Introduction to Type Theory — Herman Geuvers

Lecture 1a: Polymorphic Type Theory
OPLSS 2026

1 Untyped lambda-calculus

λ-abstraction is a notation for functions. A term such as x^2+2 is an expression with a free variable, while $\lambda x.\, x^2 + 2$ is a function that binds that variable. Application is written by juxtaposition, associates to the left, and supports currying: $g\ 3\ 4$ means $(g\ 3)\ 4$.

The untyped calculus is generated by variables, applications, and abstractions:

$$\Lambda ::= Var \mid (\Lambda\ \Lambda) \mid (\lambda Var.\Lambda).$$

Standard combinators such as

$$I = \lambda x.x, \quad K = \lambda xy.x, \quad S = \lambda xyz.xz(yz), \quad \omega = \lambda x.xx, \quad \Omega = \omega\omega$$

illustrate identity, projection, application structure, self-application, and nontermination.

Computation is β-reduction:

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

The substitution operation must avoid variable capture. Before substituting P for x in M, one renames bound variables in M that also occur free in P. This is why $(\lambda xy.yx)y$ does not reduce naively to $\lambda y.yy$.

The α-equivalence examples have the following answers:
$\lambda x.\lambda y.xy \equiv_\alpha \lambda y.\lambda x.yx$ is yes;
$\lambda x.\lambda y.xy \equiv_\alpha \lambda x.\lambda y.yx$ is no;
$\lambda x.\lambda y.xy \equiv_\alpha \lambda x.\lambda y.yy$ is no;
$\lambda x.\lambda x.xx \equiv_\alpha \lambda x.\lambda y.yy$ is yes.

A calculus is said to be inconsistent if it is possible to derive $M =_\beta P$ for every term M and P, just like an inconsistent logic can derive every formula (including false). The lambda-calculus is in fact consistent, and we can prove this using the fact that it satisfies the Church-Rosser property.

The Church-rosser property states that if $M =_\beta N$ then there exists a term P such that $M \to_\beta^* P$ and $N \to_\beta^* P$. If the lambda-calculus was inconsistent, then we could derive $K =_\beta I$. Then, there

would be a term N such that $K \to_\beta^* N$ and $I \to_\beta^* P$, but this is impossible since K and I are normal forms and different. Thus, lambda-calculus is consistent.

Untyped lambda-calculus is nevertheless Turing complete because fixed-point combinators solve recursive equations. With

$$Y := \lambda f.(\lambda x.f(xx))(\lambda x.f(xx)),$$

we have $Yf =_\beta f(Yf)$, so recursive definitions can be expressed without a built-in recursion construct.

2 Data as functions

Church encodings represent data by their eliminators. Booleans select a branch:

$$true := \lambda xy.x, \qquad false := \lambda xy.y, \qquad if\ M\ then\ P\ else\ Q := MPQ.$$

Church numerals encode n as "apply f to x exactly n times":

$$c_0 = \lambda fx.x, \quad c_1 = \lambda fx.fx, \quad c_2 = \lambda fx.f(fx), \quad c_n = \lambda fx.f^n x.$$

The successor and zero-test functions can then be represented internally, for example $Succ := \lambda nfx.f(nfx)$.

3 Natural deduction

The propositional formulas are generated from atoms using $\wedge$, $\vee$, $\to$, and $\neg$. Natural deduction presents derivability $\Gamma \vdash \varphi$ by introduction and elimination rules. Tree-style proofs mark discharged assumptions by labels; Fitch-style proofs use flags to delimit the scope of temporary assumptions.

For implication, the introduction rule assumes φ, derives ψ, and discharges the assumption to derive $\varphi \to \psi$. The elimination rule is modus ponens. The example

$$(A \to B \to C) \to (A \to B) \to A \to C$$

is proved by assuming $A \to B \to C$, $A \to B$, and A, deriving $B \to C$ and B, then C, and discharging the assumptions in reverse order.

Double-negation elimination belongs to classical logic, not intuitionistic logic. As the $\neg class$ rule, it is deliberately not part of minimal or intuitionistic proof theory.

4 Formulas as types

Typed lambda-calculus supports the proof-assistant view of formulas-as-types:

λ-term	*type*
program	*specification*
proof	*formula*

Types are syntactic classifiers, not sets. Checking that $3 + (7 \times 8)^5$ has type *nat* is a syntactic computation; checking membership in a set such as a Fermat-style subset of $\mathbb{N}$ requires a proof. Thus proof checking can be decidable even when proof finding is not.

Under the Curry-Howard-De Bruijn reading, formulas become types and proofs become terms. For a subtype-like expression

$$\{n : nat \mid \forall x, y, z \in \mathbb{N}^+ . x^n + y^n \neq z^n\},$$

a term is a pair $\langle n, p \rangle$ where $n : nat$ and p is a proof of the predicate at n. Thus, proof-checking is equivalent to type-checking, which is decidable. Proof-searching, on the other hand, corresponds to type inhabitation and is decidable for the simply typed lambda-calculus. However, it is very hard (PSPACE-complete). For second-order logic proof-searching even becomes undecidable. Later, we will see that this corresponds to the type inhabitation problem for System F being undecidable.

