Type Theory

Lecture 1a: Crash course λ-calculus and natural deduction; Formulas-as-types

H. Geuvers

Radboud University
Nijmegen, NL

OPLSS 2026 Summer 2026

Outline

Untyped λ-calculus crash course

Natural Deduction

Formulas as Types

λ-abstraction

Defining a function

$$\begin{aligned} f(x) &:= x^2 + 2 \\ f &: x \mapsto x^2 + 2 \\ g(x, y) &:= x^2 + y + 2 \end{aligned}$$

In λ-calculus we use λ-abstraction:

$$\begin{aligned} f &:= \lambda x.\, x^2 + 2 \\ g &:= \lambda x.\, \lambda y.\, x^2 + y + 2 \end{aligned}$$

- distinguish between term with a variable $x^2 + 2$ and the function $\lambda x.\, x^2 + 2$ that sends x to $x^2 + 2$.
- make explicit which variables are abstracted over.
- clearly distinguish between free and bound (occurrences) of variables.

Application

We have seen the functions f and g:

$$\begin{aligned} f &:= \lambda x.\, x^2 + 2 \\ g &:= \lambda x.\, \lambda y.\, x^2 + y + 2 \end{aligned}$$

Application:

$$f(3) \quad \text{no!} \qquad f\,3 \quad \text{or} \quad f \cdot 3 \quad \text{or} \quad (f\,3)$$

$\Rightarrow$ application is a binary operator which is usually not written.

Giving two arguments:

$$(g\,3)\,4 \qquad \text{or just} \quad g\,3\,4$$

because we omit brackets by associating them to the left.

Untyped λ-calculus

Untyped λ-calculus = Variables + λ-abstraction + application

$$\Lambda ::= \mathrm{Var} \mid (\Lambda\,\Lambda) \mid (\lambda \mathrm{Var}.\Lambda)$$

Notation

$M\,N\,P$ denotes $(M\,N)\,P$ (so not $M\,(N\,P)$)

$\lambda xyz.M$ denotes $\lambda x.\lambda y.\lambda z.M$ (or more precisely $\lambda x.(\lambda y.(\lambda z.M))$)

Examples:

- $\mathbf{I} := \lambda x.x$
- $\mathbf{K} := \lambda x\,y.x$
- $\mathbf{S} := \lambda x\,y\,z.x\,z(y\,z)$
- $\omega := \lambda x.x\,x$
- $\Omega := \omega\,\omega$

Computing with λ-terms

Computation is done via the β-rule

$$(\lambda x.x^2 + 2)\, 3 \quad \rightarrow_\beta \quad 3^2 + 2$$

DEFINITION β-equality, written as $=_\beta$ is the equivalence relation generated from the β-reduction rule:

$$\boxed{(\lambda x.M)\, P \quad \rightarrow_\beta \quad M[x := P]}$$

where $M[x := P]$ denotes the substitution of P for all occurrences of x in M.

That $\rightarrow_\beta$ is a term reduction means that it is closed under the term-forming-operators. More precisely we have

$$\frac{M \rightarrow_\beta M'}{M\, P \rightarrow_\beta M'\, P} \qquad \frac{P \rightarrow_\beta P'}{M\, P \rightarrow_\beta M\, P'} \qquad \frac{M \rightarrow_\beta M'}{\lambda x.M \rightarrow_\beta \lambda x.M'}$$

Examples

Remember $\mathbf{I} := \lambda x.x$, $\mathbf{K} := \lambda x\, y.x$, $\mathbf{S} := \lambda x\, y\, z.x\, z(y\, z)$, $\omega := \lambda x.x\, x$, $\Omega := \omega\, \omega$.

$$\begin{array}{rcl} \mathbf{I}\, P & \to_\beta & P \\ \mathbf{K}\, P\, Q & \to_\beta & \ldots \to_\beta P \\ \Omega & \to_\beta & \Omega \end{array}$$

$$\begin{array}{rcl} (\lambda x\, y.y\, x)\, P & \to_\beta & \lambda y.y\, P \\ (\lambda x\, y.y\, x)\, y & \overset{??}{\to}_\beta & \lambda y.y\, y \end{array}$$

No!

$\lambda y.M$ binds all occurrences of y in M. We cannot just substitute a term with a free y inside M.

