is not mentioned in the definition of λ-terms. We have to understand '$M[x := N]$' differently, as a *meta-notation* for the transformation of a *partial function that takes two λ-terms, M and N, as input and returns another λ-term as output* (which may or may not be defined, thus making it *partial*). It is important to keep in mind the distinction between the *object level* and the *meta level*: M and N refer to expressions at the *object level*, while '$M[x := N]$' as such, not its result after the substitution, refer to an expression at the *meta level*.

Substitution and α-Conversion

- **FACT**: for all λ-terms M and N,
  there is a λ-term $M' =_\alpha M$ such that $M'[x := N]$ is defined
- hence, if we identify λ-terms up to α-conversion, the substitution $M[x := N]$ is always defined – and there is no risk of name capture

Substitution and α-Conversion

- for λ-term M on slide 6, we can use *upward edges* in the syntax tree to eliminate names of all *bound variables*, leaving only names of *free variables*:



- the *syntax tree*+*upward edges* on the right is a unique canonical representation of λ-term M and all λ-terms α-convertible to M

β-Reduction and η-Reduction

- all *notions of reduction* are binary relations, each denoted by an arrow in $\{\rightarrow, \twoheadrightarrow, \Rightarrow, \ldots\}$, appropriately decorated and placed between its two arguments
- *one-step β-reduction*, denoted $\rightarrow_\beta$, is defined by:
  - basis: $(\lambda x\, P)\, Q \rightarrow_\beta P[x := Q]$
  - compatibility: if $M \rightarrow_\beta N$ then

$$\begin{aligned} M\, P &\rightarrow_\beta N\, P \\ P\, M &\rightarrow_\beta P\, N \\ (\lambda x\, M) &\rightarrow_\beta (\lambda x\, N) \end{aligned}$$

- more compactly, as in [LCHI, 1.3.2, p 10], $\rightarrow_\beta$ is the *least compatible relation* on Λ satisfying $(\lambda x\, P)\, Q \rightarrow_\beta P[x := Q]$
- *multi-step β-reduction*, denoted $\twoheadrightarrow_\beta$, is the transitive reflexive closure of $\rightarrow_\beta$
- *β-equality*, denoted $=_\beta$, is the least equivalence relation containing $\rightarrow_\beta$
- more conventions related to β-reduction in [LCHI, Sect 1.3, pp 10-11]

β-Reduction and η-Reduction

- *one-step η-reduction*, denoted $\to_\eta$, is defined by:
  - basis: $(\lambda x\,(M\,x)) \to_\eta M$ where $x \notin M$
  - compatibility: if $M \to_\eta N$ then

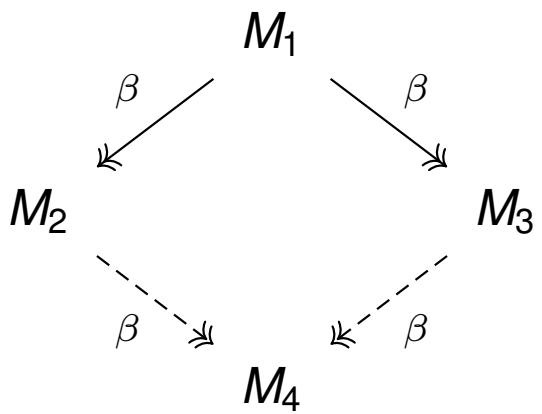
$$\begin{aligned} M\,P &\to_\eta N\,P \\ P\,M &\to_\eta P\,N \\ (\lambda x\,M) &\to_\eta (\lambda x\,N) \end{aligned}$$

- more compactly, as in [LCHI, 1.3.9, p 11] , $\to_\eta$ is the *least compatible relation* on Λ satisfying $(\lambda x\,(M\,x)) \to_\eta M$ where $x \notin M$
- more conventions and results re: η-reduction in [LCHI, Sect 1.3, pp 11-12]

Church-Rosser Theorem

- **THEOREM** (Church-Rosser)
  If $M_1 \twoheadrightarrow_\beta M_1$ and $M_1 \twoheadrightarrow_\beta M_3$,
  then there is a $M_4 \in \Lambda$ such that $M_2 \twoheadrightarrow_\beta M_4$ and $M_3 \twoheadrightarrow_\beta M_4$

  a little commutative diagram nicely describes the CR Theorem
  and gives it its alternative names, the Confluence Theorem and the
  Diamond Property of $\twoheadrightarrow_\beta$:



- non-trivial proof in [LCHI, Sect 1.4, pp 12-14]

More on Reduction

- for notions of reduction $R \in \{\rightarrow_\beta, \rightarrow_{\beta\eta}, \ldots\}$:
  - M is a *R-normal form*, or in *R-normal form*, if there is ***no*** N s.t. $M \rightarrow_R N$
  - $M \in \mathrm{WN}_R$ if M is *R-normalizing*, *i.e.*
    there is a R-reduction sequence from M to a R-normal form N
  - $M \in \mathrm{SN}_R$ if M is *R-strongly normalizing*, *i.e.*
    every R-reduction sequence from M terminates at a R-normal form N
  - $M \in \infty_R$ if $M \notin \mathrm{SN}_R$
  - if $M \in \mathrm{SN}_R$ then $M \in \mathrm{WN}_R$, but not conversely
  - relation $=_R$ (*R-equality* or *R-conversion*) is
    the least equivalence relation containing $\rightarrow_R$
  - more on the interplay between different R-reduction strategies in
    [LCHI, Sect 1.5-1.6, p 14-19]