5 Examples of λ-abstraction

$$f := \lambda x.\, (x^2 + 2) \qquad g := \lambda x.\, (\lambda y.\, (x^2 + y + 2))$$

A λ-abstraction is a term with a variable. We apply functions:

$$f\, 3 \qquad x$$

The forms $f\, 3$, $(f\, 3)$, $(f \cdot 3)$ are the same; application is a binary operator.

$$(g\, 3)\, 4 \qquad \big(\lambda x.\, (x^2 + 4 + 2)\big)\, 3 \;=\; 3^2 + 4 + 2 \;=\; 15$$

$$g\, 3\, 4 \quad \text{associated to the left.}$$

6 Backus–Naur Form

$$A \;::=\; \text{Var} \;\mid\; (AA) \;\mid\; (\lambda \text{Var}.\, A)$$

$$(M\, N)\, P \quad \checkmark \qquad\qquad M\, (N\, P) \quad \times$$

$$\lambda x\, y\, z.\, M \qquad \equiv \qquad \lambda x.\, (\lambda y.\, (\lambda z.\, M))$$

7 Combinators

$$I = \lambda x.\, x \qquad K = \lambda x.\, \lambda y.\, x \qquad S = \lambda x\, y\, z.\, x\, z\, (y\, z)$$

(S is a combinator.)

$$\omega := \lambda x.\, x\, x \qquad \Omega := \omega\, \omega$$

8 Redex and β-reduction

A **redex** (reducible expression):

$$\underbrace{(\lambda x.\, x^2 + 2)\, 3}_{\text{redex}} \;\to_\beta\; 3^2 + 2$$

$=_\beta$ is the equivalence relation generated from the β-reduction rule.

$$(\lambda x.\, M)\, P \;\to_\beta\; M[P/x] \qquad (\text{also written } M[x := P])$$

9 Compatible closure of β-reduction

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

$$I\, P \to_\beta P \qquad K\, P\, Q \to \cdots \to_\beta P \qquad \Omega \to_\beta \Omega$$

α-equality

$$\lambda x.\, M \;\equiv\; \lambda y.\, M[x := y] \quad (\text{modulo } \alpha\text{-equality})$$

α-equal: $M \equiv N$, written $M =_\alpha N$.

$\twoheadrightarrow_\beta$ is the transitive reflexive closure of $\to_\beta$.

Proof strategy

Assume they are equal and show on β-reduction they reduce to different terms.

$$\begin{array}{ccc} M & =_\alpha & N \\ M\,P\,Q & =_\alpha & N\,P\,Q \\ \downarrow\beta & & \downarrow\beta \\ I & & K \end{array}$$

Then $M =_\alpha N$.

Church–Rosser property

If $M =_\beta N$ then there is some Q with

$$M \twoheadrightarrow_\beta Q \quad \text{and} \quad N \twoheadrightarrow_\beta Q$$

(they have a common redex).

10 Simply Typed Lambda Calculus ($\lambda^{\to}$)

$$\text{Typ} ::= \alpha \mid \beta \mid \alpha \to \beta \mid \beta \to \alpha \qquad \text{Tvar} \mid \text{Typ} \to \text{Typ} \qquad \Gamma \vdash M : A$$

Typing rules

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\,N : B} \qquad \frac{\Gamma, x : A \vdash P : B}{\Gamma \vdash \lambda x{:}A.\,P : A \to B}$$

(Type checking is PSPACE-complete. Dependent type checking needs to terminate.)

Inferring types

$$\frac{\Gamma \vdash M : ?_0 \to ?_1 \qquad \Gamma \vdash N : ?}{\Gamma \vdash M\,N : ?_1} \qquad \frac{\Gamma \vdash P : A \qquad \Gamma \vdash Q : B}{\Gamma \vdash \langle P, Q\rangle : A \times B}$$

(Conformant means it satisfies the Church–Rosser property.)

η-reduction

$$\lambda x.\, M\, x \;\to_\eta\; M \qquad (\text{if } x \notin FV(M))$$

$$(\lambda x.\, M)\, P = M\, P$$

(As functions, they do the same.)

11 Combinator definitions and proof systems

$$M\, x \;=_\beta\; x\, M\, x\, ? \qquad \text{factorial function}$$

Fixed-point (Y) combinator

$$Y := \lambda f.\, (\lambda x.\, f\, (x\, x))\, (\lambda x.\, f\, (x\, x))$$
$$Y\, f \;=_\beta\; f\, (Y\, f)$$

Church numerals

$$c_0 := \lambda f.\, \lambda x.\, x \qquad c_1 := \lambda f\, x.\, f\, x \qquad \cdots \qquad c_n := \lambda f.\, \lambda x.\, f^{\,n}\, x$$
$$\lambda n\, f\, x.\, f\, (n\, f\, x)$$

A propositional logic

$$\varphi \;::=\; \text{Atoms} \;\mid\; \varphi \wedge \varphi \;\mid\; \varphi \vee \varphi \;\mid\; \varphi \to \varphi \;\mid\; \neg\varphi$$

$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}\,(i) \qquad\qquad \frac{\Gamma \vdash \varphi \to \psi \qquad \Gamma \vdash \varphi}{\Gamma \vdash \psi}\,(e)$$

Exercises

$$3 + (7 \times 8)^5 : \text{Nat}$$
$$3 \in \{\, n \in \mathbb{N} \;\mid\; \forall x, y, z \in \mathbb{N}^+.\, (x^n + y^n \neq z^n) \,\}$$

Write your own type checker; write your own proof systems; get an answer.

(If an answer out of λ-calculus is received it will be unique, if the λ-term terminates.)

References

[1] A. Church and J. B. Rosser. *Some properties of conversion.* Transactions of the American Mathematical Society, 1936.

[2] R. Milner. *A Theory of Type Polymorphism in Programming.* Journal of Computer and System Sciences, 1978.