CIS 670, Homework 6
Due 4 Dec 2002

1. Show that all polytypic specializations are well typed in System F. Let $(\tau)_n$ represent $\tau$ where each variable $\alpha$ has been replaced by the variable $\alpha_n$. Let $\phi$ be an arbitrary type constructor of $n$ arguments.

   Define:

$$
\begin{array}{lcl}
Poly\langle\star\rangle\tau_1\ldots\tau_n & = & \phi\ \tau_1\ \ldots\ \tau_n \\
Poly\langle\kappa_1 \to \kappa_n\rangle\tau_1\ldots\tau_n & = & \forall\alpha_1{:}\kappa_1.\ldots\forall\alpha_n : \kappa_1.Poly\langle\kappa_1\rangle\alpha_1\ldots\alpha_n \to Poly\langle\kappa_2\rangle(\tau_1\alpha_1)\ldots(\tau_n\alpha_n)
\end{array}
$$

$$
\begin{array}{lcl}
poly\langle\alpha\rangle & = & poly_\alpha \\
poly\langle\tau_1\tau_2\rangle & = & poly\langle\tau_1\rangle[(\tau_2)_1]\ldots[(\tau_2)_n]poly\langle\tau_2\rangle \\
poly\langle\lambda\alpha{:}\kappa.\tau\rangle & = & \Lambda\alpha_1{:}\kappa.\ldots\Lambda\alpha_n{:}\kappa.\lambda x_\alpha{:}Poly\langle\kappa\rangle\alpha_1\ldots\alpha_n.poly\langle\tau\rangle
\end{array}
$$

   Show that if for all constants $c$ of kind $\kappa_c$ that if $poly\langle c\rangle : Poly\langle\kappa_c\rangle c\ldots c$ then for all type constructors $\tau$ of kind $\kappa_\tau$,

$$\vdash poly\langle\tau\rangle : Poly\langle\kappa_\tau\rangle\tau\ldots\tau$$

2. Add a branch for iso-recursive types $\mu_k$ to the above definition, with the equivalence rule

$$\frac{\Gamma \vdash \tau : \kappa \to \kappa}{\Gamma \vdash \mu_k\tau = \tau(\mu_k\tau) : \kappa}$$

   and extend the above proof. What is the difference between a language with iso-recursive types and one with equi-recursive types with respect to polytypic programming?

3. Change the above definition of poly so that it does not rely on such syntactic tricks as $x_\alpha$ and $(\tau)_n$.

4. Define map, flatten, size and zip in Generic Haskell.

5. Define the creation function for square matrices (from Homework 3) in Generic Haskell. Informally show that the specialization of map (from the last problem) to square matrices matches the map in Homework 3. Extra Credit: can you also define subscript and update as generic functions?

   Hint: define your own three-sums instead of using the built-in sums. This will give you a place to specialize your generic definition of create.

```
newtype Empty a = E ()
newtype Id a = I a
newtype Pair v w a = P (v a, w a)
data ThreeWay a b c = Zero a | Odd b | Even c
data Squareh v w a = ThreeWay (v (v a))
                              (Squareh v (Pair w w) a)
                              (Squareh (Pair v w) (Pair w w) a)
type Square a = Squareh Empty Id a
```

6. Explain why you can't define map in $\lambda_i^{ML}$.