Weakest Precondition Calculus

You should hand in attempts to the questions indicated by (*) to your tutor at the start of each tutorial. Showing effort at answering the indicated questions will contribute to the 4% “Tutorial Preparation” component of the course; your attempts will not be marked for correctness. You may collaborate with your fellow students or others, so long as you hand in your work individually and clearly indicate who you have worked with.

Question 1

Let $A$ be the program fragment:

```plaintext
if (x>y) then
  tmp:=x;
  x:=y;
  y:=tmp
```

Note that there is no else case to this if statement. The rule for if without else was derived in lectures and can be found in the appendix.

Express the following in as simple a form as you can:

\[ wp(A, x < y) \]  

\[ \text{(*)} \]

Solution

\[
wp(A', x < y) \equiv wp(tmp := x; x := y, wp(y := tmp, x < y))
\]
\[
\equiv wp(tmp := x; x := y, x < tmp)
\]
\[
\equiv wp(tmp := x, wp(x := y, x < tmp))
\]
\[
\equiv wp(tmp := x, y < tmp)
\]
\[
\equiv y < x
\]

We then use this result to find the weakest precondition for the whole program:

\[
w(A, x < y) \equiv (x > y \land wp(A', x < y)) \lor (\neg(x > y) \land x < y)
\]
\[
\equiv (x > y \land y < x) \lor (x \leq y \land x < y)
\]
\[
\equiv x > y \lor x < y
\]
\[
\equiv x \neq y
\]
Question 2

Let $B$ be the program fragment:

\[
\begin{align*}
\text{if (} k < 0 \text{) then} \\
&\quad k := k + 1; \\
&\quad n := n \times 2
\end{align*}
\]

Express the following in as simple a form as you can:

- a) $wp(B, (k = 0 \land n = 0))$
- b) $wp(B, n = 2^k)$

Solution

- a) $wp(B', (k = 0 \land n = 0)) \equiv wp(k := k + 1, k = 0 \land 2n = 0) \equiv (k = -1 \land n = 0)$
  $wp(B, (k = 0 \land n = 0)) \equiv (k < 0 \land wp(B', (k = 0 \land n = 0))) \lor (k < 0 \land (k = 0 \land n = 0))$ $\equiv (k = -1 \lor k = 0) \land n = 0$

- b) $wp(B', (n = 2^k)) \equiv wp(k := k + 1, (2n = 2^k)) \equiv n = 2^k$
  $wp(B, (n = 2^k)) \equiv (k < 0 \land wp(B', (n = 2^k))) \lor (k < 0 \land (n = 2^k))$ $\equiv (k < 0 \land n = 2^k) \lor (k \geq 0 \land n = 2^k)$ $\equiv (n = 2^k)$

Question 3

The following program $Facto$ is intended to compute the factorial of any natural number.

\[
\begin{align*}
i &:= 0; \\
fac &:= 1; \\
\text{while } (i \neq n) \text{ do} \\
&\quad i := i + 1; \\
&\quad fac := fac \times i
\end{align*}
\]

We will use the abbreviations $Init$ for the initialization code and $Body$ for the body of the while statement $Loop$.

The postcondition we are interested in will be $fac = n!$

We define $P_k$ to be the predicate that must be true before $Loop$ executes to guarantee that the loop terminates after exactly $k$ iterations in a state that satisfies this postcondition.
1. Give expressions for \( P_0, P_1, P_2 \) in as simple a form as you can. \((*)\)

2. Infer an expression for \( P_k \). (Don’t worry about the inductive proof.) \((*)\)

3. Give an expression for \( wp(\text{Loop}, \text{fac} = n!) \) in as simple a form as you can. \((*)\)

4. Hence find \( wp(\text{Facto}, \text{fac} = n!) \). You will need to use the fact that \( 0! = 1 \). \((*)\)

**Solution**

1) 
\[
P_0 \equiv i = n \land \text{fac} = n!
\]
\[
\equiv i = n \land \text{fac} = i!
\]

Strictly speaking we do not have to write the second line above out, but it makes it a bit easier to calculate \( P_1 \) below.

