1 Predicate Abstraction

i. Let us first visualise $c$ and $\neg c$ in a Venn diagram:

\[
\begin{array}{c}
\neg c \\
\bullet \quad c
\end{array}
\]

$\text{Pred}(\neg c)$ gives the weakest under-approximation of $\neg c$. Note that $\text{Pred}(\neg c)$ implies $\neg c$, but $\neg c$ does not (in general) imply $\text{Pred}(\neg c)$. A possible visualisation in a Venn diagram might then be:

\[
\begin{array}{c}
\text{Pred}(\neg c) \\
\bullet \quad c
\end{array}
\]

By negating $\text{Pred}(\neg c)$, we get the strongest over-approximation, visualised as follows:

\[
\begin{array}{c}
\text{Pred}(\neg c) \\
\bullet \quad \neg \text{Pred}(\neg c)
\end{array}
\]

*Some exercises were adapted from earlier ones written by Stephan van Staden and Carlo A. Furia.
ii. We build a Boolean abstraction from $C_1$, one line at a time. First, we over-approximate 
\begin{verbatim}
assume x > 0 end
\end{verbatim}
with \begin{verbatim}
assume \neg\text{Pred}(\neg x > 0) end
\end{verbatim}, followed by a parallel conditional assignment updating the predicates with respect to the original \begin{verbatim}
assume
\end{verbatim} statement.

\begin{verbatim}
\neg\text{Pred}(\neg x > 0) = \neg(\neg p) \\
= p
\end{verbatim}

Hence we add \begin{verbatim}
assume p end
\end{verbatim} to $A_1$. This should be followed by a parallel conditional assignment (as described in the slides):

\begin{verbatim}
if Pred(+\text{ex}(i)) then 
  p(i) := True 
elseif Pred(-\text{ex}(i)) then 
  p(i) := False 
else 
  p(i) := ?
end
\end{verbatim}

Using the axiom $\vdash \{c \Rightarrow post\} assume c end \{post\}$ for the weakest precondition of assume statements, which instantiates to $\vdash \{x > 0 \Rightarrow post\} assume x > 0 end \{post\}$, we compute every $+/−\text{ex}(i)$ for predicates $i$:

\begin{verbatim}
+\text{ex}(p) = (x > 0 \Rightarrow x > 0) 
−\text{ex}(p) = (x > 0 \Rightarrow \neg x > 0) 
+\text{ex}(q) = (x > 0 \Rightarrow y > 0) 
−\text{ex}(q) = (x > 0 \Rightarrow \neg y > 0) 
+\text{ex}(r) = (x > 0 \Rightarrow z > 0) 
−\text{ex}(r) = (x > 0 \Rightarrow \neg z > 0)
\end{verbatim}

We apply the simplification step from the slides, and consider only the branches that correspond to a $+/−\text{ex}(i)$ that is valid. It so happens that only $+\text{ex}(p)$ is valid, so we compute:

\begin{verbatim}
\text{Pred}(+\text{ex}(p)) = \text{Pred}(x > 0 \Rightarrow x > 0) = \neg p \vee p = true
\end{verbatim}

resulting in the parallel conditional assignment:

\begin{verbatim}
if True then 
  p := True 
else 
  p := ?
end
\end{verbatim}

This simplifies even further to $p := \text{True}$, which we add to $A_1$. 


Next, we address the assignment \( z := (x \cdot y) + 1 \). Recall that an assignment \( x := f \) is over-approximated by a parallel conditional assignment:

\[
\text{if } \text{Pred}(+f(i)) \text{ then}
\quad p(i) := \text{True}
\text{elseif } \text{Pred}(-f(i)) \text{ then}
\quad p(i) := \text{False}
\text{else}
\quad p(i) := ?
\text{end}
\]

Using the axiom \( \vdash \{ \text{post}/f \} x := f \{ \text{post} \} \), which instantiates to \( \vdash \{ \text{post}/((x \cdot y) + 1)/z \} z := (x \cdot y) + 1 \{ \text{post} \} \), and the definition of \( \pm f(i) \) for predicates \( i \), we get:

\[
\begin{align*}
\text{Pred}(+f(p)) &= \text{Pred}(x > 0) \\
&= p \\
\text{Pred}(-f(p)) &= \text{Pred}(\neg x > 0) \\
&= \neg p \\
\text{Pred}(+f(q)) &= \text{Pred}(y > 0) \\
&= q \\
\text{Pred}(-f(q)) &= \text{Pred}(\neg y > 0) \\
&= \neg q \\
\text{Pred}(+f(r)) &= \text{Pred}(((x \cdot y) + 1 > 0) \\
&= (p \land q) \lor (\neg p \land \neg q) \\
\text{Pred}(-f(r)) &= \text{Pred}((x \cdot y) + 1 > 0) \\
&= \text{Pred}(((x \cdot y) + 1 \leq 0) \\
&= \text{false}
\end{align*}
\]

