Lecture 3

1 Syntax of simple typed λ-Expressions

(A) **Raw typed terms** are defined by the following grammar:

$\langle Exp \rangle ::= Var \mid Cons \mid \langle Exp \rangle \langle Exp \rangle \mid \lambda \langle Var \rangle \ : \ \tau. \ \langle Exp \rangle$, where τ ranges over the type expressions.

(B) **Typing rules:**

Type environment (assignment): function from variable names to type expressions. The notation $x_1 : \tau_1, \ x_2 : \tau_2, \ldots x_n : \tau_n$ is used for the type environment that maps x_i to τ_i $(i = 1, \ldots n)$.

Type Judgment: $H \vdash M : \ \sigma$, where H is a type environment, M is a raw typed term and σ is a type.

Inference rules for typed judgments:

Proj $H, \ x : \tau \vdash x : \tau$

Abs $\frac{H, \ x:\tau \vdash M:\sigma}{H \vdash \lambda \ x:\tau. \ M) \ :\tau \to \sigma}$

Apl $\frac{H \vdash M_1 \ : \ \tau \to \sigma, \qquad H \vdash M_2 \ : \ \tau}{H \vdash M_1 M_2 \ : \ \sigma}$

M has a type σ in a type environment H if $H \vdash M : \sigma$ is provable.

Lemma 1 *If H and H' agree on all free variables of M then $H \vdash M : \sigma$ iff $H' \vdash M : \sigma$.*

Lemma 2 *If $H \vdash M : \sigma$ and x does not appear in H then x is not free in M.*

1.1 Type checking and type restoring

Type checking Problem: Given $H, \ M, \ \sigma$. Check whether $H \vdash M : \sigma$.

Theorem 3 *There exists an algorithm which solves type checking problem.*

Type restoring:

Let ERASE be a function that deletes type information from the terms.

ERASE(x) $\triangleq$ x; ERASE(MN) $\triangleq$ERASE(M)ERASE(N); ERASE($\lambda x : \tau.\ M$) $\triangleq$ λx. ERASE(M).

ERASE maps raw terms into type free terms.

Type restoring Problem: Given type free term M. Check whether M can be well typed, i.e. whether there exists H, σ and M' such that M=ERASE(M') and $H \vdash M : \sigma$. If M can be well typed find all possible types for M.

Example 4 *$\lambda x.\ x$ can be well typed. For every type σ, term $\lambda x : \sigma.\ x$ has type $\sigma \to \sigma$. Hence, $t \to t$ is the most general polytype for $\lambda x.\ x$.*

$\lambda xy.x$ can be well typed. It has the most general polytype $t_1 \to t_2 \to t_1$.

$\lambda x.\ xx$ cannot be well typed.

Theorem 5 *There exists an algorithm which solves the type restoring problem. Moreover, the algorithm returns the most general type for the closed terms which can be well typed.*

Method: to solve type restoring problem.

Given a type free term M.

1. Assign a type variable to every variable in M in the following way:
   (a) to all free occurrences of x the same type variable is assigned.
   (b) To different free variables different type variables are assigned.
   (c) To all free occurrences of x in the scope of λx. the same type variable is assigned.
2. Assign type variable to every sub-term.
3. Construct a system of equations. For every application and abstraction in M corresponds one equation in the system. The equation are constructed according to the following rules:
   (a) For sub-term $(M^\alpha N^\beta)^\gamma$ - equation $\alpha = \beta \to \gamma$.
   (b) For sub-term $(\lambda x : \alpha.\ M^\beta)^\gamma$ - equation $\gamma = \alpha \to \beta$.
   (c) Solve the system of equations. This system is an instance of multiple unification problem. We know how to solve such problem (see Lecture 1).

Example 6 *M is* $\lambda xyz.\ xyz$.

$$M = \lambda x^{\alpha}.(\lambda y^{\beta}.(\lambda z^{\gamma}.\ ((x^{\alpha}y^{\beta})^{\delta}z^{\gamma})^{\epsilon})^{\tau_1})^{\tau_2})^{\tau_3}$$

Equations:

$$\begin{aligned} \alpha &= \beta \to \delta \\ \delta &= \gamma \to \epsilon \\ \tau_1 &= \gamma \to \epsilon \\ \tau_2 &= \beta \to \tau_1 \\ \tau_3 &= \alpha \to \tau_2 \end{aligned}$$

Solution:

$$\begin{aligned} \alpha &= \beta \to \gamma \to \epsilon \\ \tau_3 &= (\beta \to (\gamma \to \epsilon)) \to \beta \to \gamma \to \epsilon. \end{aligned}$$

Exercise: Solve type restoring problem for the terms $M_1 \equiv \lambda xyz.\ x(yz)$, $M_2 \equiv \lambda xyz.\ xz(yz)$, $M_3 \equiv (\lambda x.\ yx)\lambda yz.y(yz)$. **Exercise:** Show that if M is closed term and $\vdash M : \sigma$ then *sigma* is a tautology of propositional calculus.

