EB1 - Introduction to the Typed Lambda Calculus

We'll use $M, N, P, Q, \ldots$ for terms, $V, W, Y, \ldots$ for values and $\sigma, \tau, \rho, \ldots$ for types. Unless specified otherwise, we'll work with the following lambda calculus that has types and terms:

$$
\begin{array}{rcl}
\sigma, \tau & ::= & \mathbb{B} \\
& | & \mathbb{N} \\
& | & \sigma \to \tau
\end{array}
\qquad\qquad
\begin{array}{rcl}
M, N, P & ::= & x \\
& | & \texttt{true} \\
& | & \texttt{false} \\
& | & \mathit{if}\ M\ \mathit{then}\ N\ \mathit{else}\ P \\
& | & \lambda x : \sigma . M \\
& | & M\ N \\
& | & \mathit{suc}(M) \\
& | & \mathit{pred}(M) \\
& | & \mathit{isZero}(M)
\end{array}
$$

Exercises marked with a ★ are the bare minimum that you should consider answering. However, I strongly encourage you to work through them all.

Syntax

Exercise 1 ★

Consider the following terms:

1. $ux(yz)(\lambda v.vy)$
2. $(\lambda x : \sigma \to \tau \to \rho . \lambda y : \sigma \to \tau . \lambda z : \sigma . xz(yz))uvw$
3. $w(\lambda x : \sigma \to \tau \to \rho . \lambda y : \sigma \to \tau . \lambda z : \sigma . xz(yz))uv$

Answer the following questions/requests:

1. Insert all parenthesis according to the standard convention.
2. Indicate which variable occurrences are free and bound.
3. In which of the above terms does the following occur $(\lambda x : \sigma \to (\tau \to \rho) . \lambda y : \sigma \to \tau . \lambda z : \sigma . xz(yz))u$?
4. Does $x(yz)$ occur in $ux(yz)$?

Exercise 2 ★

Compute the following substitutions:

1. $(\lambda y : \sigma . x(\lambda x : \tau . x))\{x \leftarrow (\lambda y : \rho . xy)\}$
2. $(y(\lambda v : \sigma . xv))\{x \leftarrow (\lambda y : \tau . vy)\}$

Rename variables in both terms for better readability.

Exercise 3

Prove the following, where $|M|$ denotes the size of M (i.e. number of symbols) and $\subseteq$ the subterm relation:

1. $M\{x \leftarrow x\} = M$
2. $|M\{x \leftarrow y\}| = |M|$
3. if $x \notin FV(M)$ then $M\{x \leftarrow N\} = M$

4. if $M \subseteq N$ then $|M| \leq |N|$
5. $|N\{x \leftarrow M\}| \geq |N|$. In which cases does equality hold?

Exercise 4

Provide examples of:

1. Terms that are values.
2. Terms that are in normal form but are not values.
3. Terms that are programs.
4. Terms that are not programs.

Operational Semantics

Exercise 5 ★

Compute the values to which these terms reduce using small-step semantics:

1. $(\lambda x : Bool.\lambda y : Bool \to Bool.yx)\,\texttt{true}\;(\lambda x : Bool.x)$
2. $(\lambda x : Bool.\lambda y : Bool \to Bool.y(yx))\,\texttt{false}\;(\lambda x : Bool.x)$
3. $(\lambda x : Bool.\lambda y : Bool \to Bool.y(yx))\,((\lambda z : Bool.\texttt{true})\texttt{false})\;(\lambda x : Bool.x)$
4. $(\lambda x : Bool \to Bool \to Bool.\lambda y : Bool \to Bool.\lambda z : Bool.xz(yz))\,(\lambda s : Bool.\lambda t : Bool.s)\,(\lambda u : Bool.u)w$
   What would happen if in item 4 it said `true` or `false` instead of w?

Exercise 6

1. Suppose $\texttt{if}\ M\ \texttt{then}\ P\ \texttt{else}\ Q \to D$. Is it true that D would necessarily have the form $\texttt{if}\ M'\ \texttt{then}\ P'\ \texttt{else}\ Q'$?
2. Suppose $M \twoheadrightarrow M'$, $P \twoheadrightarrow P'$, $Q \twoheadrightarrow Q'$. Is it true that $\texttt{if}\ M\ \texttt{then}\ P\ \texttt{else}\ Q \twoheadrightarrow \texttt{if}\ M'\ \texttt{then}\ P'\ \texttt{else}\ Q'$?

