When asked to write a formal proof, or to show a calculation, you need to produce one that is readable, augmented with sufficient comments to explain and convincingly defend your steps. An incomprehensible solution may lose all points.

You are allowed to bring along the Appendix of LN1.

There are 5 tasks; No. 2 is worth 4pt; others are 1.5 pt each.

1. [1.5 pt] Consider again this graph representation you had from your 2nd assignment:

\[
\text{Graph} = \text{record}\{ n: \text{int}, c:\text{Arrow}\[\] \}\ \\
\text{Arrow} = \text{Set of int}
\]

So, a \texttt{g:Graph} represents a graph whose nodes are integers \(0, 1, \ldots, \text{g.n}-1\), and for every node \(i\), \(\text{g.c}[i]\) gives us the set of all successors of \(i\) in the graph.

The following concepts are defined to help you writing specifications later:

\[
\text{nodes}(\text{g}) = \{i \mid 0 \leq i < \text{g.n}\}
\]

\[
\text{pred}(i) = \{h \mid h \in \text{nodes}(\text{g}) \land i \in \text{g.c}[h]\}
\]

The first gives you the set of all nodes in a graph \(g\), the secodn gives you the set of all predecessor of a node \(i\).

Now the tasks:

(a) Let us introduce the notation below:

\[
i \mapsto_g j
\]

to mean that \(j\) is reachable from \(i\). That is, there is a path in the graph \(g\), that goes from \(i\) to \(j\).

Furthermore, \(i\) is considered to be automatically reachable from itself.

Give a formal definition for \(\mapsto\). You are allowed to use a recursive definition.

SOLUTION:

Keeping the index \(g\) implicit.

\[
i \mapsto j = (\exists k : k \geq 0 : i \mapsto^k j) \\
i \mapsto^0 j = (j = i) \\
i \mapsto^{k+1} j = (\exists i' : i' \in \text{g.c}[i] : i' \mapsto^k j)
\]
(b) Give a formal specification of a program `isTree(g : Graph, r : int) : bool`, that checks whether `g` is actually a tree rooted in `r`. (Only a specification is asked!)

**SOLUTION**

\[
\{ r \in nodes(g) \} \ isTree(g, r) \quad \{ \ return = \ pred(r) = \emptyset \\
\quad \land (\forall i : i \in nodes(g) : r \rightarrow i) \\
\quad \land (\forall i : i \in nodes(g) : \#pred(i) \leq 1) \}
\]

2. [4 pt] Consider an array `a` of integers. A local peak in this array is an element `a[i]` such that:

\[
a[i-1] < a[i] \land a[i] > a[i+1]
\]

Here is a predicate to express that an array contains such a peak, over the domain `[0..k]`:

\[
\text{hasPeak a k} = (\exists i : 0 < i < k-1 : a[i-1] < a[i] \land a[i] > a[i+1])
\]

Here is a simple imperative program that calculates `hasPeak`:

\[
\{ * \ n \geq 2 * \} \quad // \ pre-condition
\]

\[
i := 2 ; \\
rise := a[0] < a[1] ; \\
peak := \text{false}
\]

\[
\text{while } i < n \text{ do } \\
\quad \text{peak} := \text{peak} \lor (\text{rise} \land a[i-1] > a[i]) ; \\
\quad \text{rise} := a[i-1] < a[i] ; \\
\quad i := i+1 \\
\}
\]

\[
\{ * \ \text{peak} = \ \text{hasPeak a n} * \} \quad // \ post-condition
\]

(a) Give a formal proof for the following split property for `hasPeak`:

\[
\text{hasPeak a (k+1)} = \text{hasPeak a k} \lor \left(\begin{array}{c}
a[k-2] < a[k-1] \\
\land \\
a[k-1] > a[k]
\end{array}\right)
\]

for \( k \geq 2 \).

