Deadline: 31.10.2023 23:59:59

- Recommended book: Types and Programming Languages by B. C. Pierce; chapters 5 (untyped lambda calculus) and 9 (simply typed lambda calculus).
- Great paper on Reduction Strategies: Demonstrating Lambda Calculus Reduction by P.Sestoft

1 Simply typed lambda calculus (STLC, $\lambda_{\to}$)

1.1 Syntax (Pure Calculus extended with Base Types α_i)

$$\begin{array}{rcll} M, N & ::= & x \mid \lambda x{:}\, \tau.N \mid MN & \text{terms} \\ \tau, \sigma & ::= & \alpha_i \mid \tau \to \sigma & \text{simple types} \end{array}$$

1.2 Typing Rules

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}\ \text{Var/Ax} \qquad \frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x{:}\, \tau.M : \tau \to \sigma}\ \text{Abs}/\to\text{I} \qquad \frac{\Gamma \vdash N : \tau \quad \Gamma \vdash M : \tau \to \sigma}{\Gamma \vdash MN : \sigma}\ \text{App}/\to\text{E}$$

1.3 Reduction (Evaluation) Rules

$$\frac{M \longrightarrow M'}{M\ N \longrightarrow M'\ N}\ EApp_1 \qquad \frac{N \longrightarrow N'}{v\ N \longrightarrow v\ N'}\ EApp_2 \qquad \frac{}{(\lambda x : \tau.M)\ v \longrightarrow M[x/v]}\ EBeta$$

Note:

1. substitution always assumed to be capture-avoiding;
2. v is a value ($\lambda x : \tau.M$);
3. corresponds to call-by-value.

1.4 Examples

à la Curry	à la Church
$\lambda x.x \ :\ \alpha \to \alpha \qquad \lambda x.x \ :\ (\alpha \to \beta) \to \alpha \to \beta$	$\lambda x^{\alpha}.x \ :\ \alpha \to \alpha \qquad \lambda x^{\alpha\to\beta}.x \ :\ (\alpha \to \beta) \to \alpha \to \beta$
$\lambda xy.x \ :\ \alpha \to \beta \to \alpha$	$\lambda x^{\alpha} y^{\beta}.x \ :\ \alpha \to \beta \to \alpha$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{f : \beta \to \gamma,\ g : \alpha \to \beta,\ x : \alpha \vdash f\ :\ \beta \to \gamma}\ Ax \quad \dfrac{\dfrac{}{\cdots, g : \alpha \to \beta, \cdots \vdash g\ :\ \alpha \to \beta}\ Ax \quad \dfrac{}{\cdots,\ x : \alpha \vdash x\ :\ \beta}\ Ax}{f : \beta \to \gamma,\ g : \alpha \to \beta,\ x : \alpha \vdash g\ x\ :\ \beta}\ \to E}{f : \beta \to \gamma,\ g : \alpha \to \beta,\ x : \alpha \vdash f\ (g\ x)\ :\ \gamma}\ \to E}{f : \beta \to \gamma,\ g : \alpha \to \beta \vdash \lambda\ x^{\alpha}.\ f\ (g\ x)\ :\ \alpha \to \gamma}\ \to I}{f : \beta \to \gamma \vdash \lambda\ g^{(\alpha\to\beta)} x^{\alpha}.\ f\ (g\ x)\ :\ (\alpha \to \beta) \to \alpha \to \gamma}\ \to I$$

$$\dfrac{f : \beta \to \gamma \vdash \lambda\ g^{(\alpha\to\beta)} x^{\alpha}.\ f\ (g\ x)\ :\ (\alpha \to \beta) \to \alpha \to \gamma}{\vdash \lambda\ f^{(\beta\to\gamma)} g^{(\alpha\to\beta)} x^{\alpha}.\ f\ (g\ x)\ :\ (\beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma}\ \to I$$

$$\cfrac{\cfrac{\cfrac{x : Bool \;\in\; \{x : Bool\}}{x : Bool \vdash x : Bool}\;\text{Var}}{\vdash \lambda x : Bool.x \;:\; Bool \to Bool}\;\text{Abs} \qquad \cfrac{}{\vdash true \;:\; Bool}\;\text{T-True}}{\vdash (\lambda x : Bool.x)\; true \;:\; Bool}\;\text{App}$$

