Suggested Solutions for Homework Assignment #4

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

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.

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

$\square$

   (b) (10 points)

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

   Solution.

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

α :

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

β :

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

$\square$

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

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

Solution.

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

α :

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

β :

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

γ :

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

□

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
        fi
   fi
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 surrounded by a pair of pre- and post-conditions.

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;
// (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
        // (1 ≤ i ≤ n) ∧ isMS(G_M, x, i) ∧ isMSX(S_M, x, i)
    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]
              // (1 ≤ i ≤ n) ∧ isMS(G_M, x, i) ∧ isMSX(S_M, x, i)
          else
              // (1 ≤ i ≤ n) ∧ isMS(G_M, x, i) ∧ isMSX(0, x, i)
              Suffix_Max := 0;
              // (1 ≤ i ≤ n) ∧ isMS(G_M, x, i) ∧ isMSX(S_M, x, i)
          fi
          // (1 ≤ i ≤ n) ∧ isMS(G_M, x, i) ∧ isMSX(S_M, x, i)
    fi
    // (1 ≤ i ≤ n) ∧ isMS(G_M, x, i) ∧ isMSX(S_M, x, i)
    i := i + 1
    // (1 ≤ i ≤ n + 1) ∧ isMS(G_M, x, i − 1) ∧ isMSX(S_M, x, 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) A majority of an array of n elements is an element that has more than $\frac{n}{2}$ occurrences in the array. Below is a program that finds the majority of an array X of n elements or determines its non-existence. (Hint: if $A[i] \neq A[j]$, then the majority of A remains a majority in a new array B obtained from A by removing $A[i]$ and $A[j]$. Check out Udi Manber's algorithms book if you cannot understand the program.)

```
C,M := X[1],1;
i := 2;
while i<=n do
   if M=0 then C,M := X[i],1
          else if C=X[i] then M := M+1
                         else M := M-1
               fi
```

```
   fi;
   i := i+1
od;
if M=0 then Majority := -1
       else Count := 0;
            i := 1;
            while i<=n do
               if X[i]=C then Count := Count+1 fi;
               i := i+1
            od;
            if Count>n/2 then Majority := C
                         else Majority := -1
            fi
fi
```

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

Solution. As stated in the hint, the correctness of the code relies on the idea that, if two different elements are removed from an array A, the majority in A, if it exists, remains a majority in the remaining part B of array A. However, the majority in B may not be a majority in A, as an element might become the "majority" after two elements different from that element are removed. The repeated removals of two different elements are accomplished in the code by keeping a candidate (namely C, which may change over time) and counting its occurrences and, when a different element is encountered, the recorded number (namely M) of occurrences of the candidate is decremented to cancel out with the encountered element. The "remaining part" of X should be taken as the elements not yet scanned, i.e., elements in $X[i..n]$, plus the occurrences of the candidate, recorded in C and M, that await to be cancelled out.

Let $cnt(a, A)$ denote the number of occurrences of element a in an array A. Element a is the majority of A if $cnt(a, A) > \frac{|A|}{2}$ or $2cnt(a, A) > |A|$, where $|A|$ represents the number of elements in A. Let $isMaj(a, A)$ represent $2cnt(a, A) > |A|$, asserting that a is the majority of A, and $hasMaj(A)$ represent $\exists a(isMaj(a, A))$, asserting that A has a majority.

