Mini-test 1

Q1. Prove that:

1. $\exists x \forall y A(x, y) \to \forall y \exists x A(x, y)$

1. assume $\exists x \forall y A(x, y)$
2. $\forall y A(a, y)$ for some a by existential elimination
3. $A(a, b)$ for an arbitrary b by universal elimination
4. $\exists x A(x, b)$ by $\exists$ introduction
5. $\forall y \exists x A(x, y)$ by $\forall$ introduction
6. $\exists x \forall y A(x, y) \to \forall y \exists x A(x, y)$ by $\to$ introduction from (1) and (5)

2. $\forall x(A(x) \to B(x)) \wedge \forall x(B(x) \to C(x)) \to \forall x(A(x) \to C(x))$

1. assume $\forall x(A(x) \to B(x)) \wedge \forall x(B(x) \to C(x))$
2. $\forall x(A(x) \to B(x))$ by $\wedge$ elimination from (1)
3. $(A(x) \to B(x))$ by $\forall$ elimination from (2)
4. $\forall x(B(x) \to C(x))$ by $\wedge$ elimination from (1)
5. $(B(x) \to C(x))$ by $\forall$ elimination from (4)
6. assume $A(x)$
7. $B(x)$ by $\to$ elimination from (3)
8. $C(x)$ by $\to$ elimination from (5)
9. $(A(x) \to C(x))$ by $\to$ introduction from (6) and (8)
10. $\forall x(A(x) \to C(x))$ by $\forall$ introduction
11. $\forall x(A(x) \to B(x)) \wedge \forall x(B(x) \to C(x)) \to \forall x(A(x) \to C(x))$ by $\to$ introduction from (1) and (10)

Q2. Does the logical statement below hold? If so, give a proof. If not, give a counterexample for it.

$\forall x \exists y A(x, y) \to \exists y \forall x A(x, y)$

No, it doesn't. A counterexample can be $\{D = \{1, 2\}, A(1, 1), A(2, 2)\}$

Q3. Does the satisfaction hold?

1. $\{D = \{1, 2\}, p(1, 2), q(1)\} \models \forall x \forall y\ p(x, y) \to q(x)$ **[yes/no] YES**

2. $\{D = \{1, 2\}, p(1, 2), q(1)\} \models \forall x\ p(x, x) \to q(x)$ **[yes/no] YES**

3. $\{D = \{1, 2\}, p(1, 1), q(2)\} \models \forall x \forall y\ p(x, y) \to q(x)$ **[yes/no] NO**

Mini-test 2

Given the program text

```
         void main(int n) {
           int i, j;
L1:        i = 0; j = n;
L2:        while (i < n) { i++; j--; }
L3         assert(n == i+j);
L4:        assert(j == 0);
         }
```

Q1. Represent the program formally, i.e., as a tuple $P = (V, \varphi_{init}, \varphi_{error}, R)$.

$V = (pc, n, i, j)$
$\varphi_{init}(v) = (pc = L_1)$
$\varphi_{err}(v) = (n \neq i + j \vee j \neq 0)$
$R = (pc = L_1 \wedge pc' = L_2 \wedge i' = 0 \wedge j' = n)\vee$
$(pc = L_2 \wedge pc' = L_2 \wedge i < n \wedge i' = i + 1 \wedge j' = j + 1)\vee$
$(pc = L_2 \wedge pc' = L_3 \wedge i \geq n)\vee$
$(pc = L_3 \wedge pc' = L_4 \wedge n = i + j)\vee$
$(pc = L_3 \wedge pc' = L_{err} \wedge n \neq I + j)\vee$
$(pc = L_4 \wedge pc' = L_{safe} \wedge j = 0)\vee$
$(pc = L_4 \wedge pc' = L_{err} \wedge j \neq 0)$

Q2. Inductive invariant computation:

Q2.1. Give an inductive invariant that proves the first assertion. Give conditions that this inductive invariant has to satisfy.

$$\varphi = (pc = L_1 \vee n = i + j)$$

The conditions that this inductive invariant has to satisfy are:
$\varphi_{init} \models \varphi$
$post(\varphi, \rho R) \models \varphi$

Q2.2. Give an inductive invariant that proves the second assertion.

$$(pc = L_1 \vee (pc = L_2 \wedge n = i + j) \vee j = 0)$$

Q3. Provide a well founded set $(W, \prec)$, and a ranking function r that prove the program termination.

One ranking function whose value decreases during each loop iteration can be $r(i, n) = n - i$ with ranking bound $r(i, n) \geq 0$ such that the corresponding well-founded set is $(\mathbb{N}, >)$.

Mini-test 3

Q1. Define $post(\varphi(v), R(v, v')) =$

$\exists v'' : \varphi[v''/v] \wedge \rho[v''/v][v/v']$

Q2. Compute $post(x \leq y \wedge z = 1, x' = x + 1 \wedge y' = y - 1 \wedge z' \geq z)$

$$\begin{aligned}
&= \exists v''\ (x'' \leq y'' \wedge z'' = 1 \wedge x = x'' + 1 \wedge y = y'' - 1 \wedge z \geq z'') \\
&= \exists v''\ (x'' \leq y'' \wedge z'' = 1 \wedge x'' = x - 1 \wedge y'' = y + 1 \wedge z \geq z'') \\
&= (x - 1 \leq y + 1 \wedge z \geq 1)
\end{aligned}$$

