$\boxed{\text{CIS 670 Homework 3, Due Oct 23, 2002}}$

Nested datatypes For this problem, you should download an implementation of Haskell to try out your solution. (It won't work in ML because you need polymorphic recursion.)

Read, Chris Okasaki "From fast exponentiation to square matrices: an adventure in types" *except* for Appendix A.

Your job is to implement the functions in Appendix A.

1. Write a function to create a $n \times n$ matrix.
   `create :: Int -> Square a`
2. Write a function to apply an update function to an individual element of a matrix.
   `update:: (Int,Int) -> (a->a) -> Square a -> Square a`
3. Write a function to map a function across the elements of a matrix.
   `mapMat :: (a -> b) -> Square a -> Square b`

Type inference Do TAPL exercises 22.3.9, 22.3.10 and 22.4.6. For more fun, implement them in Haskell instead of ML.

Modeling references Here is one way to add references to the polymorphic lambda calculus.

We have a new value form, labels, ℓ, that refer to positions in an abstract heap. These labels are like variables, except they may occur in closed values.

$$\begin{array}{llcl}
\textit{(Heaps)} & H & ::= & H + \{\ell \mapsto v\} \mid \cdot \\
\textit{(Heap Types)} & \Psi & ::= & \Psi + \{\ell : \tau\} \mid \cdot \\
\textit{(Types)} & \tau & ::= & \ldots \mid \textit{ref}\ \tau \\
\textit{(Terms)} & e & ::= & \ldots \mid \ell \mid \textit{new}\ e \mid !e \mid e_1 := e_2 \\
\textit{(Values)} & v & ::= & \ldots \mid \ell \\
\textit{(Machines)} & M & ::= & (H, e)
\end{array}$$

Static semantics First we need to show how heaps are typed:

Heap types are well formed (written $\vdash \Psi$) if all types in the range are well formed (i.e. for all ℓ in domain ofΨ, $\emptyset \vdash \Psi(\ell)$).

Heaps are well typed (written $\Psi \vdash H$) if all values are closed and well-typed. For all ℓ in domain H, $\Psi\emptyset\emptyset \vdash H(l) : \Psi(l)$. (Note that a closed value may include labels to the heap. These are described by Ψ.) We need to add heap types to the context to determine if expressions are well-typed.

$$\frac{}{\Psi\Delta\Gamma \vdash \ell : \textit{ref}\,(\Psi(\ell))}$$

$$\frac{\Psi\Delta\Gamma \vdash e : \tau}{\Psi\Delta\Gamma \vdash \mathit{new}\, e : \mathit{ref}\ \tau}$$

$$\frac{\Psi\Delta\Gamma \vdash e : \mathit{ref}\ \tau}{\Psi\Delta\Gamma \vdash !e : \tau}$$

$$\frac{\Psi\Delta\Gamma \vdash e_1 : \mathit{ref}\ \tau \qquad \Psi\Delta\Gamma \vdash e_2 : \tau}{\Psi\Delta\Gamma \vdash e_1 := e_2 : \tau}$$

Finally, a machine (H, e) is well-formed if both the heap and the expression are well-formed.

$$\frac{\Psi \vdash H \qquad \Psi\emptyset\emptyset \vdash e : \tau}{\vdash (H, e) : \tau}$$

Operational semantics The operational semantics is defined over machine states: Both the current heap and the current expression evaluate together. Standard lambda-calculus operations don't change the heap, so I haven't listed them below.

The operations that do involve the heap are:

allocate

$$(H, \mathit{new}\, v) \mapsto (H + \{\ell \mapsto v\}, l)$$

dereference

$$(H, \ell) \mapsto (H, H(\ell))$$

assign

$$(H + \{\ell \mapsto v_1\}, \ell := v_2) \mapsto (H + \{\ell \mapsto v_2\}, v_2)$$

congruence

$$\frac{(H, e) \mapsto (H', e')}{(H, \mathit{new}\, e) \mapsto (H, \mathit{new}\, e')}$$

$$\frac{(H, e) \mapsto (H', e')}{(H, e := e_2) \mapsto (H', e' := e_2)}$$

$$\frac{(H, e) \mapsto (H', e')}{(H, \ell := e) \mapsto (H', l := e')}$$

- Prove type safety for this language. For each lemma, you only need to show the cases dealing with references.
- Show an infinite loop in this language.