1. [10 points] Recall that the small-step semantics we used for pure untyped lambda calculus encoded the Call-By-Value evaluation strategy (CBV). The Lazy evaluation strategy is encoded by an alternative semantics for the calculus, defined by the following inference rules:

\[
\frac{t_1 \rightarrow t'_1}{t_1 t_2 \rightarrow t'_1 t_2} \quad \text{E-App'}
\]

\[
(\lambda x. t_{11}) t_2 \rightarrow [x \mapsto t_2]t_{11} \quad \text{E-Abs'}
\]

Values in the lazy strategy are lambda abstractions, i.e. the same as those in the CBV strategy: terms of the form $\lambda x.t$.

Give the big-step semantics for pure untyped lambda calculus for the lazy evaluation strategy.

2. [26 points total] For this question, consider the extensions of simply-typed lambda calculus as discussed in Chapter 11 of the book.

(a) [20 points] Give the types of the following expressions. If they are not well-typed, state that.

i. $\lambda x : \text{Unit}. \{a = 0, b = \text{succ } 0, c = x\}$

ii. $\lambda x : \text{Unit}. \text{ref } x$

iii. $\text{let } x = \{a = 0, b = \text{succ } 0\} \text{ in } x.b$

iv. $\text{if}(\text{iszero } 0, \{a = 0, b = \text{succ } 0\}, 0)$

v. $\text{let } y = (\text{let } x = 0 \text{ in } \text{succ } x) \text{ in } \text{ref } y$

vi. $\lambda x : \text{Nat.} \text{(ref } x) := \text{succ } x$

vii. $\lambda x : \text{Nat.} \text{let } y = \text{ref } x \text{ in } y := \text{iszero } x$

viii. $\text{letrec } x = (\lambda y : \text{Nat.} \text{if}(\text{iszero } y, \text{true, } (x (\text{pred } y)))) \text{ in } x$

(b) [6 points] Consider the following type defined in OCAML:

```ocaml
type Stuff = Blip of int
  | Glob of bool
```

and consider the following OCAML expression that uses this type:

```ocaml
function x ->
  match x with
  Blip y -> (y=0)
  | Glob z -> z
```

Write the above expression in the extended lambda calculus of Chapter 11.

3. [18 points total] A pure untyped lambda term $t$ (Ch. 5) is said to be well-formed if (i) its free variable and bound variable sets are disjoint: i.e. names of all bound variables are different from that of any free variable, and (ii) every subterm of $t$ is also well-formed.

(a) [6 points] Formally define well-formed lambda terms using an inductive definition. More specifically, give an inductive definition of a function $WF$ whose domain is the set of all lambda terms and whose range is Boolean, such that $WF$ maps $t$ to $\text{true}$ iff $t$ is well-formed. You may assume the definitions of the set of free variables of $t$ (denoted by $FV(t)$) and the set of all variables of $t$ (denoted by $\text{Vars}(t)$). If you need additional auxiliary definitions, make sure those are defined inductively too.
(b) [12 points] Show that single-step evaluation under CBV preserves well-formedness of terms. That is, if \( t \) is well-formed and \( t \rightarrow t' \), then \( t' \) is well-formed.

4. [20 points] Write expressions in lambda calculus extended with let, tuples and references that, when evaluated in an empty store result in the following stores.

\[
\begin{align*}
(a) & \{ l_1 \mapsto 0 \} \\
(b) & \{ l_1 \mapsto 0, l_2 \mapsto l_1 \} \\
(c) & \{ l_1 \mapsto 0, l_2 \mapsto \{ l_1, \text{true} \} \} \\
(d) & \{ l_1 \mapsto 0, l_2 \mapsto \{ l_1, l_1 \} \} \\
(e) & \{ l_1 \mapsto 0, l_2 \mapsto \{ l_1, l_2 \} \} \\
(f) & \{ l_1 \mapsto \{ l_2, l_1 \}, l_2 \mapsto \{ l_1, l_2 \} \} \\
(g) & \{ l_1 \mapsto \lambda x : \text{Nat.} \cdot x, l_2 \mapsto \lambda x : \text{Nat.} \cdot (\text{let} \ l_1 \mapsto x \ \text{in} \ x) \} \\
(h) & \{ l_1 \mapsto \lambda x : \text{Nat.} \cdot (\text{let} \ l_2 \mapsto x \ \text{in} \ x) \}
\end{align*}
\]

5. [10 points] Let \( \text{Square} <: \text{Rectangle} <: \text{Polygon} \) be a subtype relation among base types \( \text{Square} \), \( \text{Rectangle} \) and \( \text{Polygon} \). Let \( f, g \) and \( h \) be a terms in typed lambda calculus with type \( \text{Rectangle} \rightarrow \text{Rectangle} \).

(a) [2 points] What is the type of \( h (f \ g) \)?

(b) [8 points] Let \( f' \) be a term in typed lambda calculus such that \( h (f' \ g) \) is well-typed. List the possible types of \( f' \).

For this question, consider only the base types (e.g. \( \text{Square} \)) and function types (e.g. \( \text{Square} \rightarrow \text{Rectangle}, \text{Square} \rightarrow \text{Square} \rightarrow \text{Polygon}, \text{etc.} \))

6. [16 points total]

(a) [6 points] Write a predicate \( \text{find}(L,K,V) \) that, given a list \( L \) of key-value pairs, and a key \( K \), succeeds with binding \( V \) to the value associated with the given key. For instance, \( \text{find}([(a,1), (b,2), (c,3)], b, Q) \) should succeed with \( Q=b \). If the given key does not appear in the list, \( \text{find} \) should fail. For instance, \( \text{find}([(a,1), (c,3)], b) \) should fail.

(b) [10 points] Find the most general unifier for the following pairs of terms. If a pair of terms do not have a unifier, state that.

In the following, we follow Prolog’s convention and use identifiers beginning with upper-case letters to denote variables.

i. \( f(a) = f(Y) \)

ii. \( f(g(X), X) = f(Y,a) \)

iii. \( \text{arrow}(A,B) = \text{arrow}(B,A) \)

iv. \( \text{arrow}(A,B) = A \)

v. \( \text{arrow}(\text{arrow}(A,B),A) = \text{arrow}(X,B) \)

7. [10 points] OCAML has a “while-do” construct of the form “\text{while} \ e_1 \ \text{do} \ e_2 \ \text{done}” where \( e_1, e_2 \) are OCAML expressions. The meaning of while expressions is similar to that in imperative languages: if \( e_1 \) evaluates to \text{true} then \( e_2 \) is evaluated, followed by looping back to the evaluation of \( e_1 \).

For this problem, consider further extending the lambda calculus with references (assume all extensions of Chap. 11 as needed, as well as the extensions in Chap. 13) with a “while” term with the following syntax:

\[
t := \ldots \text{existing terms} \\
| \text{while}(t,t)
\]

Give the additional evaluation rules and typing rules for this extension.

You may also, alternatively, treat \text{while} as a derived form. Then give the definition of the derived form.