Suggested Solutions for Homework Assignment #4

We assume the binding powers of the logical connectives and the entailment symbol decrease in this order: $\neg$, $\{\forall, \exists\}$, $\{\wedge, \vee\}$, $\rightarrow$, $\leftrightarrow$, $\vdash$.

1. Prove that the following annotated program segments are correct:

(a) (10 points)

$\{true\}$
if $x < y$ **then** $x, y := y, x$ **fi**
$\{x \geq y\}$

Solution.

$$\dfrac{\dfrac{\dfrac{\text{pred. calculus + algebra}}{true \wedge x < y \rightarrow y \geq x} \quad \dfrac{}{\{\ y \geq x\ \}\ x, y := y, x\ \{\ x \geq y\ \}}\text{(Assign)}}{\{\ true \wedge x < y\ \}\ x, y := y, x\ \{\ x \geq y\ \}}\text{(SP)} \quad \dfrac{\text{pred. calculus + algebra}}{true \wedge \neg(x < y) \rightarrow x \geq y}}{\{\ true\ \}\ \textbf{if}\ x < y\ \textbf{then}\ x, y := y, x\ \textbf{fi}\ \{\ x \geq y\ \}}\text{(If-Then)}$$

□

(b) (10 points)

$\{g = 0 \wedge p = n \wedge n \geq 1\}$
while $p \geq 2$ **do**
  $g, p := g + 1, p - 1$
od
$\{g = n - 1\}$

Solution.

$$\dfrac{\dfrac{\text{pred. calculus + algebra}}{g = 0 \wedge p = n \wedge n = 1 \rightarrow p > 0 \wedge p + g = n} \quad \alpha \quad \dfrac{\text{pred. calculus + algebra}}{p > 0 \wedge p + g = n \wedge \neg(p \geq 2) \rightarrow g = n - 1}}{\{\ g = 0 \wedge p = n \wedge n = 1\ \}\ \textbf{while}\ p \geq 2\ \textbf{do}\ g, p := g - 1, p + 1\ \textbf{od}\ \{\ g = n - 1\ \}}\text{(Consequence)}$$

α :

$$\dfrac{\dfrac{\beta \quad \dfrac{}{\{\ p + 1 > 0 \wedge (p + 1) + (g - 1) = n\ \}\ g, p := g - 1, p + 1\ \{\ p > 0 \wedge p + g = n\ \}}\text{(Assign)}}{\{\ p > 0 \wedge p + g = n \wedge p \geq 2\ \}\ g, p := g - 1, p + 1\ \{\ p > 0 \wedge p + g = n\ \}}\text{(SP)}}{\{\ p > 0 \wedge p + g = n\ \}\ \textbf{while}\ p \geq 2\ \textbf{do}\ g, p := g - 1, p + 1\ \textbf{od}\ \{\ p > 0 \wedge p + g = n \wedge \neg(p \geq 2)\ \}}\text{(while)}$$

β :

$$\dfrac{\text{pred. calculus + algebra}}{p > 0 \wedge p + g = n \wedge p \geq 2 \rightarrow p + 1 > 0 \wedge (p + 1) + (g - 1) = n}$$

□

(c) (20 points) For this program, prove its total correctness.

$\{y > 0 \land (x \equiv m \pmod{y})\}$
while $x \geq y$ **do**
  $x := x - y$
od
$\{(x \equiv m \pmod{y}) \land x < y\}$

Solution.

$$\dfrac{\alpha \qquad \dfrac{\text{pred. calculus + algebra}}{y > 0 \land (x \equiv m \pmod{y}) \land \neg(x \geq y) \to (x \equiv m \pmod{y}) \land x < y}}{\{\ y > 0 \land (x \equiv m \pmod{y})\ \}\ \textbf{while}\ x \geq y\ \textbf{do}\ x := x - y\ \textbf{od}\ \{\ (x \equiv m \pmod{y}) \land x < y\ \}}\ (\text{WP})$$

α :

$$\dfrac{\beta \qquad \gamma \qquad \dfrac{\text{pred. calculus + algebra}}{y > 0 \land (x \equiv m \pmod{y}) \land x \geq y \to x \geq 0}}{\begin{array}{c}\{\ y > 0 \land (x \equiv m \pmod{y})\ \} \\ \textbf{while}\ x \geq y\ \textbf{do}\ x := x - y\ \textbf{od} \\ \{\ y > 0 \land (x \equiv m \pmod{y}) \land \neg(x \geq y)\ \}\end{array}}\ (\text{while: simply total})$$

β :

$$\dfrac{\dfrac{\text{pred. calculus + algebra}}{\begin{array}{c} y > 0 \land (x \equiv m \pmod{y}) \land x \geq y \to \\ y > 0 \land ((x - y) \equiv m \pmod{y})\end{array}} \qquad \dfrac{}{\begin{array}{c}\{\ y > 0 \land ((x - y) \equiv m \pmod{y})\ \} \\ x := x - y \\ \{\ y > 0 \land (x \equiv m \pmod{y})\ \}\end{array}}\ (\text{Assign})}{\{\ y > 0 \land (x \equiv m \pmod{y}) \land x \geq y\ \}\ x := x - y\ \{\ y > 0 \land (x \equiv m \pmod{y})\ \}}\ (\text{SP})$$

