编程语言的设计原理

Design Principles of Programming Languages

2c4r

Haiyan Zhao，Di Wang

赵海燕，王迪

Peking University, Spring Term 2025

Recap: untyped lambda-calculus

Syntax

$\mathtt{t}$ ::=		*terms:*
	$\mathtt{x}$	*variable*
	$\lambda\mathtt{x.t}$	*abstraction*
	$\mathtt{t\ t}$	*application*
$\mathtt{v}$ ::=		*values:*
	$\lambda\mathtt{x.t}$	*abstraction value*

Evaluation $\boxed{\mathtt{t} \longrightarrow \mathtt{t}'}$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2} \quad \text{(E-APP1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1\ \mathtt{t}_2'} \quad \text{(E-APP2)}$$

$$(\lambda\mathtt{x.t}_{12})\ \mathtt{v}_2 \longrightarrow [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12} \quad \text{(E-APPABS)}$$

- The *λ-calculus* embodies this kind of *function definition* and *application* in the purest possible form
  - terms in the pure *λ-calculus* are often called *λ-terms*
  - terms of the form *λx. t* are called *λ-abstractions* or just abstractions

Syntax

- **Definition** [Terms]:

  Let $\mathcal{V}$ be a *countable set* of variable names.

  The set of terms is *the smallest set* $\mathcal{T}$ such that

  1. $x \in \mathcal{T}$ for every $x \in \mathcal{V}$;
  2. if $t_1 \in \mathcal{T}$ and $x \in \mathcal{V}$, then $\lambda x.t_1 \in \mathcal{T}$;
  3. if $t_1 \in \mathcal{T}$ and $t_2 \in \mathcal{T}$, then $t_1\ t_2 \in \mathcal{T}$.

Syntactic conventions

- The *λ-calculus* provides *only one-argument functions*, all multi-argument functions must be written in curried style.
- The following *conventions* make the linear forms of terms easier to read and write:
  - Application *binds more tightly than abstraction*

    e.g., $\lambda x.\ x\ y$ means $\lambda x.\ (x\ y)$ not $(\lambda x.\ x)\ y$
  - Application *associates to the left*

    e.g., $t\ u\ v$ means $(t\ u)\ v$, not $t\ (u\ v)$
  - Bodies of λ- abstractions *extend as far to the right as possible*

    e.g., $\lambda x.\ \lambda y. x\ y$ means $\lambda x.\ (\lambda y.\ x\ y)$, not $\lambda x.\ (\lambda y.\ x)\ y$

Scope

- *An occurrence* of the variable x is said to be ***bound*** when it occurs in the body t of an abstraction $\lambda \mathrm{x.t}$, i.e.,
  - the λ-abstraction term $\lambda x.\mathrm{t}$ binds the variable x , and the scope of this binding is the body t.
  - λx is a *binder* (binding construct) whose *scope* is t.
    - e.g., $(\lambda \mathrm{x}.(\lambda \mathrm{y.x\ y}))\ \mathrm{y}$
  - a binder can be *renamed* as necessary
    - so-called: *alpha-renaming*
    - e.g., $\lambda x.\mathrm{x} = \lambda \mathrm{y}.\ \mathrm{y}$

Scope

- **Definition:** Free Variables of term t, written as $FV(t)$:

$$FV(x) = \{x\}$$

$$FV(\lambda x.t_1) = FV(t_1) \setminus \{x\}$$

$$FV(t_1\ t_2) = FV(t_1) \cup FV(t_2)$$

Operational Semantics

- Computation rule

$$(\lambda x.t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12} \qquad \text{(E-AppAbs)}$$

- Congruence rules

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2} \qquad \text{(E-App1)}$$

$$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'} \qquad \text{(E-App2)}$$

Operational Semantics

- *Beta-reduction*: the only computation (substitution)

$$(\lambda x.\ t_{12})\ t_2 \longrightarrow [x \mapsto t_2]t_{12},$$

  - the term obtained by *replacing all free occurrences* of x in t_{12} by t_2
  - a *redex* (short for "*reducible expression*") : a term of the form *(λx.t) v* — a *λ-abstraction* applied to a *value*
  - the operation of rewriting a *redex* according to the above rule is called *beta-reduction*
- Examples:

$$(\lambda x.\ x)\ y \rightarrow\ y$$

$$(\lambda x.\ x\ (\lambda x.\ x))\ (u\ r) \rightarrow u\ r\ (\lambda x.\ x)$$

Substitution

$$
\begin{array}{llll}
[x \mapsto s]x & = & s & \\
[x \mapsto s]y & = & y & \text{if } y \neq x \\
[x \mapsto s](\lambda y.t_1) & = & \lambda y.\ [x \mapsto s]t_1 & \text{if } y \neq x \text{ and } y \notin FV(s) \\
[x \mapsto s](t_1\ t_2) & = & [x \mapsto s]t_1\ [x \mapsto s]t_2 &
\end{array}
$$

Alpha-conversion : Terms that *differ only in the names of bound variables* are interchangeable *in all contexts*.