Exercise 7

1. Is it the same to evaluate $suc(pred(M))$ or $pred(suc(M))$? Why?
2. Is it true that for every M, $isZero(suc(M)) \twoheadrightarrow \texttt{false}$? If not, for which ones?
3. For which terms M do we have $isZero(pred(M)) \twoheadrightarrow \texttt{true}$?

(Hint: do not think only of normal forms)

Exercise 8 ★

Find M such that both of these conditions are satisfied simultaneously:

- $M\underline{3} \twoheadrightarrow \texttt{true}$ and
- $M\underline{n} \twoheadrightarrow \texttt{false}$, if $n \neq 3$.

Exercise 9 ★

Prove that if $E \to F$ then $FV(F) \subseteq FV(E)$. Find cases in which equality holds and in which strict inclusion holds.

Exercise 10

Let M, N, P, Q be terms. Indicate whether the following are true or false.

1. If $M \to N$, then $M\{x \leftarrow P\} \to N\{x \leftarrow P\}$
2. If $P \to Q$, then $M\{x \leftarrow P\} \to M\{x \leftarrow Q\}$
3. If $M \to N$, then $MP \to NP$

4. If $P \to Q$, then $MP \to MQ$

Exercise 11 *(Boolean Connectives)* ★

Define terms that behave as the operators **Not**, **And**, **Or**, **Xor**. Eg. **And** $M\ N \twoheadrightarrow \texttt{true}$ iff $M \twoheadrightarrow \texttt{true}$ and $N \twoheadrightarrow \texttt{true}$.

Exercise 12 *(Determinism)* ★

1. Prove that the relation given by $\to$ is deterministic (i.e. a partial function). That is, show that if $M \to N$ and $M \to N'$, then $N = N'$.
2. Does the same hold for multiple steps? I.e. is it true that if $M \twoheadrightarrow M'$ and $M \twoheadrightarrow M''$, then $M' = M''$?
3. Does the following hold: if $M \to M'$ and $M \twoheadrightarrow M''$, then $M' = M''$?

Exercise 13

The variant of Lambda Calculus seen in class uses call-by-value: arguments are fully evaluated before passing them on to functions. How would you change the calculus to call-by-name (i.e. passing arguments on without evaluating them)

Exercise 14

Evaluate the following terms to normal form using the call-by-name strategy (using the rules given in exercise 13). You may assume that $comp \stackrel{\text{def}}{=} \lambda f : N \to N.\lambda g : N \to N.\lambda x : N.f(gx)$

1. $comp\ (\lambda x : N.succ(x))\ (\lambda x : N.succ(x))\ \underline{5}$
2. $\texttt{let}\ f(x : N) : N = \underline{3}\ \texttt{in letrec}\ g(x : N) : N = g(succ(x))\ \texttt{in}\ f(g\ \underline{5})$
3. $\texttt{let}\ not = \lambda x : bool.\texttt{if}\ x\ \texttt{then}\ false\ \texttt{else}\ true$
   $\texttt{in letrec}\ f = \lambda x : bool.\texttt{if}\ x\ \texttt{then}\ true\ \texttt{else}\ f(not\ x)\ \texttt{in}\ f\ false$

Note: `letrec` works just like `let` but admits recursive definitions.

Exercise 15

Assuming the call-by-name strategy, which expressions of Exercise 5 are in normal form?

Typing

Exercise 16 ★

Prove:

1. $\{x : Bool,\ y : Bool\} \triangleright \texttt{if}\ x\ \texttt{then}\ x\ \texttt{else}\ y : Bool$
2. $\{x : Bool \to Bool,\ y : Bool\} \triangleright \texttt{if}\ xy\ \texttt{then}\ y\ \texttt{else}\ xy : Bool$

Exercise 17 ★

Exhibit a term that is not typable and does not have free variables nor abstractions.

