1 Mutual references [40 points]

For the simply typed $\lambda$-calculus extended with references, we use the following definitions and judgments:

<table>
<thead>
<tr>
<th>type $A$</th>
<th>::=</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression $e$</td>
<td>::=</td>
</tr>
<tr>
<td>value $v$</td>
<td>::=</td>
</tr>
</tbody>
</table>
| store $\psi$ | ::= $\cdot | \psi, l \mapsto v$

store typing context $\Psi$ ::= $\cdot | \Psi, l \mapsto A$

$e | \psi \mapsto e' | \psi' \iff e$ with store $\psi$ reduces to $e'$ with store $\psi'$

$\Gamma | \Psi \vdash e : A \iff$ expression $e$ has type $A$ under typing context $\Gamma$ and store typing context $\Psi$

Question 1. [10 points] We use a new judgment $\psi :: \Psi$ to mean that $\psi$ is well-typed with $\Psi$. Give the inference for the judgment $\psi :: \Psi$ as given in the course notes. You may use $\text{dom}(\psi)$ for the domain of $\psi$, and $\text{dom}(\Psi)$ for the domain of $\Psi$.

$$\frac{\text{dom}(\Psi) = \text{dom}(\psi) \cdot \Psi \vdash \psi(l) : \Psi(l) \text{ for every } l \in \text{dom}(\psi)}{\psi :: \Psi} \quad \text{Store}$$

Question 2. [30 points] Type safety consists of two theorems: progress and type preservation. State the two theorems:

**Theorem 1.1 (Progress).** Suppose that expression $e$ satisfies $\cdot | \Psi \vdash e : A$ for some store typing context $\Psi$ and type $A$. Then either:

1. $e$ is a value
2. for any store $\psi$ such that $\psi :: \Psi$

there exist some expression $e'$ and store $\psi'$ such that $e | \psi \mapsto e' | \psi'$.

**Theorem 1.2 (Type preservation).** Suppose $\begin{cases} \Gamma | \Psi \vdash e : A \\ \psi :: \Psi \\ e | \psi \mapsto e' | \psi' \end{cases}$. Then there exists a store typing context $\Psi'$ such that

$$\begin{cases} \Gamma | \Psi' \vdash e' : A \\ \Psi \subset \Psi' \\ \psi' :: \Psi' \end{cases}.$$
2 Evaluation contexts [30 points]

Consider the following fragment of the simply-typed λ-calculus:

\[
\begin{align*}
\text{type} & \quad A ::= P \mid A \to A \\
\text{base type} & \quad P ::= \text{bool} \\
\text{expression} & \quad e ::= x \mid \lambda x : A.e \mid e \ e \mid \text{true} \mid \text{false} \mid \text{if} \ e \ \text{then} \ e \ \text{else} \ e \end{align*}
\]

**Question 3. [10 pts]** Give the definition of evaluation contexts for the call-by-value strategy.

\[
\begin{align*}
\text{evaluation context} & \quad \kappa ::= \Box \mid \kappa \ e \\
& \quad (\lambda x : A.e) \ \kappa \mid \text{if} \ \kappa \ \text{then} \ e \ \text{else} \ e \\
& \quad \kappa \ e \\n& \quad \kappa = \Box \\
& \quad \kappa \neq \Box
\end{align*}
\]

**Question 4. [10 pts]** Under the call-by-value strategy, give an expression \(e\) such that

- \(e = \kappa[e']\) where \(e'\) is the redex and \(\kappa \neq \Box\), and
- \(e\) reduces to \(e_0\) that is decomposed to \(\kappa'[e'']\) where \(e''\) is the redex for the next reduction and \(\kappa \neq \kappa'\).

\[
(\lambda x : \text{bool} \to \text{bool} \ x) \ (\lambda y : \text{bool} \ y) \ \text{true}
\]

**Question 5. [10 pts]** Now consider the following definitions for simply-typed λ-calculus extended with mutable references:

\[
\begin{align*}
\text{type} & \quad A ::= P \mid A \to A \mid \text{unit} \mid \text{ref} \ A \\
\text{expression} & \quad e ::= x \mid \lambda x : A.e \mid e \ e \mid () \mid \text{ref} \ e \mid \! \ e \mid e ::= e \mid l \\
\text{value} & \quad v ::= \lambda x : A.e \mid () \mid l
\end{align*}
\]

Complete the definition of the evaluation context \(\kappa\) that corresponds to the operational semantics based on the call-by-value reduction strategy:

\[
\begin{align*}
\text{evaluation context} & \quad \kappa ::= \Box \mid \kappa \ e \\
& \quad (\lambda x : A.e) \ \kappa \mid \text{ref} \ \kappa \mid \! \kappa \mid \kappa := e \mid l := \kappa
\end{align*}
\]

3 A weird reduction strategy [30 pts]

Consider the following fragment of the simply typed λ-calculus:

\[
\begin{align*}
\text{type} & \quad A ::= P \mid A \to A \\
\text{base type} & \quad P \\
\text{expression} & \quad e ::= x \mid \lambda x : A.e \mid e \ e \\
\text{value} & \quad v ::= \lambda x : A.e
\end{align*}
\]

We will develop a weird strategy specified as follows:

- Given an application \(e_1 \ e_2\), we first reduce \(e_2\).
- After reducing \(e_2\) to a value, we reduce \(e_1\).
- When \(e_1\) reduces to a λ-abstraction, we apply the β-reduction.

**Question 1. [15 points]** Give the reduction rules for the reduction judgment \(e \mapsto e'\) under the weird reduction strategy. You need three reduction rules.

\[
\begin{align*}
\frac{}{e_2 \mapsto e'_2} & \quad \frac{}{e_1 \ e_2 \mapsto e_1 \ e'_2} & \quad \frac{}{e_1 \mapsto e'_1} \\
\frac{}{e_1 \ v \mapsto e'_1 \ v} & \quad \frac{}{(\lambda x : A.e) \ v \mapsto [v/x]e}
\end{align*}
\]

**Question 2. [15 points]** Give the definition of evaluation contexts corresponding to the weird reduction strategy:

\[
\begin{align*}
\text{evaluation context} & \quad \kappa ::= \Box \mid e \ \kappa \mid \kappa \ v
\end{align*}
\]