Example:

$$
\begin{aligned}
& [x \mapsto y\ z]\ (\lambda y.\ x\ y) \\
=\ & [x \mapsto y\ z]\ (\lambda w.\ x\ w) \\
=\ & \lambda w.\ y\ z\ w
\end{aligned}
$$

Bound Variables

- Recall that bound variables can be renamed, at any moment, to enable substitution:

$$
\begin{array}{lcll}
[x \mapsto s]x & = & s & \\
[x \mapsto s]y & = & y & \text{if } y \neq x \\
[x \mapsto s](\lambda y.t_1) & = & \lambda y.\ [x \mapsto s]t_1 & \text{if } y \neq x \text{ and } y \notin FV(s) \\
[x \mapsto s](t_1\ t_2) & = & [x \mapsto s]t_1\ [x \mapsto s]t_2 &
\end{array}
$$

- Variable Representation
  - Represent variables symbolically, with variable renaming mechanism
  - Represent variables symbolically, with bound variables are all different
  - “Canonically” represent variables in a way such that renaming is unnecessary
  - No use of variables: combinatory logic

Chapter 6
Nameless Representation of Terms

Terms and Contexts

Shifting and Substitution

Terms and Contexts

Nameless Terms

- *De Bruijn* idea: Replacing named variables by *natural numbers*, where the number k stands for "the variable bound by the $k'th$ enclosing λ". e.g.,
  - λx.x λ.0
  - λx.λy. x (y x) λ.λ. 1 (0 1)

  De Bruijn terms **vs** *De Bruijn indices*

- *e.g.*, the corresponding nameless term for the following:

  c0 = λs. λz. z;

  c2 = λs. λz. s (s z);

  plus = λm. λn. λs. λz. m s (n z s);

  fix = λf. (λx. f (λy. (x x) y)) (λx. f (λy. (x x) y));

  foo = (λx. (λx. x)) (λx. x);

Nameless Terms

- Need to keep track of how many free variables each term may contain.

Definition [Terms]: Let $\mathcal{T}$ be *the smallest family of sets* $\{\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2, \ldots\}$ such that

1. $k \in \mathcal{T}_n$ whenever $0 \leq k < n$;
2. if $t_1 \in \mathcal{T}_n$ and $n>0$, then $\lambda.t_1 \in \mathcal{T}_{n-1}$;
3. if $t_1 \in \mathcal{T}_n$ and $t_2 \in \mathcal{T}_n$, then $(t_1\ t_2) \in \mathcal{T}_n$.

- **Note:**
  - terms with **no free variables** are called the **0-term**s; 1-terms (one **free variables), …**
  - $\mathcal{T}_n$ are set of terms with **at most n** free variables, n-terms, numbered between 0 and n-1: a given element of $\mathcal{T}_n$ need not have free variables with all these numbers, or indeed any free variables at all. When **t** is closed, for example, it will be an element of $\mathcal{T}_n$ for every n.
  - two ordinary terms are *equivalent* modulo renaming of bound variables iff they have the same de Bruijn representation.

Name Context

How to represent

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

as a nameless term?

Here y is free variable.

We know what to do with x, but **we cannot see the binder** for y, so it is *not clear how "far away"* it might be and we do not know *what number* to assign to it.

To deal with these terms containing free variables, we need the idea of a **naming context.**

Name Context

Definition: Suppose x_0 through x_n are variable names from ν. The naming context

$\Gamma = x_n, x_{n-1}, \ldots x_1, x_0$ assigns to each x_i the *de Bruijn index* i.

Note that the *rightmost variable* in the sequence is given the index *0*; this matches the way we count *λ binders* — ***from right to left*** — when converting a named term to nameless form.

Name Context

We write **dom(*Γ*)** for the set $\{x_n, \ldots x_1, x_0\}$ of variable names mentioned in Γ .

- e.g., Γ = x ↦ 4; y ↦ 3; z ↦ 2; a ↦ 1; b ↦ 0 , under this Γ, we have
  - x (y z)   ?
    - 4 (3 2)
  - λw. y w
    - λ. 4 0
  - λw. λa. x
    - λ. λ. 6

Shifting and Substitution

How to define substitution $[k \mapsto s]\ t$?

Shifting

- Under the naming context Γ : $x \mapsto 1, z \mapsto 2$

  $[1 \mapsto 2\ (\lambda.\ 0)]\ \lambda.\ 2 \longrightarrow$?

  i.e., $[\ \mathbf{x} \mapsto \mathbf{z}\ (\boldsymbol{\lambda}\mathbf{w.\ w})\ ]\ \boldsymbol{\lambda}\mathbf{y.\ x} \longrightarrow$?

- When a substitution **goes under a λ-abstraction**, as in $[1 \mapsto s](\lambda.2)$ (i.e.,$[x \mapsto s]\ (\lambda y.\ x)$, assuming that 1 is the index of x in the outer context), *the context* in which the substitution is taking place becomes *one variable longer than the original*.
- We need to *increment the indices* of the *free variables* in s so that they keep referring to *the same names in the new context* as they did before.
  - e.g., $s = 2\ (\lambda.\ 0)$, , i.e., $s = z\ (\lambda w.\ w)$, assuming 2 is the index of z in the outer context, we need to shift the 2 but not the 0

- Shifting is just the auxiliary operation: *renumber the indices of the free variables* in a term.