2 Reductions

Notations: Naive substitution: $M[N//x]$ replace all free occurrences of x in M by N.

Substitution: $M[N/x]$. Other notations - $[N/x]M$, $M\{N/x\}$, $\{N/x\}M$, $M[x \to N]$. Replace all free occurrences of x in M by N renaming the bound variables of M if necessary.

α**-conversion:** rename the names of bound variables preserving the skeleton. $E_1 =_\alpha E_2$.

$$\lambda x.M =_\alpha \lambda y.\ M[y/x], \text{ whenever } y \text{ is a fresh variables}$$

β**-reduction**: replace formal parameter by actual argument. λ-abstraction denotes a function. To apply $\lambda x.\ M$ to an argument N we replace x in M by N.

However, care must be taking to avoid capturing free variables of N.

Example: $(\lambda x.\lambda y.\ x)4$ reduces to the constant function $\lambda y.\ 4$. Similarly, $(\lambda x.\lambda y.\ x)z$ reduces to the constant function $\lambda y.\ z$. If we naively substitute y for x in the evaluation of $(\lambda x.\lambda y.\ x)y$ we obtain the identity function $\lambda y.\ y$. Hence, for β-reduction we have to use not naive substitution.

$(\lambda x.\ M)N \to_\beta M[N/x]$.

Our expectation from substitution: if $M_1 =_\alpha M_2$ and $N_1 =_\alpha N_2$ then $M_1[N_1/x] =_\alpha M_2[N_2/x]$.

Definition 1 *(Substitution)*

1. *Atoms*
   (a) $x[M/x] \triangleq M$
   (b) $x[M/y] \triangleq y$
   (c) $c[M/x] \triangleq c$
2. *Application* $N_1 N_2[M/x] \triangleq N_1[M/x]N_2[M/x]$
3. *Abstraction*
   (a) $(\lambda x.\ N)[M/x] \triangleq (\lambda x.\ N)$
   (b) $(\lambda y.N)[M/x] \triangleq \lambda y.\ N[M/x]$ *if* $y \notin FV(M)$
   (c) $(\lambda y.N)[M/x] \triangleq (\lambda z.\ N[z/x])[M/x]$, *(where z is a fresh variable), if* $y \in FV(M)$

Example 7 $(\lambda y.\ xy)(\lambda x.\ xx)[\lambda z.\ yz/x]$.

Exercise: Make substitution

$$(\lambda x.\ xy\lambda z.\ xy(z\lambda y.\ yz)[\lambda v.xyzv/y]$$

Reduction Order: an expression can contain several redexes. $(+(\times 2\ 7)(+4\ 5))$. Several reduction sequences.

An expression is in normal form if it does not contain any redex.

Not every expression has a β-normal form. $(\lambda x.\ xx)(\lambda x.\ xx)$.

Some reduction sequences might terminate, while others do not. $(\lambda y.3)((\lambda x.\ xx)(\lambda x.\ xx))$.

3 Operational Semantics for λ terms

3.1 Operational Semantics

An operational semantics of Prog. Lang. presents a dynamic view on programs. A program is evaluated by a sequence of **symbolic** transformations - reductions.

3.2 Summary of reduction rules for λ-expressions

Terms (expressions) are considered up to renaming of bound variables (formal parameters).

β-reduction; Apply λ-abstraction to its argument.

$$(\lambda x.\ M)N \text{ is } \beta \text{ reduced to } M[N/x]$$

Terminology, Notations

- $(\lambda x.\ M)N$ is called a *β redex; $M[N/x]$ is its contraction.*
- **One step** *of β-reduction: $M_1 \to_\beta M_2$ - replace an occurrence of a β-redex in M_1 by its contraction.*
- *M* **is many step** *β-reducible to N if $M \to_\beta M_1 \to_\beta M_2 \cdots \to_\beta M_k = N$. Notations: $M \to_\beta^*$.*
- *M is in β* **normal form** *if M does not contain any β redex.*
- *$=_\beta$ is reflexive, transitive symmetric closure of $\to_{beta}$; $=_\beta$ is called β* **conversion**.
- *N is a β* **normal form of** *M if (1) N is in β normal form and (2) $M \to_\beta^* N$.*

η-reduction: Eliminate the redundant λ-abstraction.

$$(\lambda x.\ Mx) \to_\eta M. \text{ provided } x \text{ does not occur free in } M$$

One step η reduction, many step η reduction, η-normal form, η-conversion are defined in an obvious way.