More on Reduction

- *parallel β-reduction*, denoted $\Rightarrow_\beta$, allows for simultaneous reduction of zero or more β-redexes and is defined as the least relation on Λ s.t.
  - $x \Rightarrow_\beta x$
  - if $P \Rightarrow_\beta Q$ then $(\lambda x\, P) \Rightarrow_\beta (\lambda x\, Q)$
  - if $P_1 \Rightarrow_\beta Q_1$ and $P_2 \Rightarrow_\beta Q_2$ then $(P_1\, P_2) \Rightarrow_\beta (Q_1\, Q_2)$
  - if $P_1 \Rightarrow_\beta Q_1$ and $P_2 \Rightarrow_\beta Q_2$ then $(\lambda x\, P_1)\, P_2 \Rightarrow_\beta Q_1[x := Q_2]$
- that '$\Rightarrow_\beta$' correctly defines *the simultaneous reduction of zero or more β-redexes* is confirmed by [LCHI, Lemma 1.4.2, p 13]

More on Reduction

- the *leftmost* β*-redex* is the β-redex whose λ is the furthest to the left
- every λ-term is in one of two forms:
  - $(\lambda \vec{z}.\, x\, \vec{R})$
  - $(\lambda \vec{z}.\, (\lambda x.\, P)\, Q\, \vec{R})$, in which case $(\lambda x.\, P)\, Q$ is called a *head* β*-redex*

  where $\vec{z}$ and $\vec{R}$ are possibly empty sequences of variables and terms
- the *head* β*-redex*, if it exists, is the *leftmost* β*-redex*, but not vice-versa:

  e.g., the underlined β-redex in $\lambda x.\, x\, \underline{((\lambda y.\, P)\, Q)}$ is *leftmost* but not *head*
- every β-redex which is not *head* is called *internal*
- notation:
  - $M \xrightarrow{\ell}_\beta N$, contracting the *leftmost* redex (deterministic)
  - $M \xrightarrow{h}_\beta N$, contracting the *head* redex (deterministic)
  - $M \xrightarrow{i}_\beta N$, contracting an *internal* redex (non-deterministic)

More on Reduction

- more on the interplay between $\{\Rightarrow_\beta, \xrightarrow{\ell}_\beta, \xrightarrow{h}_\beta, \xrightarrow{i}_\beta\}$ in [LCHI, Sect 1.5, p 14-17] in preparation for the proof of next theorem
- **THEOREM** (Leftmost Reduction Is Normalizing)
  if M has a β-normal form N, then $M \xrightarrow{\ell}\!\!\!\!\rightarrow_\beta N$
- corollaries of preceding theorem in [LCHI, p 17]

$\lambda\mathbf{I}$-Calculus and Conservation Theorem

- a ***λI-term*** is a *variable* x, or an *application* $(M\,N)$, or an *abstraction* $(\lambda x\,M)$ where $x \in \mathrm{FV}(M)$
- in all $\lambda\mathbf{I}$-terms, a variable-binding 'λx' is never dummy
- **THEOREM** (Conservation)
  1. for every if $\lambda\mathbf{I}$-term M, if $M \in \mathrm{WN}_\beta$ then $M \in \mathrm{SN}_\beta$
  2. for every if $\lambda\mathbf{I}$-term M, if $M \in \infty_\beta$ and $M \to_\beta N$ then $N \in \infty_\beta$
- non-trivial proof in [LCHI, Sect 1.6]

Expressibility and Undecidability

- *Church numerals* (an encoding of the set $\mathbb{N}$ of natural numbers):

$$\begin{aligned}
\mathbf{0} &\triangleq \lambda f\, x.\, x \\
\mathbf{1} &\triangleq \lambda f\, x.\, f\, x \\
\mathbf{2} &\triangleq \lambda f\, x.\, f(f\, x) \\
&\dots \\
\boldsymbol{c}_n &\triangleq \lambda f\, x.\, f^n(x) \\
&\dots
\end{aligned}$$

- a partial function $f : \mathbb{N}^k \to \mathbb{N}$ is *λ-definable* iff there is a λ-term F s.t.
  - if $f(n_1, \dots, n_k) = m$ then $F\boldsymbol{c}_{n_1} \cdots \boldsymbol{c}_{n_k} =_\beta \boldsymbol{c}_m$
  - if $f(n_1, \dots, n_k)$ is undefined then $F\boldsymbol{c}_{n_1} \cdots \boldsymbol{c}_{n_k}$ has no normal form

Expressibility and Undecidability

- **THEOREM** (λ-Definability of Partial Recursive Functions)
  All partial recursive functions are λ-definable
- **COROLLARY** (Undecidability)
  1. there is no algorithm which, given an arbitrary $M \in \Lambda$, can decide whether M has a normal form
  2. there is no algorithm which, given an arbitrary $M \in \Lambda$, can decide whether M is strongly normalizing
- proof of the theorem and its corollary in [LCHI, Sect 1.7]

(THIS SLIDE INTENTIONALLY LEFT BLANK)