**SOLUTION**

\[
\text{hasPeak a (k+1)} = \\
(\exists i : 0 < i < k : a[i-1] < a[i] \land a[i] > a[i+1])
\]

\[
= \quad // \ k \geq 2
\]

\[
(\exists i : 0 < i < k-1) \lor (i = k-1) : a[i-1] < a[i] \land a[i] > a[i+1]
\]

\[
= \quad // \ k \geq 2
\]

\[
(\exists i : 0 < i < k-1) \lor (i = k-1) : a[i-1] < a[i] \land a[i] > a[i+1]
\]

\[
\lor (\exists i : i = k-1 : a[i-1] < a[i] \land a[i] > a[i+1])
\]

\[
= \quad \text{hasPeak a k} \lor (a[k-1-1] < a[k-1] \land a[k-1] > a[k-1+1])
\]

2
(b) Now, give a formal proof that the above imperative program is correct. Crucially, you would need to come up with a good invariant. You don’t have to prove its termination.

SOLUTION:
Inv: $2 \leq i \leq n \land (\text{rise} = a[i-2] < a[i-1]) \land (\text{peak} = \text{hasPeak} a i)$
Exit condition trivial, because we terminate with $i = n$.
Initialization is ok. Note that $\text{hasPeak} a 2 = \text{false}$, because the qualification domain is empty.
For PIC, we get this wp:

$$
2 \leq i+1 \leq n \\
\land (a[i-1] < a[i] = a[i+1-2] < a[i+1-1]) \\
\land (\text{peak} \lor (\text{rise} \land (a[i-1] > a[i]))) = \text{hasPeak} a (i+1))
$$

2st and 2nd conjunts are trivial. 3rd follows from lemma proven before.

3. [1.5 pt] Consider the following uPL specification of the program $P$:

$$\{ x>0 \} \ Y := y; P(x:int,OUT y:int) \{ \text{return} \geq 0 \land (y-x^2 = \text{return} \ast Y) \}$$

Consider this call to $P$:

$$\{ k>0 \} \ r := P(k,k) \{ r+k > 0 \}$$

Calculate first the weakest pre-condition of the call with respect to the above post-condition. Use the Black Box Reduction Rule to do it; your calculation should show how this rule is applied. Limit the calculation to no more than 10 steps.
Then give an argument as to whether the specification of the call above is valid or not.

SOLUTION

$$\{ k>0 \} \ // \ should \ imply \ the \ calculated \ wp$$

// wp: k>0 \ /\ (\text{return} \geq 0 \ /\ (y' - k^2 = \text{return} \ast k) \implies \text{return} + y' > 0)

x,y := k,k // x,y assumed fresh!

// wp: x>0 \ /\ (\text{return} \geq 0 \ /\ (y' - x^2 = \text{return} \ast y) \implies \text{return} + y' > 0)

r := P(x,y); 

// wp: r+y > 0

k:=y

$$\{ r+k > 0 \}$$
4. [1.5 pt] Consider the following class:

```java
class C {
    C x;
    int y;

    // In the spec below, A and Z are logical variables of type int
    // respectively C

    // A specification for method m,
    // pre: (this.x = A) \&\& (u=Z)
    // post: this.x - Z.x = A
    m(C u) { ... }
}
```

(a) Calculate the weakest pre-conditions of the following cOore statement with respect to the given post-conditions; the statement is assumed to be inside some method in C.

```java
{ ? }
this.x := new C(); v = this.x
{
(b ? u1 : u2) = v) \lor (\forall w : C \& w \neq null \rightarrow (w.y=v.y))
}
```

You have to show the calculation, but limit it to no more than about 10 steps. Is the resulting wp satisfiable?

**SOLUTION**

```
((b ? u1 : u2) = v) \lor (\forall w : C \& w \neq null \rightarrow w.y=v.y)[this.x/v][newC/this.x]
```

```
= ((b ? u1 : u2) = this.x) \lor (\forall w : C \& w \neq null \rightarrow w.y=this.x.y)[newC/this.x]
```

```
= // shifting equalities inside cond:
(b ? u1 = this.x : u2 = this.x) \lor (\forall w : C \& w \neq null \rightarrow w.y=this.x.y)[newC/this.x]
```

```
= // left term first
(b ? false : false) \lor (\forall w : C \& w \neq null \rightarrow w.y=this.x.y)[newC/this.x]
```

```
= // we can drop left term; it is false anyway:
(\forall w : C \& w \neq null \rightarrow w.y=this.x.y)[newC/this.x]
```

```
= (\forall w : C \& (w \neq null \rightarrow w.y=this.x.y)[newC/this.x])
\& (this.x \neq null \rightarrow this.x.y=this.x.y)[newC/this.x]
```

```
= // bottom term first
(\forall w : C \& (w \neq null \rightarrow w.y=this.x.y)[newC/this.x])
\& (true \rightarrow 0=0)
```

```
= // dropping the bottom term, it is true anyway
(\forall w : C \& (w \neq null \rightarrow w.y=this.x.y)[newC/this.x])
```

```
= (\forall w : C \& w \neq null \rightarrow w.y=0)
```

