$\boxed{\text{CIS 670 Homework 2, Due Oct 9, 2002}}$

1. Finish the last two cases of the termination proof for System F.

(Optional) Recall the encoding of sums in F:

$$\begin{array}{l} \tau_1 + \tau_2 = \forall a.(\tau_1 \to a) \to (\tau_2 \to a) \to a \\ \mathit{inl} = \lambda x : \tau_1.\Lambda\alpha.\lambda f.\lambda g.fx \\ \mathit{inr} = \lambda x : \tau_2.\Lambda\alpha.\lambda f.\lambda g.gx \\ \mathit{case}\ \ e_1\ e_2\ e_3 = e_1\ [\tau]\ e_2\ e_3 \end{array}$$

Show that:

$$\begin{array}{l} \mathit{case}(\mathit{inl}\, e)\ f\ \ g =_{\beta\eta} f\ \ e \\ \mathit{case}(\mathit{inr}\, e)\ f\ \ g =_{\beta\eta} g\ \ e \end{array}$$

Can you use parametricity to show the following identity?

$$\mathit{case}\, e\ \ \mathit{inl}\ \mathit{inr} =_{\beta\eta} e$$

I think that you can, but I don't know how.

2. Prove the parametricity theorem. Recall the following logical relation between equivalence classes of $\beta\eta$-equivalent terms.

$$\begin{array}{lcl} [\![int]\!]^{\delta_1\delta_2}_{\eta} & = & \{(i,i)\} \\ [\![\tau_1 \to \tau_2]\!]^{\delta_1\delta_2}_{\eta} & = & \{(e_1,e_2) \mid\ \vdash e_1 : \delta_1(\tau_1 \to \tau_2) \quad \vdash e_2 : \delta_2(\tau_1 \to \tau_2) \\ & & \qquad \text{for all } (e_1',e_2') \in [\![\tau_1]\!]^{\delta_1\delta_2}_{\eta}, (e_1e_1', e_2e_2') \in [\![\tau_2]\!]^{\delta_1\delta_2}_{\eta} \} \\ [\![\alpha]\!]^{\delta_1\delta_2}_{\eta} & = & \eta(\alpha) \\ [\![\forall\alpha.\tau]\!]^{\delta_1\delta_2}_{\eta} & = & \{(e_1,e_2) \mid\ \vdash e_1 : \forall\alpha.\delta_1(\tau) \quad \vdash e_2 : \forall\alpha.\delta_2(\tau), \\ & & \qquad \text{for all closed } R : \tau_1 \leftrightarrow \tau_2, \\ & & \qquad (e_1[\tau_1], e_2[\tau_2]) \in [\![\tau]\!]^{\delta_1+\{\alpha\mapsto\tau_1\}\delta_2+\{\alpha\mapsto\tau_2\}}_{\eta\{\alpha\mapsto R\}} \} \end{array}$$

Show that if $\vdash e : \tau$ then $(e,e) \in [\![\tau]\!]^{\emptyset\emptyset}_{\emptyset}$

3. Add sums and existentials to System F and extend the above proof. Be careful how you define $[\![\tau_1 + \tau_2]\!]^{\delta_1\delta_2}_{\eta}$ and $[\![\exists\alpha.\tau]\!]^{\delta_1\delta_2}_{\eta}$.

4. Parametricity can also be stated as *Representation independence* for abstract types. Here is an example from Harper's book.

   A relation $R : \tau_1 \leftrightarrow \tau_2$ is a *bisimulation* between two packages of type $\exists\alpha.\tau$: $\mathit{pack} < \tau_1, v_1 >$ and $\mathit{pack} < \tau_2, v_2 >$ if and only if $(v_1,v_2) \in [\![\tau]\!]^{\delta_1+\{\alpha\mapsto\tau_1\}\delta_2+\{\alpha\mapsto\tau_2\}}_{\eta\{\alpha\mapsto R\}}$.

   When two packages e_1, e_2 are bisimilar, they are indistinguishable by any client. That means that the execution of the client cannot depend on whether e_1 or e_2 is the provided package.

   In other words, if $\alpha, x : \tau \vdash e_c : \tau_c$, then

$$(\mathit{let} < \alpha, x >= e_1\ \mathit{in}\, e_c, \mathit{let} < \alpha, x >= e_1\ \mathit{in}\, e_c) \in [\![\tau_c]\!]^{\delta_1\delta_2}_{\eta}.$$

Given the relation: $R = \{(l, (b, f)) \mid l = \texttt{append}\ b(\texttt{rev}\ f)\}$ show that the following two implementations of queues are bisimilar. You can show this fact by showing that the implementations of all of the operations are equivalent relative to R. Note: two lists are related if all of their elements are related.

```
structure Q
struct
  type queue = int list
  val empty = nil
  fun insert (x,xs) = x::xs
  fun remove xs =
    let val (x,xs') = rev xs in (x, rev xs') end
end
```

```
structure QFB
struct
  type queue = int list * int list
  val empty = (nil, nil)
  fun insert (x, (bs,fs)) = (x::bs, fs)
  fun remove (bs, nil) = remove (nil, rev bs)
    | remove (bs, f::fs) = (f, (bs,fs))
end
```

5. Add kind polymorphism to F_ω and extend Pfenning and Lee's encoding of F to F_ω.

   Kind polymorphism works pretty much like type polymorphism.

κ	$::=$	$\ldots \mid \chi$	Kind variables
τ	$::=$	$\ldots \mid \forall^+ \chi.\tau$	Type of expressions that are kind polymorphic
e	$::=$	$\ldots \mid \Lambda\chi.e \mid e[\kappa]^+$	

   Let Θ be a kind environment, a set of valid kind variables. A kind is well formed ($\Theta \vdash \kappa$) when the free kind variables of κ are contained in Θ.

   We have the following two new rules to typecheck kind abstraction and application.

$$\frac{\Theta, \chi; \Delta; \Gamma \vdash e : \tau}{\Theta; \Delta; \Gamma \vdash \Lambda^{\chi}.e : \forall^+ \chi.\tau}$$

$$\frac{\Theta; \Delta; \Gamma \vdash e : \forall^+ \chi.\tau \Theta \vdash \kappa}{\Theta; \Delta; \Gamma \vdash e[\kappa]^+ : \tau\{\kappa/\chi\}}$$

   The operational behavior of these two terms is also straightforward.

$$\frac{}{(\Lambda\chi.e)[\kappa]^+ \to e\{\kappa/\chi\}}$$

$$\frac{e_1 \to e_2}{e_1[\kappa]^+ \to e_2[\kappa]^+}$$

   Kind polymorphism will allow you to change the type of tlam to represent the kind of the abstracted type variable.