The parallel conditional assignments for \( p, q \) have no effect, hence we add only the following to \( A_1 \):

\[
\text{if } (p \text{ and } q) \text{ or } (\neg p \text{ and } \neg q) \text{ then}
\quad r := \text{True}
\text{elseif False then}
\quad r := \text{False}
\text{else}
\quad r := ?
\text{end}
\]

Finally, we address the assertion \textbf{assert } z \geq 1 \textbf{ end}. The Boolean abstraction is simply \textbf{assert } \text{Pred}(z \geq 1) \textbf{ end}. We have:

\[
\text{Pred}(z \geq 1) = r
\]

and hence add \textbf{assert } r \textbf{ end} to \( A_1 \).
Altogether, $A_1$ is the following program:

```plaintext
assume p end
p := True

if (p and q) or (not p and not q) then
  r := True
elseif False then
  r := False
else
  r := ?
end

assert r end
```

With a further simplification, we get:

```plaintext
assume p end
p := True

if (p and q) or (not p and not q) then
  r := True
else
  r := ?
end

assert r end
```
iii. (a) After normalising the program (following the details in the slides) we get:

```
if ? then
    assume x > 0 end
    y := x + x
else
    assume x <= 0 end
    if ? then
        assume x = 0 end
        y := 1
    else
        assume x /= 0 end
        y := x * x
    end
end
assert y > 0 end
```

(b) To build $A_2$ from the normalised code above, apply the transformations to each assignment, assume, and assert, analogously to how I did when constructing $A_1$ (except that this time you only have two predicates, $p$ and $q$). The resulting abstraction (after some simplifications) should be equivalent to this:

```
if ? then
    assume p end
    p := True
    q := True
else
    assume not p end
    p := False
    if ? then
        assume not p end
        p := False
    else
        assume True end -- can delete this assume
        q := ?
    end
end
assert q end
```
2 Error Traces

i. An abstract error trace is:

\[
[p, \text{not } q, r] \\
\text{assume } p \text{ end} \\
[p, \text{not } q, r] \\
p := \text{True} \\
[p, \text{not } q, r] \\
r := ? \\
[p, \text{not } q, \text{not } r] \\
\text{assert } r \text{ end}
\]

Observe that each concrete instruction corresponds to a (compound) abstract instruction. We can check whether or not this is a feasible concrete run by computing the weakest precondition of the concrete instructions with respect to \(p \land \neg q \land \neg r\), interpreting conditions (assume, conditionals, or exit conditions) as asserts. Recall that the weakest preconditions of assert statements can be computed using \(\{c \land \text{post}\} \vdash \text{assert } c \text{ end } \{\text{post}\}\).

\[
\{x > 0 \land y \leq 0 \land (x*y)+1 \leq 0\} \\
\{x > 0 \land (x > 0 \land y \leq 0 \land (x*y)+1 \leq 0)\} \\
\text{assert } x > 0 \text{ end} \\
\{x > 0 \land y \leq 0 \land (x*y)+1 \leq 0\} \\
z := (x*y) + 1 \\
\{x > 0 \land y \leq 0 \land z \leq 0\} \\
[p, \text{not } q, \text{not } r]
\]

Executing the concrete program on a state \(s\) such that

\[
s \models x > 0 \land y \leq 0 \land (x*y)+1 \leq 0
\]

will reveal the fault. One possible input state (of many) is \(s = \{x \mapsto 3, y \mapsto -2, z \mapsto \_\} \).

ii. Here is an abstract counterexample trace:

\[
[\text{not } p, \text{not } q] \\
\text{assume not } p \text{ end} \\
[\text{not } p, \text{not } q] \\
p := \text{False} \\
[\text{not } p, \text{not } q] \\
\text{assume } \text{True} \text{ end} \\
[\text{not } p, \text{not } q] \\
q := ? \\
[\text{not } p, \text{not } q] \\
\text{assert } q \text{ end}
\]

As before, we check whether or not this abstract execution reflects a feasible, concrete counterexample, by computing the weakest precondition of the corresponding concrete instructions with respect to \(\neg p \land \neg q\). Again, we interpret conditions (assumes in this case) as asserts, and apply the corresponding Hoare logic axioms:
\{x < 0 \text{ and } x \cdot x \leq 0\}
\{x \leq 0 \text{ and } (x /= 0 \text{ and } x \cdot x \leq 0)\}\}
  \text{assert } x \leq 0
\{x /= 0 \text{ and } (x \leq 0 \text{ and } x \cdot x \leq 0)\}
  \text{assert } x /= 0 \text{ end}
\{x \leq 0 \text{ and } x \cdot x \leq 0\}
y := x \cdot x
\{x \leq 0 \text{ and } y \leq 0\}
[\text{not } p, \text{ not } q]

Observe that in this case, the weakest precondition we have constructed is equivalent to false. There is no assignment to \(x\) that will satisfy the assertion. Hence the abstract counterexample is infeasible (spurious) in the concrete program; abstraction refinement is needed.