Q3. Compute $post(a = b, a' = a + 1 \wedge b' = b - 2)$

$$\begin{aligned}
&= \exists v''\ (a'' = b'', a = a'' + 1 \wedge b = b'' - 2) \\
&= \exists v''\ (a'' = b'', a'' = a - 1 \wedge b'' = b + 2) \\
&= (a - 1 = b + 2)
\end{aligned}$$

Q4. Prove $\forall\varphi\forall\psi\forall R : post(\varphi \vee \psi, R) \models\!\!\!\!\!\dashv post(\varphi, R) \vee post(\psi, R)$

1. assume $post(\varphi \vee \psi, R)$
2. $\exists v" : (\varphi(v) \vee \psi(v))[v"/v] \wedge R[v"/v][v/v']$ by reducing *post* into its definition
3. $\exists v" : (\varphi(v") \wedge (R(v", v)) \vee \exists v" : (\psi(v") \wedge R(v", v))$ by distributing the conjunction and the existential quantifier over the disjunction
4. $post(\varphi, R) \vee post(\psi, R)$ by rewriting back in terms of *post*
5. $post(\varphi, R) \vee post(\psi, R)$ by rewriting back in terms of *post*
6. therefore, $post(\varphi \vee \psi, R) \models post(\varphi, R) \vee post(\psi, R)$
7. $\forall\varphi\forall\psi\forall R : post(\varphi \vee \psi, R) \models post(\varphi, R) \vee post(\psi, R)$ by applying $\forall$ introduction

1. assume $(post(\varphi, R) \vee post(\psi, R))$
2. $(\exists v" : \varphi(v)[v"/v] \wedge R(v, v')[v"/v][v/v']) \vee (\exists v" : \psi(v)[v"/v] \wedge R(v, v'))[v"/v][v/v'])$ by reducing *post* into its definition
3. $\exists v" : (\varphi(v") \vee \psi(v")) \wedge R(v", v))$ by collecting terms over the existential quantifier and R
4. $post(\varphi \vee \psi, R)$ by rewriting back in terms of *post*
5. therefore, $(post(\varphi, R) \vee post(\psi, R)) \models post(\varphi \vee \psi, R)$
6. $\forall\varphi\forall\psi\forall R : (post(\varphi, R) \vee post(\psi, R)) \models post(\varphi \vee \psi, R)$ by applying $\forall$ introduction

Mini-test 4

Q1. Prove that α is monotonic, i.e., $\forall\phi\forall\psi : (\phi \models \psi) \rightarrow (\alpha(\phi) \models \alpha(\psi))$.

Refer to *Homework 4, Exercise 2, bullet point 2.*

Q2. Given the set of predicates $P = \{x \geq 5, x \leq 10, x = 6\}$, compute

1. $post(x = 6, x' = x + 1) = (x{=}7)$.
2. $\alpha(x \geq 6) = (x \geq 5)$.
3. $post^{\#}(x = 6, x' = x + 1) = (x \geq 5 \wedge x \leq 10)$.

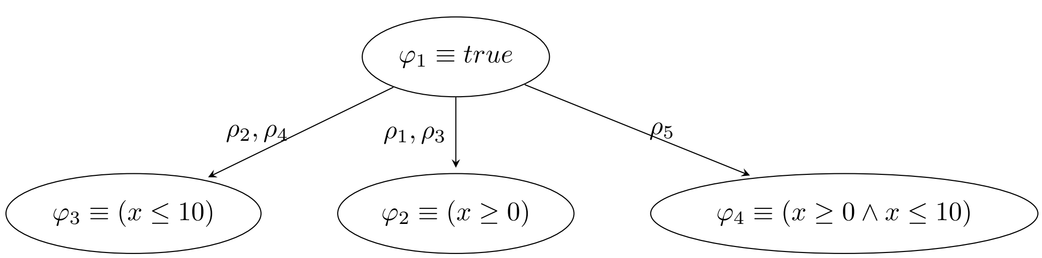
Q3. Given a program text.

```
int x;
A: if (x>0) {
B:   while(x > 0) x--;
   } else {
C:   x = 10;
   }
D:
```

Given the predicates $\{x \geq 0, x \leq 10\}$, compute the corresponding abstract reachability tree.

Let
$\varphi_{init}(v) = (pc = A)$
$\rho_1(v, v') = (pc = A \wedge pc' = B \wedge x > 0 \wedge x' = x)$
$\rho_2(v, v') = (pc = A \wedge pc' = C \wedge x \leq 0 \wedge x' = x)$
$\rho_3(v, v') = (pc = B \wedge pc' = B \wedge x > 0 \wedge x' = x + 1)$
$\rho_4(v, v') = (pc = B \wedge pc' = D \wedge x \leq 0 \wedge x' = x)$
$\rho_5(v, v') = (pc = C \wedge pc' = D \wedge x' = 10)$

We start from the abstract initial state and continue applying each of the 5 transition relations on each state until no new state is reached.
$\alpha(\varphi_{init}(v)) = \alpha(pc = A) = true \equiv \varphi_1$
$post^{\#}(\varphi_1, \rho_1) = (x \geq 0) \equiv \varphi_2$
$post^{\#}(\varphi_1, \rho_2) = (x \leq 10) \equiv \varphi_3$
$post^{\#}(\varphi_1, \rho_3) = (x \geq 0) \equiv \varphi_2$ (already reached!)
$post^{\#}(\varphi_1, \rho_4) = (x \leq 10) \equiv \varphi_3$ (already reached!)
$post^{\#}(\varphi_1, \rho_5) = (x \geq 0 \wedge x \leq 10) \equiv \varphi_4$
We continue to do so for the remaining states φ_2, φ_3, and φ_4. But since there is no new state reached the abstract reachability computation stops here. The resulting tree is given below.