4
(b) Consider this call to the above method m: the call happens in another method of the same class C:

\[
\{k.x = 0\} \ k.m(this) \ {k.x = this.x}\]

Give the verification condition, according to cOore’s Rule of Adaptation, that corresponds to the validity of the above specification of the call. Is the verification condition valid? (Just give an answer; no proof is required).

SOLUTION:

\[
\text{Heap} \land \text{locs} \\
\land (k.x\diamond = 0) \\
\land (\forall A, Z, this': H \bullet (k.x\diamond = A) \land (this = Z) \land (k = this') \Rightarrow (this'.x - Z.x = A)) \\
\Rightarrow \\
(k.x = this.x)
\]

(More on the next page!)
5. [1.5 pt] Consider this predicate expressing that an element \(a[i]\) in an array \(a\) is greater than the sum of all its predecessor:

\[
\text{sooBig } a[i] = \text{SUM}(a[0..i]) < a[i]
\]

Below is an imperative program for checking if such an element exists in \(a\). It has been annotated with the pre- and post-conditions, as well as the invariant.

```plaintext
// Pre-condition P: \{ s=0 /\ i=0 /\ k=0 /\ n>=0 \}
// invariant I:
// 0<=i<=n
// \ /
// s = \text{SUM}(a[0..i])
// \ /
// k = \text{COUNT}[ j | j from [0..i) /\ \text{sooBig } a[j] ]

while i<n do {
    // check if \(a[i]\) is sooBig; if so, add it to counter \(k\):
    if s<a[i] then k:=k+1 else skip ;
    // keep track of sum:
    s := s+a[i] ;
    i:=i+1
}

// Post-condition Q:
{ k>0 = (exists j: 0<=j<n: \text{sooBig } a[j] ) }
```

Someone has already proven all these results below, where `body` is the body of the loop above, and P, Q, I are as annotated above:

- \(P \Rightarrow I\)
- \{I\} while i<n do body \{Q\}
- \{I \land i<n\} body \{I\}

Now we want to optimize the program by adding a break (in the form of an additional conjunct, added to your loop’s guard):

```plaintext
\{ P \} while i<n /\ (k>=0) do \{ ... // same body as above \} \{ Q \}
```

Prove that the new program satisfies the same specification. **Hint:** you don’t want to prove this from the scratch. You may need to use this property of `COUNT`:

\[
\text{COUNT } s > 0 \Rightarrow (\exists x :: x \in s)
\]

**SOLUTION**

Using the Loop Strengthening Rule it is sufficient to prove \(I \land i < n \land (k > 0) \Rightarrow Q\).

**PROOF:**

A1: \(0 \leq i \leq n\)

A2: \(s = \ldots\)

A3: \(k = \text{COUNT}[ j | j from [0..i) \land \text{sooBig } a[j] ]\)

A4: \(i < n\)

A5: \(k > 0\)
G: \( k > 0 = (\exists j : 0 \leq j < n : \text{sooBig a j}) \)

BEGIN

1. // A3,A5, prop. COUNT
\( (\exists x :: x \in [j | j \text{ from } [0..i) \land \text{sooBig a j}] ) \)

2. // \( \exists \)-elim
\[ \text{SOME } x \mid x \in [j | j \text{ from } [0..i) \land \text{sooBig a j}] \]

3. // element of comprehension
\( (\exists j : j \in [0..i) \land \text{sooBig a j : } x = j) \)

4. // \( \exists \)-elim
\( j \in [0..i) \land \text{sooBig a j} \land (x = j) \)

5. // follows from 1st conjunct above, and A4
\( 0 \leq j < n \)

6. // \( \exists \)-intro on 5 and 2nd conjunct of 4
\( (\exists j : 0 \leq j < n : \text{sooBig a j}) \)

7. // A4 and A5
\( k > 0 = (\exists j : 0 \leq j < n : \text{sooBig a j}) \)

DONE