Exercise 18 ★

Let σ, τ, ρ be types. Consider the following terms. They are called *combinators* and arise from *Combinatory Logic*. Combinatory Logic is Turing-complete and does not include any notion of variables binding. In this example, we consider encodings, in lambda calculus, of these combinators:

$$\begin{aligned} \mathbf{S}_{\sigma\tau\rho} &= \lambda x : \sigma \to \tau \to \rho.\lambda y : \sigma \to \tau.\lambda z : \sigma.xz(yz) \\ \mathbf{K}_{\sigma\tau} &= \lambda x : \sigma.\lambda y : \tau.x \\ id_\sigma &= \lambda x : \sigma.x \end{aligned}$$

Use the above mentioned definitions to prove that:

1. $\triangleright \mathbf{S}_{\sigma\tau\rho} : (\sigma \to \tau \to \rho) \to (\sigma \to \tau) \to (\sigma \to \rho)$
2. $\triangleright \mathbf{S}_{\sigma\tau\sigma}\mathbf{K}_{\sigma\tau} : (\sigma \to \tau) \to \sigma \to \sigma$
3. $\triangleright \mathbf{K}_{\sigma\tau} id_\sigma : \tau \to \sigma \to \sigma$

Exercise 19 *(Numerales de Church)*

Notation: $M^0(N) = N, M^{n+1}(N) = M(M^n(N))$ Give appropriate types σ and τ so that terms of the form $\lambda y : \sigma, x : \tau.y^n(x)$ are typable for all $n > 0$. The pair (σ, τ) must be the same type for all terms. Observe whether all terms are given the same type.

Exercise 20 ★

1. Show with an example that *uniqueness of types* can fail if different typing contexts are considered (i.e. $\Gamma \triangleright M : \sigma$ and $\Gamma' \triangleright M : \sigma'$ do not imply $\sigma = \sigma'$).
2. Show with an example that *type preservation* (subject reduction) can fail if different contexts are assumed (i.e. $\Gamma \triangleright M : \sigma$ and $M \to M'$ do not imply $\Gamma' \triangleright M' : \sigma$).

Exercise 21 ★

Prove that there exist no Γ and σ s.t. $\Gamma \triangleright xx : \sigma$. What conclusion can you arrive at from this? (Note: it may be proved that $\lambda x : \sigma.xx$, $id(\lambda x : \sigma.xx)$ is not typable either, and many other terms too.)

For this exercise you may use the following *generation lemma*: if $\Gamma \triangleright M : \sigma$ is derivable, then:

- if $M = x$, then $x : \sigma \in \Gamma$
- if $M = n$, then $\sigma = nat$
- if $M = \lambda x : \tau.N$, then there exists ρ s.t. $\sigma = \tau \to \rho$ and $\Gamma, x : \tau \triangleright N : \rho$ is derivable
- if $M = PQ$, then there exists ρ s.t. $\Gamma \triangleright P : \rho \to \sigma$ and $\Gamma \triangleright Q : \rho$ are derivable.

Exercise 22

1. Prove that if $\Gamma \triangleright M : Nat$ and $M \to M'$, then $\Gamma \triangleright suc(M') : Nat$ and $\Gamma \triangleright pred(M') : Nat$.
2. Prove that if $\Gamma \triangleright$ `if` B `then` P `else` $Q : \sigma$, $B \to B'$, $P \to P'$ and $Q \to Q'$, then $\Gamma \triangleright$ `if` B' `then` P' `else` $Q' : \sigma$.

Exercise 23

Determine whether the following statements are T or F:

1. $\Gamma \triangleright M$: σ and $\Gamma \triangleright N : \sigma'$, then there exists τ s.t. $\Gamma \triangleright MN$: τ
2. $\Gamma \triangleright M$: σ then for all σ' there exists τ s.t. $\Gamma \triangleright \lambda x : \sigma'.M$: τ

Exercise 24 *(Substitution Lemma)*

Prove: if $\Gamma, x : \tau \triangleright M : \sigma$ and $\Gamma \triangleright N : \tau$ are derivable, then $\Gamma \triangleright M\{x \leftarrow N\} : \sigma$ is derivable.

Extensions

Exercise 25 ★

1. Define the following term as a macro (i.e. syntactic sugar): a function $apply_\sigma$ that takes a function $f : \sigma \to \sigma$ and a $x : \sigma$, and applies f to x.
2. Define the following extension (typing rules and reduction rules): the function *mult* that multiplies natural numbers.
3. Define as a macro the function *and* that takes two booleans and returns their conjunction (as defined in Exercise 11). Which version is more practical?

