

## Today's agenda

## Typed LC

Syntax of Pure Lambda calculus:

$M ::= x$	variables	term	
$(\lambda x. M)$	abstraction	term	
$(M \ N)$	application	term	$(M_1 \ M_2)$ is same as $(M \ N)$

There is no type in pure LC, so it is also referred to as untyped LC.

Now we associate types with arguments and return type. LC with types is called typed LC.

## Types of the common functions:

A function is a mapping from one set to another, e.g.,  $f : B \rightarrow C$

$M : A$  to be read has term  $M$  has type  $A$

- Identity function:**  $I = \lambda x. x$  untyped version of identity function  
 $\text{Let } x : A,$   
 $I = \lambda x:A. x$  typed version of identity function

What is the type of this function?

$I : A \rightarrow A$  arrow means function type  
 $\text{Let } A = \text{int}$   $I : \text{int} \rightarrow \text{int}$

- First:**  $\lambda x. \lambda y. x$   
 $\text{Let } x : A, y : B,$   
 $\text{First} : \lambda x:A. \lambda y:B. x$   
 $\text{First} : A \rightarrow B \rightarrow A$

$\rightarrow$  is right associative

$A \rightarrow B \rightarrow A$  is equivalent to  $A \rightarrow (B \rightarrow A)$

$A \rightarrow (B \rightarrow A)$  to be read as: a function type that takes as input an argument of type  $A$  and returns another function that takes an argument of type  $B$  and returns an output of type  $A$ .

This is equivalent to saying that the function takes two arguments of types  $A$  and  $B$  and returns an output of type  $B$ .

Note:  $\rightarrow$  is right associative but function association is left associative  $[f \ g \ h = (f \ g) \ h]$

$$\lambda x. \lambda y. y$$

Second :  $A \rightarrow B \rightarrow B$

$$\lambda f. \lambda x. f x$$

Apply:  $(A \rightarrow B) \rightarrow A \rightarrow B$

all the types can be produced by using compositions of these only.

Let  $Basic = \{int\}$

Then we may obtain the types  $int$ ,  $int \rightarrow int$ ,  $int \rightarrow int \rightarrow int$ , and so on. There are infinitely many types that can be obtained. (recall that  $\rightarrow$  associates from right)

Let  $Basic = \{int, bool\}$ .

Then we may obtain the types  $int$ ,  $bool$ ,  $int \rightarrow int$ ,  $bool \rightarrow bool$ ,  $int \rightarrow int \rightarrow int$ , and so on

Syntax of simply typed lambda terms:

$t ::=$		terms
	$x$	variable
	$(\lambda x : \sigma. t)$	typed abstraction $\sigma \in \lambda^{\rightarrow}$
	$(t \ t)$	application

Eg,  $\lambda x : int. x$  is a simply typed lambda term, and the type is

$(\lambda x : int. x) : int \rightarrow int$  which can be generalized as

$(\lambda x : \sigma. x) : \sigma \rightarrow \sigma$  which is of the form

$M : \tau$  where  $M$  is a simply typed lambda term,  $\tau$  is the type of the term

We say that  $M$  has type  $\tau$  (has-type relation :).

### Interpretation of types

Each type expression is interpreted as a set, the set of values of that type.

e.g., the type  $int$  denotes the set of all integers

$\sigma \rightarrow \tau$  is the type or set of all functions from  $\sigma$  to  $\tau$ .

So if  $int$  is a set of all integers, then

$int \rightarrow int$  denotes the set of all functions with domain  $int$  and range  $int$ .

End of lecture

Sec 18-19.

27-28.3.2025

### Definition 1 Types:

- (i) A type variable is a type (basic/atomic type)
- (ii) If  $\sigma$  and  $\tau$  are types then  $(\sigma \rightarrow \tau)$  is a type (composite type)

### Type assignment theory $TA_\lambda$

**Definition 2:** A **type assignment** is any expression  $M : \tau$  where  $M$  is a  $\lambda$ -term and  $\tau$  is a type.  
 $M : \tau$  (means)  $M$  has type  $\tau$  or assign to  $M$  the type  $\tau$

**Definition 3:** A **type context**  $\Gamma$  is any finite, perhaps empty, set of type-assignments

$\Gamma = \{x_1 : \rho_1, \dots, x_n : \rho_n\}$  such that no  $x_i$  has more than one assignment

Such a set is called **consistent**.

$\text{Subjects}(\Gamma) = \{x_1, \dots, x_n\}$

**Notation:** the result of removing an assignment with term variable  $x$  is denoted by  $\Gamma - x$ .

**Definition 4:**  $\Gamma_1$  is consistent with  $\Gamma_2$  iff  $\Gamma_1 \cup \Gamma_2$  is consistent.

**Definition 5:**  $TA_\lambda$ -formulae. For any  $\Gamma$ ,  $M$ , and  $\tau$  the triple  $\langle \Gamma, M, \tau \rangle$  is called a  $TA_\lambda$ -formula and is written as  $\Gamma \vdash M : \tau$  in the context of gamma,  $M$  has the given type.