Mini-test 5

Q1. Given the program Prog = (V, φ_{init}, φ_{error}, $\{\rho_1, \rho_2\}$) where:

$$\begin{aligned}
\varphi_{init} &= (pc = \ell_1 \land x = 0 \land y = 0)\\
\rho_1 &= (pc = \ell_1 \land pc' = \ell_2 \land x' = x + 5 \land y' = y)\\
\rho_2 &= (pc = \ell_2 \land pc' = \ell_3 \land x' = x + 1 \land y' = x')\\
\varphi_{error} &= (pc = \ell_3 \land y \leq 5)
\end{aligned}$$

1. For the set of predicates $P = \{pc = \ell_1, pc = \ell_2, pc = \ell_3, x \geq 0, y \geq 5\}$, show that the abstraction along the path $\rho_1\rho_2$ reaches an error state, i.e., $post^{\#}(post^{\#}(\alpha(\varphi_{init}), \rho_1), \rho_2) \land \varphi_{error} \not\models false$.

$$\begin{aligned}
&\alpha(\varphi_{init}) &&= (pc = \ell_1 \land x \geq 0)\\
&post^{\#}(\alpha(\varphi_{init}), \rho_1) &&= post^{\#}(pc = \ell_1 \land x \geq 0, pc = \ell_1 \land pc' = \ell_2 \land x' = x + 5 \land y' = y)\\
& &&= post^{\#}(pc = \ell_2 \land x \geq 5) = (pc = \ell_2 \land x \geq 0)\\
&post^{\#}(post^{\#}(\alpha(\varphi_{init}), \rho_1), \rho_2) &&= post^{\#}(pc = \ell_2 \land x \geq 0, pc = \ell_2 \land pc' = \ell_3 \land x' = x + 1 \land y' = x')\\
& &&= post^{\#}(pc = \ell_3 \land x \geq 1 \land y = x) = (pc = \ell_3 \land x \geq 0)\\
&post^{\#}(post^{\#}(\alpha(\varphi_{init}), \rho_1), \rho_2) \land \varphi_{error} &&= (pc = \ell_3 \land x \geq 0) \land (pc = \ell_3 \land y \leq 5)\\
& &&= (pc = \ell_3 \land x \geq 0 \land y \leq 5)
\end{aligned}$$

$(pc = \ell_3 \land x \geq 0 \land y \leq 5)$ is satisfiable which implies that $(pc = \ell_3 \land x \geq 0 \land y \leq 5) \not\models false$.

2. Check if the path without abstraction reaches an error state.

$$\begin{aligned}
&\varphi_{init} &&= (pc = \ell_1 \land x = 0 \land y = 0)\\
&post(\varphi_{init}, \rho_1) &&= post(pc = \ell_1 \land x = 0 \land y = 0, pc = \ell_1 \land pc' = \ell_2 \land x' = x + 5 \land y' = y)\\
& &&= (pc = \ell_2 \land x = 5 \land y = 0)\\
&post(post(\varphi_{init}, \rho_1), \rho_2) &&= post(pc = \ell_2 \land x = 5 \land y = 0, pc = \ell_2 \land pc' = \ell_3 \land x' = x + 1 \land y' = x')\\
& &&= (pc = \ell_3 \land x = 6 \land y = x)\\
&post(post(\varphi_{init}, \rho_1), \rho_2) \land \varphi_{error} &&= (pc = \ell_3 \land x = 6 \land y = x) \land (pc = \ell_3 \land y \leq 5)\\
& &&= (pc = \ell_3 \land x = 6 \land y = x \land y \leq 5)
\end{aligned}$$

$(pc = \ell_3 \land x = 6 \land y = x \land y \leq 5)$ is unsatisfiable which implies that $(pc = \ell_3 \land x = 6 \land y = x \land y \leq 5) \models false$.

3. Refine the set of predicates by finding ψ_1, ψ_2, and ψ_3 such that $\varphi_{init} \models \psi_1$, $\psi_1 \land \rho_1 \models \psi_2$, $\psi_2 \land \rho_2 \models \psi_3$, and $\psi_3 \land \varphi_{error} \models false$.

   We will apply interpolation to refine the set. Since we have two transition relations involved in reaching the error, we have three instances of interpolation to solve.

   (a) Let $\delta_1(v) = \exists v'v''(\rho_1(v, v') \circ \rho_2(v', v'')) \land \varphi_{err}(v'')$, we find $\psi_1(v)$ such that:
   $\varphi_{init}(v) \models \psi_1(v)$
   $\psi_1(v) \land \delta_1(v) \models false$

$$\begin{aligned}
\delta_1(v) &= \exists v'v''(\rho_1(v, v') \circ \rho_2(v', v'')) \land \varphi_{err}(v'')\\
&= \exists v'v''(pc = \ell_1 \land pc' = \ell_2 \land x' = x + 5 \land y' = y) \land (pc' = \ell_2 \land pc'' = \ell_3 \land x'' = x' + 1 \land y'' = x'') \land (pc'' = \ell_3 \land y'' \leq 5\\
&= \exists v''(pc = \ell_1 \land x'' - 1 \leq x + 5 \land y'' = x'' \land pc'' = \ell_3) \land (pc'' = \ell_3 \land y'' \leq 5)\\
&= (pc = \ell_1 \land x \leq -1)
\end{aligned}$$

   After replacing $\varphi_{init}(v)$ and $\delta_1(v)$ in the equation above, we get:

   $(pc = \ell_1 \land x = 0 \land y = 0) \models \psi_1(v)$
   $\psi_1(v) \land (pc = \ell_1 \land x \leq -1) \models false$

   and, then we find $\psi_1(v)$ as an interpolant of the formulas $(pc = \ell_1 \land x = 0 \land y = 0)$ and $(pc = \ell_1 \land x \leq -1)$.