Exercise 26 ★

We will extend the Lambda Calculus with Lists. First we add new types.

$$\sigma \quad ::= \quad \mathbb{N} \mid \mathbb{B} \mid \sigma \to \sigma \mid [\sigma]$$

where $[\sigma]$ represents the type of lists of type σ. The set of terms is extended as follows:

$$M \quad ::= \quad \ldots \mid nil_\sigma \mid M :: N \mid \texttt{case}_\sigma \ M \ \texttt{of} \ \{nil \to P \mid h :: t \to Q\}$$

where nil_σ is the empty list of elements of type σ, $M :: N$ adds M to the list N, and $\texttt{case}_\sigma M \ \texttt{of} \ \{nil \to P \mid h :: t \to Q\}$ is the eliminator/observer for lists (h and t are variables that are bound in Q).

1. Add the typing rules for the new expressions.
2. Add the reduction rules for the new expressions.

Exercise 27

Consider the following extension μ, to the Lambda Calculus. It is the same as the Lambda Calculus except that the mechanism for constructing functions ($\lambda x.M$) and applying them ($M \ N$) are replaced by new ones ($\mu x_1, ..., x_n.M$) and ($M \ \#_i \ N$). The type system is modified as follows: instead of $\sigma \to \tau$ we now have $\{\sigma_1, ..., \sigma_n\} \to \tau$. Note that $\{\sigma_1, ..., \sigma_n\}$ is not a type, it is part of the new type for functions.
In summary, we have:

$$M \ ::= \ ... \mid \mu x_1 : \sigma_1, ..., x_n : \sigma_n.M \mid M \ \#_i \ N \qquad \sigma \ ::= \ ... \mid \{\sigma_1, ..., \sigma_n\} \to \tau$$

The term $\mu x_1 : \sigma_1, ..., x_n : \sigma_n.M$ allows one to build a function with n ordered parameters and the operator $\#_i$ allows us to apply it to the i-th parameter. Note that if the number of parameters of a function is greater than 1, then when it is applied to an argument results in a function with one parameter less. But if it expects just one parameter, when applied to an argument it returns the result. Note also that the order of the parameters is important: eg. $\{nat, nat, bool\} \to nat$ and $\{bool, nat, nat\} \to nat$ do not have the same type.

1. Define the new typing rules.
2. Define the new set of values and reduction rules.
3. Define the standard constructions of the Lambda Calculus (λ and application) as *syntactic sugar* (macros) of the μ system.

Exercise 28

We'll consider an extension to the Lambda Calculus that allows to mix call-by-name (*cbn*) and -by-value (*cbv*) reduction.
We add a new term constructor $pack(M)$ that allows to "pack or freeze" a term M. When such a term is supplied as an argument to a function, i.e. in the case $(\lambda x.M) \ pack(N)$, reduction should proceed in *cbn*, and N must continue to be packed. The term is unpacked when computation focusses on it. When computation focusses on an unpacked term, the usual *cbv* behavior is expected. Packed terms will be considered values.
The syntax of the extension is:

$$M \ ::= \ \ldots \mid pack(M)$$

The set of types is not modified.

1. Define the new typing rules.
2. Note that the set of values is not modified. Define the new reduction rules, ensuring that reduction is deterministic.

Exercise 29 ★

Define the following functions in the Lambda Calculus with lists (Exercise 26). You may define them as macros or extension to the Lambda Calculus.

1. $head : [\sigma] \to \sigma$ and $tail : [\sigma] \to [\sigma]$ (assume that $\bot_\sigma \stackrel{\text{def}}{=} \mathit{fix}\, x : \sigma.x$).
2. *iterate* of type $(\sigma \to \sigma) \to \sigma \to [\sigma]$ that given f and x returns the infinite list $x :: f\ x :: f(f\ x) :: f(f(f\ x)) :: \ldots$
3. $zip : [\rho] \to [\sigma] \to [\rho \times \sigma]$ that behaves like in Haskell.
4. $take : nat \to ([\sigma] \to [\sigma])$ that behaves like in Haskell.