γ :

$$\dfrac{\dfrac{\text{pred. calculus + algebra}}{y > 0 \land (x \equiv m \pmod{y}) \land x \geq y \land x = Z \to x - y < Z} \qquad \dfrac{}{\{\ x - y < Z\ \}\ x := x - y\ \{\ x < Z\ \}}\ (\text{Assign})}{\{\ y > 0 \land (x \equiv m \pmod{y}) \land x \geq y \land x = Z\ \}\ x := x - y\ \{\ x < Z\ \}}\ (\text{SP})$$

□

2. (20 points) Given a sequence x_1, x_2, $\cdots$, x_n of real numbers (not necessarily positive), a maximum subsequence x_i, x_{i+1}, $\cdots$, x_j is a subsequence of consecutive elements from the given sequence such that the sum of the numbers in the subsequence is maximum over all subsequences of consecutive elements. Below is a program that determines the sum of such a sequence.

```
Global_Max := 0;
Suffix_Max := 0;
for i := 1 to n do
   if x[i] + Suffix_Max > Global_Max then
      Suffix_Max := Suffix_Max + x[i];
      Global_Max := Suffix_Max
   else if x[i] + Suffix_Max > 0 then
      Suffix_Max := Suffix_Max + x[i]
   else Suffix_Max := 0
od;
```

Annotate the program into a *standard* proof outline, showing clearly the partial correctness of the program; a standard proof outline is essentially an annotated program where every statement is preceded by a pre-condition and the entire program is followed by a post-condition.

Solution. Let $isMS(s, x, i)$ denote that s is the sum of the maximum subsequence in $x[1..i]$ and $isMSX(s, x, i)$ denote that s is the sum of the maximum subsequence that is also a suffix in $x[1..i]$. In particular, $isMS(0, x, 0)$ and $isMSX(0, x, 0)$ both hold, as $x[1..0]$ denotes the empty sequence. To shorten formulae, we denote `Global_Max` and `Suffix_Max` respectively by G_M and S_M in all assertions.

```
// assume n ≥ 1, which is preserved by the code and will be omitted later
Global_Max := 0;
// isMS(G_M, x, 0)
Suffix_Max := 0;
// isMS(G_M, x, 0) ∧ isMSX(S_M, x, 0)
i := 1;
// inv: (1 ≤ i ≤ n + 1) ∧ isMS(G_M, x, i − 1) ∧ isMSX(S_M, x, i − 1)
while i <= n do
    // (1 ≤ i ≤ n) ∧ isMS(G_M, x, i − 1) ∧ isMSX(S_M, x, i − 1)
    if x[i] + Suffix_Max > Global_Max then
        // (1 ≤ i ≤ n) ∧ isMS(x[i] + S_M, x, i) ∧ isMSX(x[i] + S_M, x, i)
        Suffix_Max := Suffix_Max + x[i];
        // (1 ≤ i ≤ n) ∧ isMS(S_M, x, i) ∧ isMSX(S_M, x, i)
        Global_Max := Suffix_Max
    else
          // (1 ≤ i ≤ n) ∧ isMS(G_M, x, i) ∧ isMSX(S_M, x, i − 1)
          if x[i] + Suffix_Max > 0 then
              // (1 ≤ i ≤ n) ∧ isMS(G_M, x, i) ∧ isMSX(x[i] + S_M, x, i)
              Suffix_Max := Suffix_Max + x[i]
          else
              // (1 ≤ i ≤ n) ∧ isMS(G_M, x, i) ∧ isMSX(0, x, i)
              Suffix_Max := 0;
          fi
    fi;
    // (1 ≤ i ≤ n) ∧ isMS(G_M, x, i) ∧ isMSX(S_M, x, i)
    i := i + 1
od;
// isMS(G_M, x, i − 1) ∧ isMSX(S_M, x, i − 1) ∧ i = n + 1 (implying isMS(G_M, x, n))
```

□

3. (40 points) Given a directed graph represented by an $n \times n$ adjacency matrix (named `Know[1..n, 1..n]`), the following program determines whether there exists an i (the sink or "celebrity" of the graph) such that all the entries in the i-th column (except for the ii-th entry) are 1, and all the entries in the i-th row (except for the ii-th entry) are 0.

```
i, j, next := 1, 2, 3;
while next <= n+1 do
  if Know[i,j] then i := next
  else j := next;
```

```
  next := next + 1;
od
if i = n+1 then candidate := j
else candidate := i;

wrong := false;
k := 1;
Know[candidate,candidate] := false;
while not wrong and k <= n do
  if Know[candidate,k] then wrong := true;
  if not Know[k,candidate] then
    if candidate <> k then wrong := true;
  k := k + 1;
od
if not wrong then celebrity := candidate
else celebrity := 0;
```

Annotate the program into a standard proof outline, showing clearly the partial correctness of the program.