Shifting

DEFINITION [SHIFTING]: The d-place shift of a term $\mathtt{t}$ above cutoff c, written $\uparrow_c^d(\mathtt{t})$, is defined as follows:

$$
\begin{aligned}
\uparrow_c^d(\mathtt{k}) &= \begin{cases} \mathtt{k} & \text{if } k < c \\ \mathtt{k} + d & \text{if } k \geq c \end{cases} \\
\uparrow_c^d(\lambda.\mathtt{t}_1) &= \lambda.\ \uparrow_{c+1}^d(\mathtt{t}_1) \\
\uparrow_c^d(\mathtt{t}_1\ \mathtt{t}_2) &= \uparrow_c^d(\mathtt{t}_1)\ \uparrow_c^d(\mathtt{t}_2)
\end{aligned}
$$

We write $\uparrow^d(\mathtt{t})$ for $\uparrow_0^d(\mathtt{t})$. □

1. What is $\uparrow^2(\lambda.\lambda.\ \mathtt{1}\ (\mathtt{0}\ \mathtt{2}))$?
2. What is $\uparrow^2(\lambda.\ \mathtt{0}\ \mathtt{1}\ (\lambda.\ \mathtt{0}\ \mathtt{1}\ \mathtt{2}))$?

Substitution

DEFINITION [SUBSTITUTION]: The substitution of a term $\mathtt{s}$ for variable number $\mathtt{j}$ in a term $\mathtt{t}$, written $[\mathtt{j} \mapsto \mathtt{s}]\mathtt{t}$, is defined as follows:

$$
\begin{array}{lcl}
[\mathtt{j} \mapsto \mathtt{s}]\mathtt{k} & = & \begin{cases} \mathtt{s} & \text{if } \mathtt{k} = \mathtt{j} \\ \mathtt{k} & \text{otherwise} \end{cases} \\
[\mathtt{j} \mapsto \mathtt{s}](\lambda.\mathtt{t}_1) & = & \lambda.\,[\mathtt{j}{+}1 \mapsto \uparrow^1(\mathtt{s})]\mathtt{t}_1 \\
[\mathtt{j} \mapsto \mathtt{s}](\mathtt{t}_1\ \mathtt{t}_2) & = & ([\mathtt{j} \mapsto \mathtt{s}]\mathtt{t}_1\ [\mathtt{j} \mapsto \mathtt{s}]\mathtt{t}_2)
\end{array}
$$

□

$$
\begin{array}{lcll}
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{x} & = & \mathtt{s} & \\
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{y} & = & \mathtt{y} & \text{if } \mathtt{y} \neq \mathtt{x} \\
[\mathtt{x} \mapsto \mathtt{s}](\lambda\mathtt{y}.\mathtt{t}_1) & = & \lambda\mathtt{y}.\ [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1 & \text{if } \mathtt{y} \neq \mathtt{x} \text{ and } \mathtt{y} \notin FV(\mathtt{s}) \\
[\mathtt{x} \mapsto \mathtt{s}](\mathtt{t}_1\ \mathtt{t}_2) & = & [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1\ [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_2 &
\end{array}
$$

Evaluation

- To define the *evaluation relation* on nameless terms, the only thing we *need to change* (i.e., the only place where *variable names* are mentioned) is the *beta-reduction rule* *(computation rules),* while keep the other rules identical to what as Figure 5-3.

$$(\lambda x.\, t_{12})\ t_2 \longrightarrow [x \mapsto t_2]t_{12},$$

- How to change the above rule for nameless representation?

Evaluation

- Example:

$$(\lambda x.\ t_{12})\ t_2 \longrightarrow [x \mapsto t_2]t_{12},$$

$$(\lambda.t_{12})\ v_2 \longrightarrow \uparrow^{-1}([0 \mapsto \uparrow^{1}(v_2)]t_{12})$$

$$(\lambda.1\ 0\ 2)\ (\lambda.0) \longrightarrow 0\ (\lambda.0)\ 1$$

Homework

- Read Chapter 6.
  – Do Exercise 6.3.2.

6.3.2 EXERCISE [★★★]: De Bruijn's original article actually contained two different proposals for nameless representations of terms: the deBruijn *indices* presented here, which number lambda-binders "from the inside out," and *de Bruijn levels*, which number binders "from the outside in." For example, the term `λx. (λy. x y) x` is represented using deBruijn indices as `λ. (λ. 1 0) 0` and using deBruijn levels as `λ. (λ. 0 1) 0`. Define this variant precisely and show that the representations of a term using indices and levels are isomorphic (i.e., each can be recovered uniquely from the other). □

- Read Chapter 7 and digest the *fulluntyped* implementation includes extensions such as numbers and booleans.