Problems: Axiomatic Semantics

Question 1

Using Hoare logic, give a proof that the following C code fragment implements a swap of `x` and `y`:

```
x = x ^ y;
y = x ^ y;
x = x ^ y;
```

Question 2

Suppose that we change the axiom of assignment to use the strongest post-condition:

$$\overline{\{P\}\texttt{x = e;}\{\exists v.x = (e[v/x]) \wedge P[v/x]\}}$$

Using Hoare logic, give a proof that the following C code fragment

```
temp = x;
x = y;
y = temp;
```

implements a swap of `x` and `y`.

Question 3

Using Hoare logic, give a proof of the following Hoare triple:

```
{0 ≤ x ∧ 0 < y}
r = x;
q = 0;
while (r >= y) {
    r = r - y;
    q = q + 1;
}
{x = y * q + r ∧ 0 ≤ r < y}
```

Question 4

This question is about the following code, which attempts to implement swap using the pointer variables px and py (and the integer temporary temp):

```
temp = *px;
*px = *py;
*py = temp;
```

The code fragment attempts to swap the values in the locations pointed to by px and py; however, it does not work in all situations.

Part (a)

Show a pair of before and after configurations of memory that cause the above code fragment to be unsuccessful due to aliasing, along with the intermediate configurations that arise after each statement in the above code fragment. In each configuration, show px, py, the memory locations that px and py point to, and the values in those memory locations.

Part (b)

For this part, we will use symbolic reasoning to give a formula for *all* situations in which aliasing causes the above code fragment to fail to perform a swap. First, we introduce auxiliary variables X and Y to record the values of *px and *py, respectively, in the initial state, as shown below:

```
X = *px;
Y = *py;
temp = *px;
*px = *py;
*py = temp;
```

Call the fragment above buggy-swap.

The post-condition $\{(\mathtt{X} \neq \mathtt{*py}) \vee (\mathtt{Y} \neq \mathtt{*px})\}$ characterizes when the code fails to perform a swap. To simplify matters, we will just work with one disjunct of the post-condition: $\mathtt{X} \neq \mathtt{*py}$.

Warning: The formulas that arise in this problem become quite complicated, so to simplify matters, assume that X, Y, and temp are not addressible locations (e.g., they are like registers in most machine-code instruction sets), and hence they can be modeled in the logic with constants $c_{\mathtt{X}}$, $c_{\mathtt{Y}}$, and $c_{\mathtt{temp}}$ that hold their *values*. You will have two other constants, $c_{\&\mathtt{px}}$ and $c_{\&\mathtt{py}}$, to model the *addresses* of variables px and py, respectively. Thus, the semantics of X = *px; can be stated as

$$c'_{\mathtt{X}} \hookleftarrow F_\rho(F_\rho(c_{\&\mathtt{px}}))$$

and the semantics of `temp = *px;` can be stated as

$$c'_{\texttt{temp}} \hookleftarrow F_\rho(F_\rho(c_{\&\texttt{px}})),$$

but the semantics of `*px = *py;` is

$$F'_\rho \hookleftarrow F_\rho[F_\rho(c_{\&\texttt{px}}) \mapsto F_\rho(F_\rho(c_{\&\texttt{py}}))].$$

End Warning.

Using the Cartwright-Oppen technique (substitution using the semantics expressed in logic, but *no simplification of function-updates to if-then-else expressions*), give the formula for $\mathcal{WLP}(\texttt{buggy-swap}, \texttt{X} \neq \texttt{*py})$—i.e., for

$$\mathcal{WLP}(\texttt{buggy-swap}, c_{\texttt{X}} \neq F_\rho(F_\rho(c_{\&\texttt{py}}))).$$