Exercise 1 (H) (Type Inference)

Give a derivation tree for the following statement, and so determine the type $\tau$:

$$[] \vdash \lambda x y z. x y (y z) : \tau$$

Exercise 2 (H) (Subject Reduction Property)

A type system has the subject reduction property if evaluating an expression preserves its type. Prove that the simply typed $\lambda$-calculus ($\lambda^\to$) has the subject reduction property:

$$\Gamma \vdash t : \tau \land t \to^\beta t' \implies \Gamma \vdash t' : \tau$$

Hints: Use induction over the inductive definition of $\to^\beta$ (Def. 1.2.2). State your inductive hypotheses precisely—it may help to introduce a binary predicate $P(t, t')$ to express the property you are proving by induction. Also note that the proof will require rule inversion: Given $\Gamma \vdash t : \tau$, the shape of $t$ (variable, application, or $\lambda$-abstraction) may determine which typing rule must have been used to derive the typing judgment.

Within your proof, you are free to use the following lemma about substitution:

$$\Gamma \vdash u : \tau_0 \land \Gamma[x : \tau_0] \vdash t : \tau \implies \Gamma \vdash t[u/x] : \tau \quad (1)$$

Exercise 3 (T) (Let-Polymorphism)

Give a derivation tree for the following statement, and so determine the type $\tau$:

$$[z : \tau_0] \vdash \text{let } x = \lambda y z. z y y \text{ in } x (x z) : \tau$$

Exercise 4 (T) (Recursive Let)

Recursive let expressions are one way (besides $Y$-combinators) to add recursion to $\lambda^\to$.

$$t ::= x \mid (t_1 t_2) \mid (\lambda x. t) \mid \text{letrec } x = t_1 \text{ in } t_2$$

a) Modify the standard typing rule for “let” to create a suitable rule for “letrec”.

b) Give a derivation tree for the following statement, and so determine the type $\tau$:

$$[] \vdash \text{letrec } x = \lambda y. x (x y) \text{ in } x x : \tau$$

Exercise 5 (T) (Normal Form)

Show that every type-correct $\lambda^\to$-term has a $\beta$-normal form.