1.5 Example: STLC extended with Let-bindings

$$\begin{array}{rcll} M, N & ::= & x \mid \lambda x.N \mid MN \mid \text{let } x = N \text{ in } M & \qquad\qquad \text{terms} \\ \tau, \sigma & ::= & \alpha_i \mid \tau \to \sigma & \qquad\qquad \text{types} \\ v & ::= & \lambda x.M & \qquad\qquad \text{value} \end{array}$$

new typing rule:

$$\cfrac{\Gamma \vdash N : \tau_1 \quad \Gamma, x : \tau_1 \vdash M : \tau_2}{\Gamma \vdash \text{let } x = N \text{ in } M \;:\; \tau_2}\; let$$

new evaluation rules:

$$\text{let } x = v \text{ in } M \longrightarrow M[x/v] \quad LetV \qquad\qquad \cfrac{N \longrightarrow N'}{\text{let } x = N \text{ in } M \longrightarrow \text{let } x = N' \text{ in } M}\; Let$$

1.6 $\text{Int}_{\to}$ (Implicative part of Propositional Intuitionistic Logic)

$$\cfrac{}{\Gamma, \alpha \vdash \alpha}\; Ax \qquad \cfrac{\Gamma \vdash \alpha \to \beta \quad \Gamma \vdash \alpha}{\Gamma \vdash \beta} \to E$$ Modus ponens $$\cfrac{\Gamma, \alpha \vdash \beta}{\Gamma \vdash \alpha \to \beta} \to I$$ Hilbert's Deduction Theorem

Example:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{}{\Gamma \vdash \beta \to \gamma}\; Ax \quad \cfrac{\cfrac{}{\Gamma \vdash \alpha \to \beta}\; Ax \quad \cfrac{}{\Gamma \vdash \beta}\; Ax}{\Gamma \vdash \beta} \to E}{\Gamma \equiv \beta \to \gamma,\ \alpha \to \beta,\ \alpha \vdash \gamma} \to E}{\beta \to \gamma,\ \alpha \to \beta \vdash \alpha \to \gamma} \to I}{\beta \to \gamma \vdash (\alpha \to \beta) \to \alpha \to \gamma} \to I}{\vdash (\beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma} \to I$$

1.7 Encodings

Booleans

$T = true = \lambda t.\lambda f.\, t$
$F = false = \lambda t.\lambda f.\, f$
$test = \lambda l.\lambda m.\lambda n.\, l\; m\; n$ ($\sim$ if then else)
$and = \lambda l.\lambda m.\lambda n.\, l\; m\; n$

Pairs

$pair = \lambda f.\lambda s.\lambda b.\, b\; f\; s$
$fst = \lambda p.\, p\; T$
$snd = \lambda p.\, p\; F$

Church Numerals

$c_0 = \lambda s.\lambda z.\ z$
$c_1 = \lambda s.\lambda z.\ s\ z$
$c_i = \lambda s.\lambda z.\ s^i\ z$
$succ = \lambda n.\lambda s.\lambda z.\ s\ (n\ s\ z)$
$plus = \lambda m.\lambda n.\lambda s.\lambda z.\ m\ s\ (n\ s\ z)$
$times = \lambda m.\lambda n.\lambda s.\lambda z.\ m\ (n\ s)\ z$
$iszero = \lambda m.\ m\ (\lambda x.\ F)\ T$
$pred = \lambda n.\lambda s.\lambda z.\ n\ (\lambda f.\lambda h.\ f\ (g\ f))\ (\lambda u.z)\ (\lambda v.v)$
$subt = \lambda m.\lambda n.\ n\ pred\ m$

2 Recursion

Recursion in untyped lambda calculus can be expressed with so-called fix-point combinators.
Theorem. Forall term F there exists term V such that $V =_\beta F\ V$.
Proof: $V = (\lambda x.\ F\ (x\ x))(\lambda x.\ F\ (x\ ))$ Qed.
Theorem. $\exists \mathbb{Y} :\ \forall F.\ \mathbb{Y}\ F\ \ \to_\beta F\ (\mathbb{Y}\ F)$.
Proofs:

1. $\mathbb{Y} = \lambda f.\ (\lambda x.\ f\ (x\ x))\ (\lambda x.\ f\ (x\ x))$
2. $A = \lambda x.\lambda y.\ y\ (x\ x\ y)$; $\Theta = A\ A$
3. ...

