1 Abstract machine N [60 pts]

In this problem, we design an abstract machine N based on the call-by-need reduction strategy. The call-by-need reduction strategy is a variant of the call-by-name strategy: it reduces a function application without evaluating the argument, but is also designed so that it never evaluates the same argument more than once. To this end, it delays the evaluation of the argument until the result is needed. Once the argument is fully evaluated, it stores the result in the heap so that when it needs the argument again, it does not need to repeat the same evaluation.

The heap stores two different kinds of objects: *delayed expressions* and *computed values*. When the machine attempts to evaluate a function application, it first allocates a new delayed expression for the argument in the heap and then proceeds to reducing the function application. When the machine later evaluates the variable bound to the argument for the first time, it retrieves the actual argument from the delayed expression to evaluate it. Then it replaces the delayed expression with a computed value in the heap so that all subsequent references to the same variable can directly use the result without repeating the same evaluation.

The abstract machine N uses the following definitions:

<table>
<thead>
<tr>
<th>Type</th>
<th>Expression</th>
<th>Value</th>
<th>Stored Value</th>
<th>Heap</th>
<th>Environment</th>
<th>Frame</th>
<th>Stack</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$x$</td>
<td>$e$</td>
<td>$v$</td>
<td>$h$</td>
<td>$\eta$</td>
<td>$\phi$</td>
<td>$\sigma$</td>
<td>$s$</td>
</tr>
<tr>
<td>$P$</td>
<td>$A \to A$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A value $v$ contains the result of evaluating an expression. A stored value $sv$ is an object to be stored in the heap $h$, and thus is either a delayed expression or a computed value.

In the definition of state $s$:

- $h \parallel \sigma \triangleright e \triangleleft \eta$ means that the machine with heap $h$ and stack $\sigma$ is currently analyzing $e$ under environment $\eta$.
- $h \parallel \sigma \triangleright v$ means that the machine with heap $h$ and stack $\sigma$ is currently returning $v$.

The reduction judgment for the abstract machine N is as follows:

$s \rightarrow_N s'$ ⇔ *the machine makes a transition from state s to another state s'*

To define the operational semantics, we use a couple of auxiliary functions. First we define the function $\text{dom}(h)$ which returns the set of all locations in a given heap $h$; we also use $[l \leftrightarrow sv]h$ for updating the contents of $l$ in $h$ with $sv$:

\[
\begin{align*}
\text{dom}(\cdot) &= \emptyset \\
\text{dom}(h,l \leftrightarrow sv) &= \text{dom}(h) \cup \{l\} \\
[l \leftrightarrow sv'](h,l \leftrightarrow sv) &= h,l \leftrightarrow sv'
\end{align*}
\]

Complete the definitions of value $v$, stored value $sv$, and frame $\phi$. Then define transition rules for the abstract machine N. You may introduce as many transition rules as you need.

(Definitions)

<table>
<thead>
<tr>
<th>Type</th>
<th>Expression</th>
<th>Value</th>
<th>Stored Value</th>
<th>Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v$</td>
<td>$[\eta, \lambda x : A.e]$</td>
<td></td>
<td>$\text{computed}(v) \mid \text{delayed}(e, \eta)$</td>
<td>$\triangleright \eta e \mid [l]$</td>
</tr>
</tbody>
</table>
(Transition rules)

\[
\frac{x \leftrightarrow l \in \eta \quad l \leftrightarrow \text{computed}(v) \in h}{h \parallel \sigma \triangleright x @ \eta \mapsto_N h \parallel \sigma \triangleleft v}
\]

\[
\frac{x \leftrightarrow l \in \eta \quad l \leftrightarrow \text{delayed}(e, \eta') \in h}{h \parallel \sigma \triangleright x @ \eta \mapsto_N h \parallel \sigma; [l] \triangleright e @ \eta'}
\]

\[
\frac{h \parallel \sigma; [l] \triangleleft v \mapsto_N [l \leftrightarrow \text{computed}(v)]h \parallel \sigma \triangleleft v}{h \parallel \sigma \triangleright e_1 e_2 @ \eta \mapsto_N h \parallel \sigma; \square_\eta e_2 \triangleright e_1 @ \eta}
\]

\[
\frac{h \parallel \sigma \triangleright \lambda x : A.e @ \eta \mapsto_N h \parallel \sigma \triangleleft [\eta, \lambda x : A.e]}{l \notin \text{dom}(h)
\]

\[
\frac{h \parallel \sigma; \square_\eta e_2 \triangleleft [\eta', \lambda x : A.e] \mapsto_N h, l \leftrightarrow \text{delayed}(e_2, \eta) \parallel \sigma \triangleright e @ \eta', x \leftrightarrow l}{h \parallel \sigma \triangleright e e' \mapsto_N e' @ \eta}
\]

2 System F

Question 1. [10 points] Complete the abstract syntax of System F:

<table>
<thead>
<tr>
<th>type</th>
<th>$A ::= A \to A \mid \alpha \mid \forall \alpha.A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>$e ::= x \mid \lambda x : A.e \mid e e \mid \Lambda \alpha.e \mid e [A]$</td>
</tr>
<tr>
<td>value</td>
<td>$v ::= \lambda x : A.e \mid \Lambda \alpha.e$</td>
</tr>
</tbody>
</table>

Question 2. [30 points] We use the following definition of ordered typing contexts:

| typing context | $\Gamma ::= \cdot \mid \Gamma, x : A \mid \Gamma, \alpha$ type |

We assume that variables and type variables in a typing context are all distinct. We consider a type variable $\alpha$ as valid only if its type declaration appears to its left. We use $\Gamma \vdash A$ type for the type judgment which means that $A$ is a valid type with respect to typing context $\Gamma$. We use $\Gamma \vdash e : A$ for the typing judgment.

Write the rules for the type judgment:

\[
\frac{\Gamma \vdash A \text{ type} \quad \Gamma \vdash B \text{ type} \quad \text{Ty} \to}{\Gamma \vdash A \to B \text{ type}} \quad \frac{\alpha \text{ type} \in \Gamma \quad \text{TyVar}}{\Gamma, \alpha \vdash A \text{ type}} \quad \frac{\Gamma, \alpha \vdash A \text{ type}}{\Gamma \vdash \forall \alpha.A \text{ type}} \quad \frac{\text{Ty} \to \alpha}{\Gamma \vdash \forall \alpha.A \text{ type}}
\]

Write the typing rules for the typing judgment $\Gamma \vdash e : A$:

\[
\frac{x : A \in \Gamma \quad \text{Var}}{\Gamma \vdash x : A} \quad \frac{\Gamma, x : A \vdash e : B \quad \text{Var}}{\Gamma \vdash \lambda x : A.e : A \to B} \quad \frac{\Gamma \vdash e : A \to B \quad \Gamma \vdash e' : A \quad \text{]->E}}{\Gamma \vdash e e' : A} \quad \frac{\Gamma, \alpha \vdash e : A \quad \text{forall}}{\Gamma \vdash \Lambda \alpha.e : \forall \alpha.A} \quad \frac{\Gamma \vdash e : \forall \alpha.B \quad \Gamma \vdash A \text{ type}}{\Gamma \vdash e [A] : [A/\alpha]B} \quad \frac{\text{\forall}E}{\Gamma \vdash e : \forall \alpha.B}
\]