"If X has a majority, then the remaining part has a majority" is a loop invariant of the first while loop which carries out the removals of pairs of different elements while keeping a candidate. This can be stated as "$hasMaj(X) \to \exists a((C = a \wedge 2(cnt(a, X[i..n]) + M) > (M + n - i + 1)) \vee (C \neq a \wedge 2cnt(a, X[i..n]) > (M + n - i + 1)))$", where $(M + n - i + 1)$ equals the number of elements in the remaining part. Let us abbreviate this invariant as $majPreserved(X, i, C, M)$. The invariant is in the form of an implication, the contrapositive of which says that, if the remaining part of X does not have a majority, then X does not have a majority.

```
// assume n ≥ 1, which is preserved by the code and will be omitted later
C,M := X[1],1;
// C = X[1] ∧ M = 1
i := 2;
// (2 ≤ i ≤ n+1) ∧ M ≥ 0 ∧ majPreserved(X,i,C,M)
while i<=n do
   // (2 ≤ i ≤ n) ∧ M ≥ 0 ∧ majPreserved(X,i,C,M)
```

```
    if M=0 then
      // (2 ≤ i ≤ n) ∧ M = 0 ∧ majPreserved(X, i, C, M)
      C,M := X[i],1
      // (2 ≤ i ≤ n) ∧ M > 0 ∧ majPreserved(X, i + 1, C, M)
    else
      // (2 ≤ i ≤ n) ∧ M > 0 ∧ majPreserved(X, i, C, M)
      if C=X[i] then
        // (2 ≤ i ≤ n) ∧ M > 0 ∧ majPreserved(X, i, C, M) ∧ C = X[i]
        M := M+1
        // (2 ≤ i ≤ n) ∧ M > 0 ∧ majPreserved(X, i + 1, C, M) ∧ C = X[i]
      else
        // (2 ≤ i ≤ n) ∧ M > 0 ∧ majPreserved(X, i, C, M) ∧ C ≠ X[i]
        M := M-1
        // (2 ≤ i ≤ n) ∧ M ≥ 0 ∧ majPreserved(X, i + 1, C, M) ∧ C ≠ X[i]
      fi
      // (2 ≤ i ≤ n) ∧ M ≥ 0 ∧ majPreserved(X, i + 1, C, M)
    fi;
    // (2 ≤ i ≤ n) ∧ M ≥ 0 ∧ majPreserved(X, i + 1, C, M)
    i := i+1
    // (2 ≤ i ≤ n + 1) ∧ M ≥ 0 ∧ majPreserved(X, i, C, M)
od;
// M ≥ 0 ∧ majPreserved(X, n + 1, C, M)
if M=0 then
    // ¬hasMaj(X)
    Majority := −1
    // Majority = −1 ∧ ¬hasMaj(X)
else
    // hasMaj(X) → isMaj(C, X)
    Count := 0;
    // hasMaj(X) → isMaj(C, X) ∧ Count = 0
    i := 1;
    // hasMaj(X) → isMaj(C, X) ∧ Count = cnt(C, X[1..i − 1]) ∧ (1 ≤ i ≤ n + 1)
    while i<=n do
      // hasMaj(X) → isMaj(C, X) ∧ Count = cnt(C, X[1..i − 1]) ∧ (1 ≤ i ≤ n)
      if X[i]=C then
        // hasMaj(X) → isMaj(C, X) ∧ Count = cnt(C, X[1..i − 1]) ∧ (1 ≤ i ≤ n) ∧ X[i] = C
        Count := Count+1
        // hasMaj(X) → isMaj(C, X) ∧ Count = cnt(C, X[1..i]) ∧ (1 ≤ i ≤ n)
      fi;
      // hasMaj(X) → isMaj(C, X) ∧ Count = cnt(C, X[1..i]) ∧ (1 ≤ i ≤ n)
      i := i+1
      // hasMaj(X) → isMaj(C, X) ∧ Count = cnt(C, X[1..i − 1]) ∧ (1 ≤ i ≤ n + 1)
    od;
    // hasMaj(X) → isMaj(C, X) ∧ Count = cnt(C, X[1..n])
    if Count>n/2 then
      // isMaj(C, X)
      Majority := C
      // Majority = C ∧ isMaj(C, X)
```

```
    else
        // ¬hasMaj(X)
        Majority := −1
        // Majority = −1 ∧ ¬hasMaj(X)
    fi
    // (Majority = C ∧ isMaj(C, X)) ∨ (Majority = −1 ∧ ¬hasMaj(X))
fi
// (Majority = C ∧ isMaj(C, X)) ∨ (Majority = −1 ∧ ¬hasMaj(X))
```

□