Free and bound variables, alpha-equivalence

- $\lambda y.M$ binds all occurrences of y in M.
- We distinguish bound variables and free variables in a term: BV(M) and FV(M).
  (Better to say: bound and free occurrences of variables.)
- We consider terms modulo renaming of bound variables (also called "modulo α-equality"):

$$\lambda x.M \equiv \lambda y.M[x := y]$$

  if y does not occur in M.

A more precise definition of $\rightarrow_\beta$:

$$(\lambda x.M)\, P \rightarrow_\beta M[x := P]$$

where the substitution $M[x := P]$ is defined by:
(1) rename the bound variables in M that occur free in P, obtaining M';
(2) replace all free occurrences of x in M' by P.

Alpha equivalence

Two terms M, N are α-equal, $M \equiv N$, in case they can be obtained from eachother via renaming bound variables.

EXAMPLES

$$\lambda x.\lambda y.x\,y \overset{??}{\equiv} \lambda y.\lambda x.y\,x$$

$$\lambda x.\lambda y.x\,y \overset{??}{\equiv} \lambda x.\lambda y.y\,x$$

$$\lambda x.\lambda y.x\,y \overset{??}{\equiv} \lambda x.\lambda y.y\,y$$

$$\lambda x.\lambda x.x\,x \overset{??}{\equiv} \lambda x.\lambda y.y\,y$$

Multi-step reduction and β-equality

- $\twoheadrightarrow_\beta$ is the transitive reflexived closure of $\rightarrow_\beta$.
  So $M \twoheadrightarrow_\beta P$ iff M β-reduces to P in 0 or more steps.
- $=_\beta$ is the transitive, reflexive, symmetric closure of $\rightarrow_\beta$.
  So $=_\beta$ is the least congruence obtained from $\rightarrow_\beta$.

EXAMPLES of reductions:

$$
\begin{array}{rcl}
\mathbf{I}\,P & \rightarrow_\beta & P \\
\mathbf{K}\,P\,Q & \twoheadrightarrow_\beta & P \\
\mathbf{K}\,\mathbf{I}\,P\,Q & \twoheadrightarrow_\beta & Q \\
\mathbf{S}\,\mathbf{K}\mathbf{K} & \twoheadrightarrow_\beta & \mathbf{I} \\
\Omega & \rightarrow_\beta & \Omega
\end{array}
$$

Is λ-calculus consistent?

Why does λ-calculus "make sense"?
Could it be the case that $M =_\beta P$ for all M, P? (Then λ-calculus would be inconsistent...)

THEOREM λ-calculus satisfies the Church-Rosser property.

COROLLARY $\mathbf{K} \neq_\beta \mathbf{I}$ and so λ-calculus is consistent.

The computational power of λ-calculus

Untyped λ-calculus is Turing complete
Its power lies in the fact that you can solve recursive equations:
Is there a term P such that

$$P\,x =_\beta x\,P\,x\,P?$$

Is there a term M such that

$$M\,x =_\beta \textbf{if}\ (\textbf{Zero}\,x)\ \textbf{then}\ 1\ \textbf{else}\ \mathsf{Mult}\,x\,(M\,(\mathsf{Pred}\,x))?$$

Yes, because we have a fixed point combinator:
- $\mathbf{Y} := \lambda f.(\lambda x.f(x\,x))(\lambda x.f(x\,x))$

Property:

$$\boxed{\mathbf{Y}\,f =_\beta f(\mathbf{Y}\,f)}$$

Untyped λ-calculus (ctd.)

Solving recursive equations using the fixed point combinator:

- For M a λ-term, $\mathbf{Y}\,M$ is a fixed point of M, that is

$$M\,(\mathbf{Y}\,M) =_\beta \mathbf{Y}\,M$$

- As a consequence, a question like "Is there a P such that $P\,x =_\beta x\,P\,x\,P$ (for all x)?" can be answered affirmatively:

$$\begin{array}{lll}
P\,x & =_\beta & x\,P\,x\,P \\
 & \Uparrow & \\
P & =_\beta & \lambda x.x\,P\,x\,P \\
 & \Uparrow & \\
P & =_\beta & (\lambda p\,x.x\,p\,x\,p)P \\
 & \Uparrow & \\
P & \text{is a} & \text{fixed point of } \lambda p\,x.x\,p\,x\,p \\
 & \Uparrow & \\
P & := & \mathbf{Y}(\lambda p.\lambda x.x\,p\,x\,p)
\end{array}$$