   To make things simpler, we can leave out pc since it has the same value in both formulas. Then, we have the simplified task of finding interpolant for $(x = 0 \land y = 0)$ and $(x \leq -1)$. We can easily see that one interpolant is $x \geq 0$. Therefore, we have $\psi_1(v) = (x \geq 0)$. Since we have it already in the set of predicates, the set remains the same.

(b) Let $\delta_2(v) = \exists v' \rho_2(v, v') \wedge \varphi_{err}(v')$, we find $\psi_2(v)$ such that:
$\exists v'(\psi_1(v') \wedge \rho_1(v', v)) \models \psi_2(v)$
$\psi_2(v) \wedge \delta_2(v) \models false$

$$\begin{aligned}
\delta_2(v) &= \exists v' \rho_2(v, v') \wedge \varphi_{err}(v') \\
&= \exists v'(pc = \ell_2 \wedge pc' = \ell_3 \wedge x' = x + 1 \wedge y' = x') \wedge (pc' = \ell_3 \wedge y' \leq 5) \\
&= (pc = \ell_2 \wedge x \leq 4) \\
\exists v'(\psi_1(v') \wedge \rho_1(v', v)) &= (x' \geq 0) \wedge (pc' = \ell_1 \wedge pc = \ell_2 \wedge x = x' + 5 \wedge y = y') \\
&= (pc = \ell_2 \wedge x \geq 5)
\end{aligned}$$

After replacing these two formulas in the equation above, we get:

$(pc = \ell_2 \wedge x \geq 5) \models \psi_2(v)$
$\psi_2(v) \wedge (pc = \ell_2 \wedge x \leq 4) \models false$

and, then we find $\psi_2(v)$ as an interpolant of the formulas $(pc = \ell_2 \wedge x \geq 5)$ and $(pc = \ell_2 \wedge x \leq 4)$.

Like we did for the first case, we can leave out pc since it has the same value in both formulas. Then, we have the simplified task of finding interpolant for $(x \geq 5)$ and $(x \leq 4)$. One such interpolant is $x \geq 5$.
Therefore, we have $\psi_2(v) = (x \geq 5)$, and we refine the set of predicates into $P = \{pc = \ell_1, pc = \ell_2, pc = \ell_3, x \geq 0, y \geq 5, x \geq 5\}$ by adding $\psi_2(v)$ into P.

(c) In this last step, we simply compute ψ_3 such that:
$\exists v'(\psi_2(v') \wedge \rho_2(v', v)) \models \psi_3(v)$
$\psi_3(v) \wedge \varphi_{err}(v) \models false$

$$\begin{aligned}
\exists v'(\psi_2(v') \wedge \rho_2(v', v)) &= (x' \geq 5) \wedge (pc' = \ell_2 \wedge pc = \ell_3 \wedge x = x' + 1 \wedge y = x) \\
&= (pc = \ell_3 \wedge x \geq 6 \wedge y \geq 6)
\end{aligned}$$

After replacing this formula and $\varphi_{err}(v)$ in the equation above, we get:
$(pc = \ell_3 \wedge x \geq 6 \wedge y \geq 6) \models \psi_3(v)$
$\psi_3(v) \wedge (pc = \ell_3 \wedge y \leq 5) \models false$

and, then we find $\psi_3(v)$ as an interpolant of the formulas $(pc = \ell_3 \wedge x \geq 6 \wedge y \geq 6)$ and $(pc = \ell_3 \wedge y \leq 5)$.

We also leave out pc since it has the same value in both formulas. Then, we have the simplified task of finding interpolant for $(x \geq 6 \wedge y \geq 6)$ and $(y \leq 5)$. One such interpolant is $y \geq 6$.
Therefore, we have $\psi_3(v) = (y \geq 6)$, and we refine the set of predicates into $P = \{pc = \ell_1, pc = \ell_2, pc = \ell_3, x \geq 0, y \geq 5, x \geq 5, y \geq 6\}$ by adding $\psi_3(v)$ into P.

Therefore, we have refined P such that $P = \{pc = \ell_1, pc = \ell_2, pc = \ell_3, x \geq 0, y \geq 5, x \geq 5, y \geq 6\}$.

