CS704: Assignment 3

The first question is adapted from Nielsen and Nielsen.

1 Operational Semantics

In class, we defined the operational semantics of a simple imperative programming language with while loops. In this question, you will extend the language and its semantics.

A Suppose we extend the language with a `repeat` P `until` b statement, where b is a Boolean expression and P is a program. Provide the rule(s) defining the big-step (natural) semantics of the repeat-until statement—i.e., define the $\rightarrow$ relation. Your semantics should model the intuitive meaning of a repeat-until statement: that P is repeatedly executed until b is *true*, in which case we exit the loop.

B Prove that for all programs P and Boolean expressions b, the programs

`repeat` P `until` b

and

P`; if` b `then skip else (repeat` P `until` b`)`

are equivalent.

Recall that two programs P_1, P_2 are equivalent iff for all states s, s', we have

$$\langle P_1, s\rangle \rightarrow s' \textit{ iff } \langle P_2, s\rangle \rightarrow s'$$

C As above, provide the semantics for a for-loop construct of the form

`for x =` a `to` a' `do` P

where a and a' are arithmetic expressions.

2 Hoare Logic

In this question, all variables hold real-world integers—i.e., don't worry about machine arithmetic and overflows.

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

$\{x \geq 0 \land y > 0\}$

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

$\{x = y * q + r \land 0 \leq r < y\}$

You need only provide an annotation of the form $\{P\}$ for every location in the program; you do not need to show a derivation tree. Accompany your answer with an English description.

B Using Hoare logic, give a proof that the following sequence of statements swaps the values of x and y.

```
x = x + y;
y = x - y;
x = x - y;
```

You may use auxiliary variables to denote the initial/final values of x and y. Again, you need only supply annotations of the form $\{P\}$ for every location along the program. Accompany your answer with an English description.

C Consider the following Hoare triple.

$\{true\}$

```
x = 10;
y = 10;

while (x + y > 0)
  x = x - 1;
  y = y - 1;
  z = x + y;
```

$\{z = 0\}$

Prove that the Hoare triple is valid by giving an inductive loop invariant. Show that your answer is indeed an inductive loop invariant.

3 Type Inference

In class, we used Robinson's unification algorithm to solve a set of type equations of the form $\{S_i = T_i\}_{i \in 1 \ldots n}$. In cases where no principal unifier exists, the algorithm returns *fail*.

For completeness, we supply the unification algorithm below, in the style of TAPL, as covered in lecture. The input to the algorithm is a set C of equations of the form $S = T$. In the conditionals, we use $S = X$ to mean that S is a single variable X. Similarly, $S = S_1 \to S_2$ means that S is of the form $S_1 \to S_2$, where S_i could be variables, types, or more complex expressions.

Algorithm 1 Unification Algorithm

procedure $\textsc{unify}(C)$
    if $C = \emptyset$ **then return** [ ]
    else
        let $\{S = T\} \cup C' = C$
        if $S = T$ **then** $\textsc{unify}$(C')
        else if $S = X$ and $X \notin FV(T)$ **then**
            return $\textsc{unify}([X \mapsto T]C') \circ [X \mapsto T]$
        else if $T = X$ and $X \notin FV(S)$ **then**
            return $\textsc{unify}([X \mapsto S]C') \circ [X \mapsto S]$
        else if $S = S_1 \to S_2$ and $T = T_1 \to T_2$ **then**
            return $\textsc{unify}(C' \cup \{S_1 = T_1, S_2 = T_2\})$
        else
            return *fail*

A Provide an example set of equations where the algorithm fails. Ensure that your equations *do not include* cases where (1) a variable X appears in both S_i and T_i, or (2) S_i is a function type and T_i is not (or vice versa). In other words, these cases ensure that you do not give a trivial answer where the algorithm immediately fails when it considers such equation.

B Prove that $\textsc{unify}$ always terminates.