Representing data in λ-calculus

Booleans

$$\begin{aligned} \textbf{true} &:= \lambda x\, y.\, x \\ \textbf{false} &:= \lambda x\, y.\, y \\ \textbf{if } M \textbf{ then } P \textbf{ else } Q &:= M\, P\, Q \end{aligned}$$

Natural Numbers via the so-called Church Numerals

$$\begin{aligned} c_0 &:= \lambda f\, x.x \\ c_1 &:= \lambda f\, x.f\, x \\ c_2 &:= \lambda f\, x.f(f\, x) \\ \ldots & \\ c_n &:= \lambda f\, x.f^n\, x \end{aligned}$$

$f^n\, x$ is an n-times application of f on x: $\underbrace{f(f \ldots (f}_{n\times}\ x) \ldots)$

Then, e.g.

$$\begin{aligned} \textbf{Succ} &:= \lambda n\, f\, x.f(n\, f\, x) \\ \textbf{Zero} &:= \lambda n.n\,(\lambda y.\, \textbf{false}\,)\ \textbf{true} \end{aligned}$$

Natural deduction for Proposition logic

The formulas are given by

$$\varphi ::= \mathrm{At} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \mid \neg\varphi.$$

The derivation rules define how to derive a formula φ from a set of formulas Γ, notation

$$\Gamma \vdash \varphi$$

Derivation rules

$$\frac{\varphi \in \Gamma}{\Gamma \vdash \varphi}\ \mathsf{ax} \qquad \frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}\ \to_i \qquad \frac{\Gamma \vdash \varphi \to \psi \quad \Gamma \vdash \varphi}{\Gamma \vdash \psi}\ \to_e$$

$$\frac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi}\ \vee_{i1} \qquad \frac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \vee \psi}\ \vee_{i2} \qquad \frac{\Gamma \vdash \varphi \vee \psi \quad \Gamma, \varphi \vdash \chi \quad \Gamma, \psi \vdash \chi}{\Gamma \vdash \chi}\ \vee_e$$

$$\frac{\Gamma \vdash \varphi \quad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}\ \wedge_i \qquad \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \varphi}\ \wedge_{e1} \qquad \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \psi}\ \wedge_{e2}$$

$$\frac{\Gamma, \varphi \vdash \psi \quad \Gamma, \varphi \vdash \neg\psi}{\Gamma \vdash \neg\varphi}\ \neg_i \qquad \frac{\Gamma \vdash \neg\varphi \quad \Gamma \vdash \varphi}{\Gamma \vdash \psi}\ \neg_e \qquad \frac{\Gamma \vdash \neg\neg\varphi}{\Gamma \vdash \varphi}\ \neg_{\text{class}}$$

Derivation trees with Γ as leaves

Just the $\to$ and $\vee$ cases:

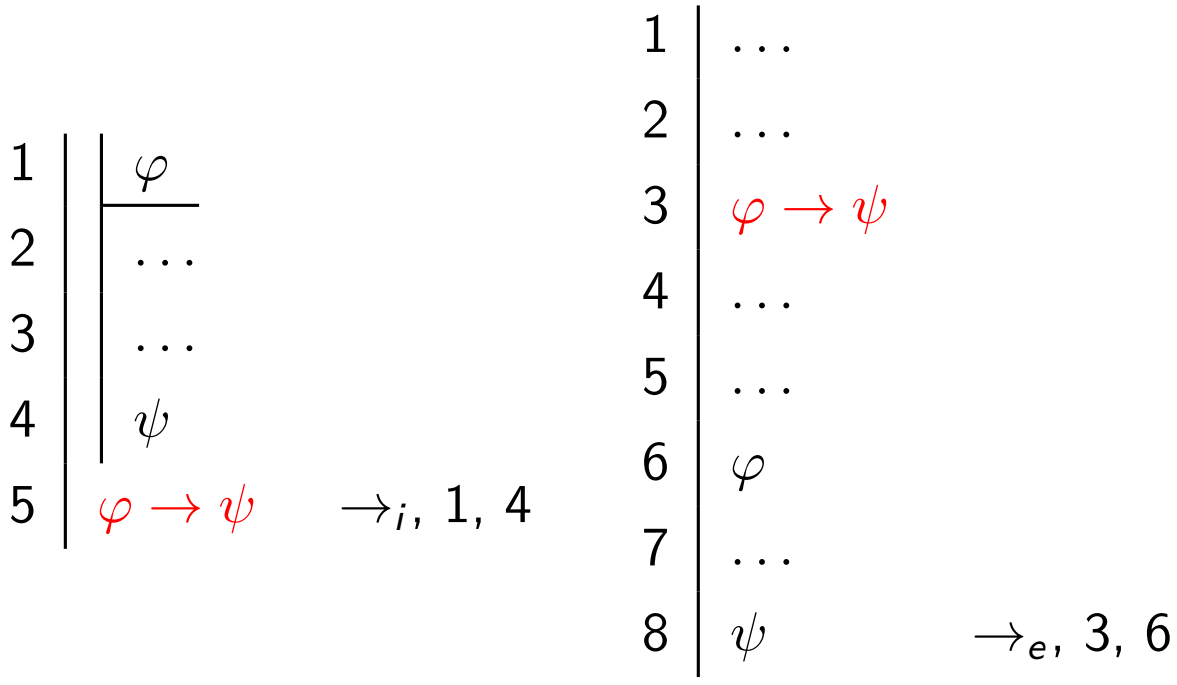
$$\dfrac{\begin{array}{c}[\varphi]^{\ell}\\ \vdots\\ \psi\end{array}}{\varphi \to \psi}\ \to_i^{\ell} \qquad \dfrac{\begin{array}{cc}\vdots & \vdots\\ \varphi \to \psi & \varphi\end{array}}{\psi}\ \to_e$$

$$\dfrac{\begin{array}{c}\vdots\\ \varphi\end{array}}{\varphi \vee \psi}\ \vee_{i1} \qquad \dfrac{\begin{array}{c}\vdots\\ \psi\end{array}}{\varphi \vee \psi}\ \vee_{i2} \qquad \dfrac{\begin{array}{ccc} & [\varphi]^k & [\psi]^k\\ \vdots & \vdots & \vdots\\ \varphi \vee \psi & \chi & \chi\end{array}}{\chi}\ \vee_e^k$$

The k and ℓ are labels to indicate which assumptions are being discharged at which rule application.

Flag style deductions

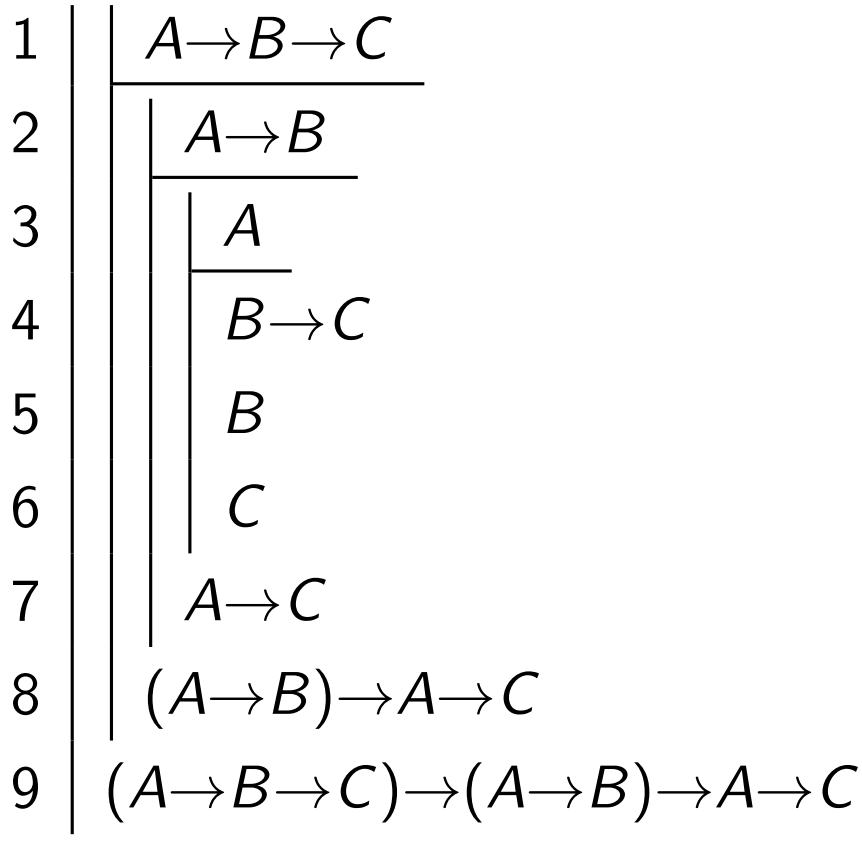
Also known as Fitch style natural deduction: linear derivations with flags to delimit scope.