4. Check that $post^{\#}(post^{\#}(\alpha(\varphi_{init}), \rho_1), \rho_2) \models false$ when using the refined abstraction function.

$$\begin{aligned}
\alpha(\varphi_{init}) &= (pc = \ell_1 \wedge x \geq 0) \\
post^{\#}(\alpha(\varphi_{init}), \rho_1) &= post^{\#}(pc = \ell_1 \wedge x \geq 0, pc = \ell_1 \wedge pc' = \ell_2 \wedge x' = x + 5 \wedge y' = y) \\
&= post^{\#}(pc = \ell_2 \wedge x \geq 5) = (pc = \ell_2 \wedge x \geq 5) \\
post^{\#}(post^{\#}(\alpha(\varphi_{init}), \rho_1), \rho_2) &= post^{\#}(pc = \ell_2 \wedge x \geq 5, pc = \ell_2 \wedge pc' = \ell_3 \wedge x' = x + 1 \wedge y' = x') \\
&= post^{\#}(pc = \ell_3 \wedge x \geq 6 \wedge y \geq 6) = (pc = \ell_3 \wedge x \geq 5 \wedge y \geq 6) \\
post^{\#}(post^{\#}(\alpha(\varphi_{init}), \rho_1), \rho_2) \wedge \varphi_{error} &= (pc = \ell_3 \wedge x \geq 5 \wedge y \geq 6) \wedge (pc = \ell_3 \wedge y \leq 5) \\
&= (pc = \ell_3 \wedge x \geq 5 \wedge y \geq 6 \wedge y \leq 5)
\end{aligned}$$

$(pc = \ell_3 \wedge x \geq 5 \wedge y \geq 6 \wedge y \leq 5)$ is unsatisfiable which implies that $(pc = \ell_3 \wedge x \geq 5 \wedge y \geq 6 \wedge y \leq 5) \models false$.

Q2. Find an interpolant for $x \geq 5 \wedge z \geq y + x$ and $z \leq y + a \wedge a \leq 4$.

We compute the interpolant applying the algorithm:
Let $\phi = (x \geq 5 \wedge z \geq y + x)$ and $\psi = (z \leq y + a \wedge a \leq 4)$. After arranging and rewritting the formulas to only use the inequality $\leq$, we get $\phi = (-x \leq -5 \wedge x + y - z \leq 0)$ and $\psi = (-y + z - a \leq 0 \wedge a \leq 4)$. We represent the formulas in matrix form as follows:

$$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ a \end{pmatrix} \leq \begin{pmatrix} -5 \\ 0 \end{pmatrix} \qquad (for\ \phi)$$

$$\begin{pmatrix} 0 & -1 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ a \end{pmatrix} \leq \begin{pmatrix} 0 \\ 4 \end{pmatrix} \qquad (for\ \psi)$$

By using these matrices, we find a 1×4 matrix $(\lambda\ \mu)$ where λ and μ themselves are 1×2 matrices such that:

$$\begin{aligned} &\exists \lambda\ \exists \mu : \\ &\quad \lambda \geq 0 \wedge \mu \geq 0 \wedge \\ &(\lambda \quad \mu) \begin{pmatrix} -1 & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 \\ 0 & -1 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix} = 0 \wedge (\lambda \quad \mu) \begin{pmatrix} -5 \\ 0 \\ 0 \\ 4 \end{pmatrix} \leq -1 \end{aligned}$$

One solution can be $(\lambda\ \mu) = (1\ 1\ 1\ 1)$. We then compute the interpolant by applying λ which is $(1\ 1)$ on the coeficient matrix of ϕ. i.e.

$$\begin{aligned} i &= (\lambda) \begin{pmatrix} -1 & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & -1 & 0 \end{pmatrix} \\ i_0 &= (\lambda) \begin{pmatrix} -5 \\ 0 \end{pmatrix} = -5 \end{aligned}$$

Therefore, the interpolant we computed is:

$$ix \leq i_0 \equiv \begin{pmatrix} 0 & 1 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ a \end{pmatrix} \leq -5 \equiv (y - z \leq -5)$$

Mini-test 6 Given the program text:

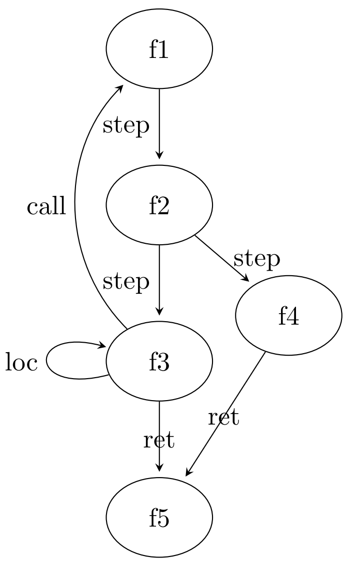
```
int x;
int f(int a) {
  int b;
f1:  b = a+b;
f2:  if (a>=0)
f3:     { f(a-1);}
     else
f4:     { return b+x; }
f5: }
```

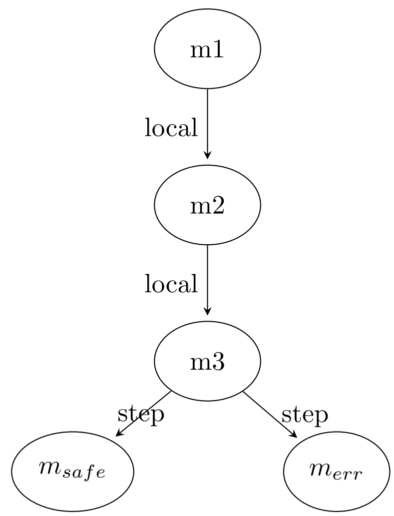
```
void main(void) {
  int n, m;
  assume(x>=0);
m1:  n = f(x);
m2:  m = f(-x);
m3:  assert(m==n);
 }
```

Q1. Give global variables of the program and the local variables of each procedure.