Qed.
Theorem (First Recursion Theorem). $\forall M.\exists F :\ F =_\beta M[f/F]$.
Proofs: $F = \mathbb{Y}\ (\lambda f.M)$ Qed.

Note, in $\lambda_\to$ recursion is impossible. One can extend $\lambda_\to$ with explicit fixed-point combinator by defining corresponding new term expression, typing and evaluation rules. The correct form of fixed-point combinator depends on evaluation strategy, for example $\mathbb{Y}$ is standard combinator for call-by-name, while Θ is standard call-by-value fixed-point combinator.
Example: factorial function $fact$ on church numerals
$f = \lambda f.\lambda n.$ if $iszero\ n$ then c_1 else $times\ n\ (f\ (pred\ n))$
$fact = fix\ g$
where fix is the corresponding fixed-point combinator.

2.1 Exercises

1. Prove that the following statements are derivable in STLC (provide type derivation)
   (a) $f : Bool \to Bool \vdash f\ (if\ false\ then\ true\ else\ false)\ :\ Bool$
   (b) $f : Bool \to Bool \vdash \lambda x : Bool.\ f\ (if\ x\ then\ false\ else\ x)\ :\ Bool \to Bool$

assuming

$$\frac{\Gamma \vdash e \;:\; Bool \quad \Gamma \vdash v \;:\; \tau \quad \Gamma \vdash u \;:\; \tau}{\Gamma \vdash if\ e\ then\ v\ else\ u \;:\; \tau}\ \text{T-If}$$

2. Find all inhabitants (closed terms) of the following types (both in à la Curch and à la Curry):
   (a) $(\alpha \to \beta) \to (\beta \to \gamma) \to \alpha \to \gamma$
   (b) $\alpha \to \beta \to (\alpha \to \beta \to \gamma) \to \gamma$
   (c) $((\alpha \to \beta \to \alpha) \to \alpha) \to \alpha$
   (d) $\beta \to ((\alpha \to \beta) \to \gamma) \to \gamma$
   (e) $\alpha \to (\alpha \to \alpha) \to \alpha$
3. Compute the most general (principal) type of the following terms
   (a) $S = \lambda x\, y\, z.\ x\, z\, (y\, z)$
   (b) $K = \lambda x\, y.\ x$
   (c) SKK
   (d) $I = \lambda x.\ x$
4. Construct a derivation of type $((\alpha \to \beta) \to \gamma) \to \beta \to \gamma$ and the associated typed λ-term
5. Add product types to $\lambda_\to$, that is add $\sigma \times \tau$ to the types and
   (a) Add the appropriate term constructor and projections
   (b) Define typing rules
   (c) Define reduction rules for the new term constructors
   (d) Define (bii)map function for pairs and provide derivation for it
6. Besides β-equivalence there exists another form of equivalence on lambda terms called η-equivalence or η-coercion (denoted $=_\eta$; η-expansion $\to_\eta$ and η-reduction $\leftarrow_\eta$). It is defined by $M =_\eta \lambda x.\ M\ x$. Note, it can't be expressed via β-reductions. Also, note that in untyped calculus a term can be η-expanded an arbitrary number of times, while in simply typed lambda calculus η-expansion is obviously limited by the term's type. Term of $\lambda_\to$ is said to be in η-long form if it is fully η-expanded; formally, it can be defined with the following grammar where $m \in \mathbb{N}$, $n, p \in \mathbb{N}_0$:

$$\begin{aligned} \Lambda^{lf}_{\text{odd}} &::= \lambda x : \tau_1 \ldots x : \tau_p.\ \Lambda^{lf}_{\text{even}} \\ \Lambda^{lf}_{\text{odd}} &::= x\ {\Lambda_1}^{lf}_{\text{odd}}\ \ldots\ {\Lambda_n}^{lf}_{\text{odd}} \mid {\Lambda_0}^{lf}_{\text{odd}}\ @_{\text{long}}\ {\Lambda_1}^{lf}_{\text{odd}}\ \ldots\ {\Lambda_m}^{lf}_{\text{odd}} \end{aligned}$$

   In other words, being viewed as a tree, all odd level nodes are abstractions over an arbitrary number of variables, while even level nodes are applications. Find the η-long form of the following term:

$$test\ (mult\ c_3\ c_2)\ (snd\ (pair\ (and\ T\ F)\ c_1))$$

7. Provide step-by-step evaluation of term $fact\ c_3$ with both call-by-name and call-by-value reduction strategies.