Reductions for build-in constant.

$$+5\ 4 \to_{arith} 9$$

$$And\ true\ false \to_{bool} false$$

$$IF - THEN - ELSE\ true M_1 M_2 \to M_1$$

$$IF - THEN - ELSE\ false M_1 M_2 \to M_2$$

3.3 Presentation of β-reduction by inference rules

$$\frac{}{(\lambda x.\ M)N \to_\beta M[N/x]}$$

$$\frac{M_1 \to_\beta M_2}{M_1 N \to_\beta M_2 N}$$

$$\frac{M_1 \to_\beta M_2}{N M_1 \to_\beta N M_2}$$

Many properties of operational semantics are proved by Induction on Number of inference rules.

3.4 Reduction Sequences

$$(\lambda x.\ y)(\lambda z.\ z) \to_\beta y$$

$$(\lambda x.xxx)(\lambda z.\ z) \to_\beta (\lambda z.z)(\lambda z.z)(\lambda z.z) \to_\beta (\lambda z.z)(\lambda z.z) \to_\beta (\lambda z.z)$$

$(\lambda x.(\lambda y.yx)z)zw-$ has two reduction sequences leading to the same normal form $z(zw)$

Process of reduction should simplify the expression without changing its MEANING. Many reduction sequences for an expression make reduction semantics non-deterministic. Are the result consistent?

Contraction does not always simplify expressions

Example 8

$$(\lambda x.\ xx)(\lambda x.\ xx) \to$$

$$(\lambda x.\ xxx)(\lambda x.\ xxx) \to$$

$$(\lambda x.\ f(xx))(\lambda x.\ f(xx)) \to$$

Unique infinite reduction sequence.

Example 9 $(\lambda xy.\ y)((\lambda x.\ xx)(\lambda x.\ xx))$ *-left reduction sequence finite, right reduction sequence infinite.*

4 Programming in λ-calculus.

Our interest is in functional programming languages that can be obtained by an extension of λ-calculus. However, one can actually can program in pure λ-notations.

Encoding the booleans: Must define terms **if, true and false** satisfying.

if true$MN = M$ and **if false** $MN = M$

Define **true**$\equiv \lambda xy.\ x$, **false** $\equiv \lambda xy.\ y$, **if** $\equiv \lambda pxy.\ pxy$

Check:

$\mathbf{true} MN \to_\beta^* M$
$\mathbf{true} MN \to_\beta^* N$
$\mathbf{if}\ \mathbf{true} MN \to_\beta^* M$
$\mathbf{if}\ \mathbf{false}\ MN \to_\beta^* N$

$\mathbf{and} \equiv \lambda pq.\ \mathbf{if}\ p\ q\ \mathbf{false}$
$\mathbf{or} \equiv \lambda pq.\ \mathbf{if}\ p\ \mathbf{true}\ q$
$\mathbf{not} \equiv \lambda p.\ \mathbf{if}\ p\ \mathbf{false}\ \mathbf{true}$

Encoding natural numbers:
$0 \equiv \lambda fx.\ x$

$1 \equiv \lambda fx.\ fx$

$2 \equiv \lambda fx.\ f(fx)$

$n \equiv \lambda fx.\ \underbrace{f \ldots (f}_{n}\, x).$

$\mathbf{add} \equiv \lambda mnfx.\ mf(nfx)$

$\mathbf{mult} \equiv \lambda mnfx.\ m(nf)x$

$\mathbf{suc} \equiv \lambda nfx.\ f(nfx)$

$\mathbf{iszero} \equiv \lambda n.\ n(\lambda x.\mathbf{false})\mathbf{true}.$

Definition 2 *A λ-term M is a* **representation** *of a partial function $f : Nat \to Nat$ if for every $n \in Nat$ and corresponding Church numeral $\bar{n}$: $f(n)$ is defined iff $M\bar{n}$ has β normal form. Moreover, if $f(n) = m$ then $\bar{m}$ is the β-normal form of $M\bar{n}$.*

Theorem 10 *A partial function has a λ-representation if and only if it is a partial recursive function.*