Globals: x, ret
Locals of f: a, b
Locals of $main$: m, n

Q2. Give a control flow graphs for each procedure.





Q3. For procedure f, give the transition relations $step_f$, $call_{f,f}$, ret_f, and $local_f$.

- $step_f =$
  $(pc_f = f_1 \wedge b' = a + b \wedge a' = a \wedge x' = x \wedge pc'_f = f_2) \vee$
  $(a \geq 0 \wedge pc_f = f_2 \wedge b' = b \wedge a' = a \wedge x' = x \wedge pc'_f = f_3) \vee$
  $(a < 0 \wedge pc_f = f_2 \wedge b' = b \wedge a' = a \wedge x' = x \wedge pc'_f = f_4)$
- $call_{f,f} = (pc_f = f_3 \wedge pc'_f = f_1 \wedge a' = a - 1)$
- $ret_f = (pc_f = f_3 \wedge ret' = ret) \vee (pc_f = f_4 \wedge ret' = b + x)$
- $local_f = true$

Q4. For procedure $main$, give the transition relations $step_{main}$, $call_{main,f}$, ret_{main}, and $local_{main}$.

- $step_{main} =$
  $(pc_{main} = m_3 \wedge m = n \wedge m' = m \wedge n' = n \wedge x' = x \wedge pc'_{main} = m_{safe}) \vee$
  $(pc_{main} = m_3 \wedge m \neq n \wedge m' = m \wedge n' = n \wedge x' = x \wedge pc'_{main} = m_{err})$
- $call_{main,f} = (pc_{main} = m_1 \wedge pc_f = f_1 \wedge a' = x) \vee$
  $(pc_{main} = m_2 \wedge pc_f = f_1 \wedge a' = \neg x)$
- $ret_f = (pc_{main} = m_2 \wedge n' = ret) \vee (pc_{main} = m_3 \wedge m' = ret)$

- $local_f = (pc_{main} = m_1 \wedge pc'_{main} = m_2 \wedge m' = m) \vee$
  $(pc_{main} = m_2 \wedge pc'_{main} = m_3 \wedge n' = n)$

Q5. We consider a call site where p calls q. Show rules that present changes to the summaries of p and q respectively:

1. Summeraization inference rule for $summ_p$

$$\frac{\begin{array}{ccc} ((g, l_p), (g', l'_p)) \in summ_p & & ((g', l'_p, l_q)) \models call_{p,q}(V_G, V_p, V_q) \\ ((g', l_q), (g'', l'_q)) \in summ_q & (g'', l'_q, q''') \models ret_q(V_G, V_q, V'_G) & (l'_p, l''_p) \models loc_p(V_p, V'_p) \end{array}}{((g, l_p), (g''', l''_p)) \in summ_p}$$

(Or as entailment)

$$\begin{aligned} &summ_p((V_G, V_p), (V'_G, V'_p)) \wedge call_{p,q}((V'_G, V'_p, V_q)) \wedge summ_q((V'_G, V_q), (V''_G, V'_q)) \wedge \\ &\quad ret_q(V''_G, V'_q, V'''_G) \wedge loc_p(V'_p, V''_p) \models summ_p((V_G, V_p), (V'''_G, V''_p)) \end{aligned}$$

2. Summeraization inference rule for $summ_q$

$$\frac{((g, l_p), (g', l'_p)) \in summ_p \qquad ((g', l'_p, l_q)) \models call_{p,q}(V_G, V_p, V_q)}{((g', l_q), (g', l_q)) \in summ_q}$$

(Or as entailment)

$$summ_p((V_G, V_p), (V'_G, V'_p)) \wedge call_{p,q}((V'_G, V'_p, V_q)) \models summ_q((V'_G, V_q), (V'_G, V_q))$$

Mini-test 7

Q1 Given a mutual exclusion algorithm for 2 threads:

```
initially turn∈ {1,2} ∧ Q1 = Q2 = false
// Thread 1:                     || // Thread 2:
A: Q1:=true;                     || A: Q2:=true;
B: turn:=2;                      || B: turn:=1;
C: await ⟨¬Q2 ∨ turn = 1⟩;       || C: await ⟨¬Q1 ∨ turn = 2⟩;
D: Q1 := false; goto A;          || D: Q2 := false; goto A;
```

1. Is mutual exclusion for locations D still satisfied? Why or why not?

   The strongest inductive invariant is:

$$
\begin{aligned}
I = \quad & (PC_1 = A \wedge PC_2 = A \wedge \neg Q_1 \wedge \neg Q_2) && \vee\ (PC_1 = B \wedge PC_2 = A \wedge Q_1 \wedge \neg Q_2) \vee \\
& (PC_1 = A \wedge PC_2 = B \wedge \neg Q_1 \wedge Q_2) && \vee\ (PC_1 = C \wedge PC_2 = A \wedge Q_1 \wedge \neg Q_2 \wedge turn = 2) \vee \\
& (PC_1 = B \wedge PC_2 = B \wedge Q_1 \wedge Q_2) && \vee\ (PC_1 = A \wedge PC_2 = C \wedge \neg Q_1 \wedge Q_2 \wedge turn = 1) \vee \\
& (PC_1 = D \wedge PC_2 = A \wedge Q_1 \wedge \neg Q_2 \wedge turn = 2) && \vee\ (PC_1 = C \wedge PC_2 = B \wedge Q_1 \wedge Q_2 \wedge turn = 2) \vee \\
& (PC_1 = B \wedge PC_2 = C \wedge Q_1 \wedge Q_2 \wedge turn = 1) && \vee\ (PC_1 = A \wedge PC_2 = D \wedge \neg Q_1 \wedge Q_2 \wedge turn = 1) \vee \\
& (PC_1 = D \wedge PC_2 = B \wedge Q_1 \wedge Q_2 \wedge turn = 2) && \vee\ (PC_1 = C \wedge PC_2 = C \wedge Q_1 \wedge Q_2 \wedge turn = 1) \vee \\
& (PC_1 = C \wedge PC_2 = C \wedge Q_1 \wedge Q_2 \wedge turn = 2) && \vee\ (PC_1 = B \wedge PC_2 = D \wedge Q_1 \wedge Q_2 \wedge turn = 1) \vee \\
& (PC_1 = D \wedge PC_2 = C \wedge Q_1 \wedge Q_2 \wedge turn = 1) && \vee\ (PC_1 = C \wedge PC_2 = D \wedge Q_1 \wedge Q_2 \wedge turn = 2) \vee
\end{aligned}
$$

   and, we can see that there is no any state that satisfies $(PC_1 = D \wedge PC_2 = D)$. To access location D simultaneously, both $(\neg Q_2 \vee turn = 1)$ and $(\neg Q_1 \vee turn = 2)$ must hold. But, we know that $Q_1 = Q_2 = true$ when both threads want to access location D. Therefore, to access the location $(turn = 1 \wedge turn = 2)$ must hold, which can never be satisfied.

2. Compute the number of states of the protocol.
   (Hint: $State = Shared \times \prod_{i=1}^{|\mathrm{Tid}|} Local_i$).

   There are three shared variables, Q_1, Q_2 and $turn$, each with two possible values, and one local variable PC with four possible values for each thread.
   $|States| = (2 \times 2 \times 2) \times 4 \times 4 = 128$

Q2 Given a mutual exclusion algorithm for 2 threads:

```
initially t=s=0;
// Thread 1:                     || // Thread 2:
while(true){                     || while(true){
    A: a=t;                      ||     A: b=t;
       t:=t+1;                   ||        t:=t+1;
    B: await <a=s>;              ||     B: await <b=s>;
    C: //critical section        ||     C: //critical section
       s:=s+1;                   ||        s:=s+1;
}                                || }
```

and, an inductive invariant:

$$
\begin{aligned}
I = \quad & (pc_1 = A \wedge pc_2 = A \ \wedge t = s) \vee && (D_1(v)) \\
& (pc_1 = A \wedge pc_2 = B \ \wedge t = s + 1 \wedge b = t - 1) \vee && (D_2(v)) \\
& (pc_1 = A \wedge pc_2 = C \ \wedge t = s + 1 \wedge b = s) \vee && (D_3(v)) \\
& (pc_1 = B \wedge pc_2 = A \ \wedge t = s + 1 \wedge a = t - 1) \vee && (D_4(v)) \\
& (pc_1 = B \wedge pc_2 = B \ \wedge t = s + 2 \wedge a = t - 1 \wedge b = t - 2) \vee && (D_5(v)) \\
& (pc_1 = B \wedge pc_2 = B \ \wedge t = s + 2 \wedge a = t - 2 \wedge b = t - 1) \vee && (D_6(v)) \\
& (pc_1 = B \wedge pc_2 = C \ \wedge t = s + 2 \wedge a = t - 1 \wedge b = s) \vee && (D_7(v)) \\
& (pc_1 = C \wedge pc_2 = B \ \wedge t = s + 2 \wedge a = s \wedge b = t - 1) \vee && (D_8(v)) \\
& (pc_1 = C \wedge pc_2 = A \ \wedge t = s + 1 \wedge a = s) && (D_9(v))
\end{aligned}
$$

Prove the stability of I under the transitions:
(for the sake of reference, each disjunct of the invariant is given a name).

1. $A \to B$ of $thread_1$

   We first represent the transition formally as $\rho(v, v') = (pc_1 = A \wedge pc_1' = B \wedge a' = t \wedge t' = t + 1 \wedge b' = b \wedge pc_2' = pc_2)$, and then apply *post* over the inductive invariant to check if the computed states are in the invariant or not. This transition is appicable only on the disjuncts $D_1(v)$, $D_2(v)$, and $D_3(v)$.

$$\begin{aligned}
post(D_1, \rho) &= post(pc_1 = A \wedge pc_2 = A \wedge t = s, pc_1 = A \wedge pc_1' = B \wedge a' = t \wedge t' = t + 1 \wedge b' = b \wedge pc_2' = pc_2) \\
&= (pc_1 = B \wedge pc_2 = A \wedge t = s + 1 \wedge a = t - 1) \models D_4 \\
post(D_2, \rho) &= post(pc_1 = A \wedge pc_2 = B \wedge t = s + 1 \wedge b = t - 1, pc_1 = A \wedge pc_1' = B \wedge a' = t \wedge t' = t + 1 \wedge b' = b \wedge pc_2' = pc_2) \\
&= (pc_1 = B \wedge pc_2 = B \wedge t = s + 2 \wedge a = t - 1 \wedge b = t - 2) \models D_5 \\
post(D_3, \rho) &= post(pc_1 = A \wedge pc_2 = C \wedge t = s + 1 \wedge b = s, pc_1 = A \wedge pc_1' = B \wedge a' = t \wedge t' = t + 1 \wedge b' = b \wedge pc_2' = pc_2) \\
&= (pc_1 = B \wedge pc_2 = C \wedge t = s + 2 \wedge a = t - 1 \wedge b = s) \models D_7
\end{aligned}$$

   We can see that application of this transition results in the states that are already in the inductive invariant. Therefore, the invariant I is stable under the transition.