Solution. Let $K(i,j)$ denote that `Know[i,j]`$=1$; hence, $\neg K(i,j)$ means `Know[i,j]`$=0$, as `Know[i,j]`$=0$ or 1.

Let *isSink*(v,m) assert that node v is a sink of the graph considering only vertices 1 through m. Formally, $\mathit{isSink}(v,m) \triangleq (1 \le v \le n) \wedge \forall i(1 \le i \le m \wedge i \ne v \to K(i,v)) \wedge \forall j(1 \le j \le m \wedge j \ne v \to \neg K(v,j))$. The special case $\mathit{isSink}(v,n)$ asserts that v is the sink of the entire graph.

// assume $n \geq 1$, which is preserved by the code and will be omitted later
i, j, next := 1, 2, 3;
// **inv:** $(1 \le \min(i,j) < \max(i,j) = \mathit{next}-1 \le n+1) \wedge \exists v(\mathit{isSink}(v,\mathit{next}-2) \to (v = i \vee v = j))$
while next <= n+1 **do**
  // $(1 \le \min(i,j) < \max(i,j) = \mathit{next}-1 \le n) \wedge \exists v(\mathit{isSink}(v,\mathit{next}-2) \to (v = i \vee v = j))$
  if Know[i,j] **then**
    // $K(i,j) \wedge (1 \le \min(i,j) < \max(i,j) = \mathit{next}-1 \le n) \wedge \exists v(\mathit{isSink}(v,\mathit{next}-2) \to v = j)$
    i := next
  else
    // $\neg K(i,j) \wedge (1 \le \min(i,j) < \max(i,j) = \mathit{next}-1 \le n) \wedge \exists v(\mathit{isSink}(v,\mathit{next}-2) \to v = i)$
    j := next
  fi;
  // $(1 \le \min(i,j) < \max(i,j) = \mathit{next} \le n-1) \wedge \exists v(\mathit{isSink}(v,\mathit{next}-1) \to (v = i \vee v = j))$
  next := next + 1
od;
// $(1 \le \min(i,j) < \max(i,j) = \mathit{next}-1 = n+1) \wedge \exists v(\mathit{isSink}(v,n) \to (v = i \vee v = j))$
if i = n+1 **then**
  // $\exists v(\mathit{isSink}(v,n) \to v = j)$
  candidate := j

```
else
   // ∃v(isSink(v,n) → v = i)
   candidate := i
fi;
// ∃v(isSink(v,n) → v = cand.)

wrong, k, Know[candidate,candidate] := false, 1, false;
// inv: (1 ≤ k ≤ n+1)∧¬K(cand., cand.)∧((¬wrong∧isSink(cand., k−1))∨(wrong∧¬isSink(cand., k − 1))) ∧ ∃v(isSink(v,n) → v = cand.)
while not wrong and k <= n do
   // (1 ≤ k ≤ n)∧¬K(cand., cand.)∧¬wrong∧isSink(cand., k−1)∧∃v(isSink(v,n) → v = cand.)
   if Know[candidate,k] then
      // (1 ≤ k ≤ n) ∧ ¬K(cand., cand.) ∧ ¬isSink(cand., k) ∧ ∃v(isSink(v,n) → v = cand.)
      wrong := true
   fi;
   // (1 ≤ k ≤ n) ∧ ¬K(cand., cand.) ∧ ((¬wrong ∧ isSink(cand., k − 1)) ∨ (wrong ∧ ¬isSink(cand., k))) ∧ ∃v(isSink(v,n) → v = cand.)
   if not Know[k,candidate] then
      // (1 ≤ k ≤ n)∧¬K(cand., cand.)∧¬K(k, cand.)∧((¬wrong∧isSink(cand., k−1))∨(wrong∧¬isSink(cand., k)))∧∃v(isSink(v,n) → v = cand.)
      if candidate <> k then
         // (1 ≤ k ≤ n)∧¬K(cand., cand.)∧¬isSink(cand., k)∧∃v(isSink(v,n) → v = cand.)
         wrong := true
   fi;
   // (1 ≤ k ≤ n) ∧ ¬K(cand., cand.) ∧ ((¬wrong ∧ isSink(cand., k)) ∨ (wrong ∧ ¬isSink(cand., k))) ∧ ∃v(isSink(v,n) → v = cand.)
  k := k + 1;
od;
// ((¬wrong∧isSink(cand., n))∨(wrong∧¬isSink(cand., n)))∧∃v(isSink(v,n) → v = cand.)
if not wrong then
   // ¬wrong ∧ isSink(cand., n) ∧ ∃v(isSink(v,n) → v = cand.)
   celebrity := candidate
else
   // wrong ∧ ¬isSink(cand., n) ∧ ∃v(isSink(v,n) → v = cand.)
   celebrity := 0
fi;
// ((celebrity ≠ 0) ∧ isSink(celebrity,n)) ∨ ((celebrity = 0) ∧ ¬∃v(isSink(v,n)))
```

□