\[
P_1 \equiv i \neq n \land wp(\text{Body}, P_0)
\]
\[
\equiv i \neq n \land wp(i := i + 1, wp(\text{fac} := \text{fac} \times i, P_0))
\]
\[
\equiv i \neq n \land wp(i := i + 1, i = n \land \text{fac} \times i = i!)
\]
\[
\equiv i \neq n \land i + 1 = n \land \text{fac} \times (i + 1) = (i + 1)!
\]
\[
\equiv i + 1 = n \land \text{fac} = i!
\]

The last step looks a bit dangerous - are we allowed to divide by \( i + 1 \) if \( i + 1 \) might be zero? However we already have the term \( i! \), which would not be well defined if \( i \) were negative, so \( i + 1 \) is not zero. If this implicit argument disturbs you then you can redo the whole proof with the ‘safe’ postcondition \( \text{fac} = n! \land n \geq 0 \).

\( P_2 \) follows similarly and we do not give all details:

\[
P_2 \equiv i + 2 = n \land \text{fac} = i!
\]

2) This looks likely:

\[
P_k \equiv i + k = n \land \text{fac} = i!
\]

To be sure that we have this correct we would need to do an inductive argument as follows:

Inductive hypothesis: 
\[
P_k \equiv i + k = n \land \text{fac} = i!.
\]

Weakest Preconditions 3
Now consider $P_{k+1}$: we want it to be $i + (k + 1) = n \land fac = i!$

\[ P_{k+1} \equiv (i \neq n) \land wp(Body, P_k) \]
\[ \equiv (i \neq n) \land wp(i := i + 1; fac := \text{fac } \ast i, P_k) \]
\[ \equiv (i \neq n) \land wp(i := i + 1, wp(\text{fac := fac } \ast i, P_k)) \]
\[ \equiv (i \neq n) \land wp(i := i + 1, wp(\text{fac := fac } \ast i, (i + k = n) \land fac = i!))) \]
\[ \equiv (i \neq n) \land wp(i := i + 1, (i + k = n) \land (fac \ast i) = i!)) \]
\[ \equiv (i \neq n) \land ((i + 1) + k = n) \land (fac \ast (i + 1)) = (i + 1)! \]
\[ \equiv (i \neq n) \land (i + (k + 1) = n) \land (fac = i!) \]
\[ \equiv (i + (k + 1) = n) \land (fac = i!) \]

3) \[ wp(\text{Loop, fac = n!}) \equiv \exists k. (k \geq 0 \land i + k = n \land fac = i!) \]
\[ \equiv n \geq i \land fac = i! \]

4) \[ wp(\text{Facto, fac = n!}) \equiv \]
\[ wp(Init, wp(\text{Loop, fac = n!})) \]
\[ \equiv wp(Init, n \geq i \land fac = i!) \]
\[ \equiv wp(i := 0, wp(\text{fac := 1, n } \geq i \land fac = i!)) \]
\[ \equiv wp(i := 0, n \geq i \land 1 = i!)) \]
\[ \equiv n \geq 0 \land 1 = 0! \]
\[ \equiv n \geq 0 \]

Weakest Preconditions
1 Appendix: Weakest Precondition Rules

\[
wp(x := e, Q(x)) \equiv Q(e)
\]
\[
wp(S_1; S_2, Q) \equiv wp(S_1, wp(S_2, Q))
\]
\[
wp(\text{if } b \text{ then } S_1 \text{ else } S_2, Q) \equiv (b \Rightarrow wp(S_1, Q)) \land (\neg b \Rightarrow wp(S_2, Q))
\]
\[
wp(\text{if } b \text{ then } S, Q) \equiv (b \Rightarrow wp(S, Q)) \land (\neg b \Rightarrow Q)
\]
\[
wp(\text{while } b \text{ do } S, Q) \equiv \exists k. (k \geq 0 \land P_k)
\]

\(P_k\) is the weakest predicate that must be true before while \(b\) do \(S\) executes, in order for the loop to terminate after exactly \(k\) iterations in a state that satisfies \(Q\).