**Notation:** in the context portion, a comma  $(,)$  denotes set union; the set symbol may be omitted.

$\Gamma, x$  is an abbreviation of  $\Gamma \cup \{x : \sigma\}$   
 $x : \sigma$  is an abbreviation of  $\{x : \sigma\}$

**Notation and meaning of a deduction rule:**

$$\frac{\text{Premise}_1 \dots \text{Premise}_k}{\text{Conclusion}} \quad (\text{deduction rule})$$

If all the Premises are true then the Conclusion is also true. If a Premise is an axiom then it is true; otherwise apply the rules to prove that it is true.

### Definition 6: the system $TA_\lambda$

$TA_\lambda$  has an infinite set of axioms and two deduction rules ( $\rightarrow$ E or  $\rightarrow$  Elimination) and ( $\rightarrow$ I or  $\rightarrow$  Introduction)

**Axiom:** for every term variable  $x$  and every type  $\tau$ ,  $x : \tau \vdash x : \tau$

this is actually modes ponens

Deduction rules:

$$\frac{\Gamma_1 \mapsto P : (\sigma \rightarrow \tau) \quad \Gamma_2 \mapsto Q : \sigma}{\Gamma_1 \cup \Gamma_2 \mapsto (P Q) : \tau} \quad \text{if } \Gamma_1 \cup \Gamma_2 \text{ is consistent} \quad ( \rightarrow E )$$

the tree formed by these rules is called deduction or proof tree.

$$\frac{\Gamma \mapsto P : \tau}{\Gamma - x \mapsto (\lambda x. P) : (\sigma \rightarrow \tau)} \quad \text{if } \Gamma \text{ is consistent with } x : \sigma \quad ( \rightarrow I )$$

$$\frac{\Gamma_1, x : \sigma \mapsto P : \tau}{\Gamma_1 \mapsto (\lambda x. P) : (\sigma \rightarrow \tau)} \quad \text{if } x \notin \text{Subjects}(\Gamma_1) \quad ( \rightarrow I )\text{-main}$$

$$\frac{\Gamma_1 \mapsto P : \tau}{\Gamma_1 \mapsto (\lambda x. P) : (\sigma \rightarrow \tau)} \quad \text{if } x \notin \text{Subjects}(\Gamma_1) \quad ( \rightarrow I )\text{-vac}$$

A  $TA_\lambda$  deduction  $\Delta$  is a tree of  $TA_\lambda$ -formulae, those at the tops of branches being axioms and those below being deduced from those immediately above them by a rule.

If  $\Gamma \mapsto M : \tau$  we call  $\Delta$  a deduction of  $\Gamma \mapsto M : \tau$  and say that  $\Gamma \mapsto M : \tau$  is  $TA_\lambda$ -deductible.

When  $\Gamma = \emptyset$ ,  $\Delta$  may be called a proof of the assignment  $M : \tau$ .

If  $\Gamma \mapsto M : \tau$  is  $TA_\lambda$ -deductible then  $\text{Subjects}(\Gamma) = \text{FV}(M)$ .

The construction of the proof tree is bottom up. At the bottom is the conclusion and the top are the axioms. The height of a proof tree is no more than the number of occurrences of variables in a  $\lambda$ -term.  
 $\text{length}(x) = 1$ ,  $\text{length}(M N) = \text{length}(M) + \text{length}(N)$ ,  $\text{length}(\lambda x. M) = 1 + \text{length}(M)$ .

1. Give a deduction of  $\mapsto id : a \rightarrow a$  is same as Prove that  $id : a \rightarrow a$

First, we observe that rule  $(\rightarrow I)$  has to be applied. We will examine  $(\rightarrow I)$ -main.

$P = x, \tau = a, x = x, \sigma = a, \Gamma_1 = \emptyset$ . So, we obtain the following where the condition is satisfied.

Now the premise is an axiom, and we are done.

$$\frac{x : a \mapsto x : a \quad (\text{axiom})}{\mapsto (\lambda x. x) : (a \rightarrow a)} \quad ( \rightarrow I )\text{-main}$$



2. Give a deduction of  $\vdash id : (a \rightarrow a) \rightarrow (a \rightarrow a)$

$$\frac{x : (a \rightarrow a) \vdash x : (a \rightarrow a) \quad (\text{axiom})}{\vdash (\lambda x. x) : (a \rightarrow a) \rightarrow (a \rightarrow a)} \quad (\rightarrow I)\text{-main}$$

### Exercise:

1. TPT:  $(\lambda x: a. \lambda y: b. x) : a \rightarrow b \rightarrow a$
2. TPT:  $(\lambda x: a \rightarrow b. \lambda y: c \rightarrow a. \lambda z: c. x (yz)) : (a \rightarrow b) \rightarrow (c \rightarrow a) \rightarrow b$
3. TPT:  $id \ id : a \rightarrow a$

End of lecture

### Tutorial 3

Identify 5  $\lambda$ -terms each of length at least 4. Give a deduction of each term in  $TA_\lambda$ .  
This constitutes 5 problems.

Date of submission 7.4.2025 11am to 12:30pm