Lecture 17 **24.3.2025**

Today's agenda

Typed LC

Syntax of Pure Lambda calculus:

M ::= x	variables	term	
| (λx. 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

1. **Identity function:** $I = \lambda x.\ x$ untyped version of identity function

   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

   Let A= int $I : int \rightarrow int$

2. **First:** $\lambda x.\ \lambda y.\ x$

   Let x : A, y : B,

   First: $\lambda x{:}A.\ \lambda y{:}B.\ x$

   First : $A \rightarrow B \rightarrow A$

   → 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: → is right associative but function association is left associative [f g h = (f g) h]

3. **Second:** $\lambda x.\ \lambda y.\ y$

Second: $\lambda x{:}A.\ \lambda y{:}B.\ y$

Second : $A \to B \to B$

4. **Apply**: $\lambda f.\ \lambda x.\ f\ x$

Let x : A,

what would be the type of f?

(the input of f is x, let the output be of type B)

So f has type $A \to B$ $\quad f : A \to B$

Apply: $(A \to B) \to A \to B$ example of HOF

This says that apply takes two arguments: one is a function, another variable.

We can see that the types of f and x must be different, since f is an application, x is simply a variable. Thus we get the above one.

Why is the parenthesis needed? Otherwise we get $A \to B \to (A \to B)$

This could mean that the type of first argument is A, second is B, and return type is $A \to B$ which is not intended. $\to$ is right associative

5. **Twice**: $\lambda f.\ \lambda x.\ f\ (f\ x)$

Let x : A,

So f x say has type B

Then f takes input of type B. But it cannot take different types in the same definition. So B must be A.

Twice :$(A \to A) \to A \to A$ HOF

6. **Comp** = $\lambda f.\ \lambda g.\ \lambda x.\ g\ (f\ x)$

$x : A \quad f : A \to B \quad g : B \to C$

Comp : $(A \to B) \to (B \to C) \to A \to C$ HOF

Simply typed LC

There are two forms of simply typed LC—Church style and Curry style. We shall follow the **Curry style**. For the examples above, say, $x : A$, $f : A \to B$, ... we need a language of the type expressions highlighted. We consider a language $\lambda^{\to}$ that allows basic types and arrow types only.

The language (simply typed LC) of type expressions is denoted by $\lambda^{\to}$

$\sigma ::=$ types

K $\quad K \in Basic$ basic/atomic type

$\sigma \to \tau$ composite type

Let *Basic* = {int}
Then we may obtain the types int, int→int, int→int→int, and so on. There are infinitely many types that can be obtained. (recall that → associates from right)

Let *Basic* = {int, bool}.
Then we may obtain the types int, bool, int→int, bool→bool, int→int→int, and so on

Syntax of simply typed lambda terms:

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

Eg, λx:int. x is a simply typed lambda term, and the type is
(λx: int. x) : int → int which can be generalized as
(λx:σ. x) : σ → σ which is of the form
M : τ where M is a simply typed lambda term, τ is the type of the term

We say that M has type τ (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
σ→τ is the type or set of *all* functions from σ to τ .
So if *int* is a set of all integers, then
int → *int* denotes the set of all functions with domain *int* and range *int*.

End of lecture

Lec 18-19 .

27-28.3.2025

Definition 1 Types:

(i) A type variable is a type (basic/atomic type)

(ii) If σ and τ are types then (σ -> τ) is a type (composite type)

Type assignment theory TA_λ

Definition 2: A **type assignment** is any expression $M : \tau$ where M is a λ-term and τ is a type.

$M : \tau$ (means) M has type τ or assign to M the type τ

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

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

Such a set is called **consistent**.

Subjects(Γ) = $\{ x_1, \cdots, x_n \}$

Notation: the result of removing an assignment with term variable x is denoted by Γ - x.

Definition 4: Γ_1 is consistent with Γ_2 iff $\Gamma_1 \cup \Gamma_2$ is consistent.

Definition 5: TA_λ-formulae. For any Γ, M, and τ the triple $\langle \Gamma, M, \tau \rangle$ is called a TA_λ-formula and is written as $\Gamma \mapsto M : \tau$

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

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

Notation and meaning of a deduction rule:

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

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_λ

TA_λ has an infinite set of axioms and two deduction rules (-->E or -- > Elimination) and (-- > I or -- > Introduction)

Axiom: for every term variable x and every type τ, $x : \tau \mapsto x : \tau$

Deduction rules:

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

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

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

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

A TA_λ deduction Δ is a tree of TA_λ-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 Δ a deduction of $\Gamma \mapsto M : \tau$ and say that $\Gamma \mapsto M : \tau$ is TA_λ-deductible.
When $\Gamma = \emptyset$, Δ may be called a proof of the assignment $M : \tau$.

If $\Gamma \mapsto M : \tau$ is TA_λ-deductible then Subjects(Γ) = 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 λ-term. length (x) = 1, length (M N) = length (M) + length(N), length (λx. M) = 1 + length(M).

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

   First, we observe that rule (-- > I) has to be applied. We will examine -- > 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.

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

2. Give a deduction of $\mapsto id : (a \to a) \to (a \to a)$

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

Exercise:

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

End of lecture

Tutorial 3

Identify 5 λ-terms each of length at least 4. Give a deduction of each term in TA_λ.
This constitutes 5 problems.

Date of submission 7.4.2025 11am to 12:30pm