A rule application refers to line numbers to make explicit which hypotheses are used in the rule.

Example

Convention: $A \to B \to C$ denotes $A \to (B \to C)$ etc.



Typed λ calculus as the basis for a Proof Assistant

Typed λ calculus forms the basis for a variety of proof Assistants, e.g. Rocq (and Lean, Agda, Nuprl, Matita).

λ-term	type
program	specification
proof	formula

Integrated system for proving and programming

Types are not sets

Types are a bit like sets, but types give syntactic information, e.g.

$$3 + (7 \times 8)^5 \ : \ \mathsf{nat}$$

whereas sets give semantic information, e.g.

$$3 \ \in \ \{n \in \mathbb{N} \mid \forall x, y, z \in \mathbb{N}^+ (x^n + y^n \neq z^n)\}.$$

- $3 + (7 \times 8)^5$ is of type nat because $3, 7, 8$ are natural numbers and $\times$, $+$ and power are operations on natural numbers.
- $3 \in \{n \in \mathbb{N} \mid \forall x, y, z \in \mathbb{N}^+ (x^n + y^n \neq z^n)\}$ because there are no positive x, y, z such that $x^3 + y^3 = z^3$, which is an instance of Fermat's last Theorem, proved by Wiles.
- To establish that 3 is an element of the given set, we need a proof, we can't just read it off from the components of the statement.
- To establish $3 + (7 \times 8)^5 : \mathsf{nat}$ we don't need a proof but a simple computation (the "reading the type of of the term").

Decidability of :, undecidability of $\in$

- Membership is undecidable in set theory, as it requires a proof to establish $a \in A$.
- Type checking is decidable: Verifying whether M is of type A requires purely syntactic methods, which can be cast into a typing algorithm.

$$3 + (7 \times 8)^5 : \mathsf{nat} \qquad \text{versus} \qquad \frac{1}{2} \sum_{n=0}^{\infty} 2^{-n} \in \mathbb{N}$$

Formulas-as-Types embedding

Also called the Curry-Howard embedding or Curry-Howard isomorphism.

- Often it is not an isomorphism,
- Some of the original ideas stem from De Bruijn,
- The crucial part is more "proof-as-terms" then "formulas-as-tyes", so Curry-Howard-De-Bruijn proofs-as-terms embedding might be a better name...but we won't use that

Question: Can we turn (e.g.)

$$\{n \in \mathbb{N} \mid \forall x, y, z \in \mathbb{N}^{+}(x^n + y^n \neq z^n)\}$$

into a (syntactic) type, with decidable type checking?

Phrased differently: can we talk about this set as a "subtype of nat"?

Formulas are also types; proofs are terms

$$\{n \in \mathsf{nat} \mid \forall x, y, z \in \mathbb{N}^+ (x^n + y^n \neq z^n)\}$$

is a type.
Its terms are pairs $\langle n, p \rangle$ where

- n : nat
- p : $\forall x, y, z \in \mathbb{N}^+ (x^n + y^n \neq z^n)$

So p is a proof, and we view the formula
$\forall x, y, z \in \mathbb{N}^+ (x^n + y^n \neq z^n)$ as the type of its proofs.

If we have decidable proof checking, then it is decidable whether a given pair $\langle n, p \rangle$ is typable with the above type or not.

We summarize:

- proof checking = type checking,
- type checking is decidable (so proof checking is decidable),
- proof finding is not decidable (proof finding is required to check an $\in$-judgment).

Formulas-as-types; proofs-as-terms

In the next lectures, we will make formulas-as-types and proofs-as-terms precise by illustrating it on various logics and type theories.

Questions?