2. $C \to A$ of $thread_2$

   The transition is represented formally as $\rho(v, v') = (pc_2 = C \wedge pc_2' = A \wedge s' = s + 1 \wedge pc_1' = pc_1 \wedge a' = a \wedge b' = b)$, and it is applicable on the disjuncts $D_3(v)$, and $D_7(v)$.

$$\begin{aligned}
post(D_3, \rho) \;&= post(pc_1 = A \wedge pc_2 = C \wedge t = s + 1 \wedge b = s, pc_2 = C \wedge pc_2' = A \wedge s' = s + 1 \wedge pc_1' = pc_1 \wedge a' = a \wedge b' = b) \\
&= (pc_1 = A \wedge pc_2 = A \wedge t = s \wedge b = s - 1) \models D_1 \\
post(D_7, \rho) \;&= post(pc_1 = B \wedge pc_2 = C \wedge t = s + 2 \wedge a = t - 1 \wedge b = s, pc_2 = C \wedge pc_2' = A \wedge s' = s + 1 \wedge pc_1' = pc_1 \wedge \\
&\quad a' = a \wedge b' = b) \\
&= (pc_1 = B \wedge pc_2 = A \wedge t = s + 1 \wedge a = t - 1 \wedge b = s - 1) \models D_4
\end{aligned}$$

   Here also the transition results in the states that are already in the invariant I. Therefore, the invariant I is stable under this transition as well.

Mini-test 8

In all the exercises, if you use abbreviations, define them upfront. The symbol $\mathfrak{P}$ denotes the powerset.

Q1. Consider the two-threaded program given by the following code:

```
                 initially x=0
// Thread 1:                     || // Thread 2:
A: ⟨ await x=0;   x:=1 ⟩        || A: ⟨ await x=0;   x:=2 ⟩
B:                               || B:
```

Q1.1. Give a suitable formal representation of the above program.

$$\text{Tid} = \{1, 2\} \quad ,$$

$$\text{Shared} = (\{x\} \mapsto \{0, 1, 2\}) \quad ,$$

$$\text{Local}_1 = (\{pc_1\} \mapsto \{A, B\}) \quad ,$$

$$\text{Local}_2 = (\{pc_2\} \mapsto \{A, B\}) \quad ,$$

$$\rightarrow_1 = \{(([x \mapsto 0], [pc_1 \mapsto A]), ([x \mapsto 1], [pc_1 \mapsto B]))\} \quad ,$$

$$\rightarrow_2 = \{(([x \mapsto 0], [pc_2 \mapsto A]), ([x \mapsto 2], [pc_2 \mapsto B]))\} \quad ,$$

$$init = \{([x \mapsto 0], [pc_1 \mapsto A], [pc_2 \mapsto A])\} \quad .$$

Q1.2. Prove mutual exclusion for locations `B` (i.e., in any reachable state the threads cannot simultaneously have control flow location `B`) by thread-modular verification. For your reference, the thread-modular inference rules are

$$\text{(INIT)}\ \frac{t \in \text{Tid} \qquad (g, l) \in init}{(g, l_t) \in R_t} \qquad \text{(STEP)}\ \frac{t \in \text{Tid} \qquad (g, l) \in R_t \qquad (g, l) \rightarrow_t (g', l')}{(g', l') \in R_t \qquad (g, g') \in G_t}$$

$$\text{(ENV)}\ \frac{t \in \text{Tid} \qquad (g, l) \in R_t \qquad \hat{t} \in \text{Tid} \setminus \{t\} \qquad (g, g') \in G_{\hat{t}}}{(g', l) \in R_t}$$

and the multithreaded Cartesian concretization is

$$\gamma_{\text{mc}} : \prod_{t \in \text{Tid}} \mathfrak{P}(\text{Shared} \times \text{Local}_t) \rightarrow \text{State}\,,$$
$$\gamma_{\text{mc}}((S_t)_{t \in \text{Tid}}) = \{(g, l) \in \text{State} \mid \forall\, t \in \text{Tid} \colon (g, l_t) \in S_t\}\,.$$

$R_1 = \{([x \mapsto 0], [pc_1 \mapsto A]), ([x \mapsto 1], [pc_1 \mapsto B]), ([x \mapsto 2], [pc_1 \mapsto A])\}$, $R_2 = \{([x \mapsto 0], [pc_2 \mapsto A]), ([x \mapsto 2], [pc_2 \mapsto B]), ([x \mapsto 1], [pc_2 \mapsto A])\}$,

$G_1 = \{([x \mapsto 0], [x \mapsto 1])\}$, $G_2 = \{([x \mapsto 0], [x \mapsto 2])\}$,

$\text{Reach}^\# = \gamma_{\text{mc}}((R_t)_{t \in \text{Tid}}) = \{([x \mapsto 0], [pc_1 \mapsto A], [pc_2 \mapsto A]), ([x \mapsto 1], [pc_1 \mapsto B], [pc_2 \mapsto A]), ([x \mapsto 2], [pc_1 \mapsto A], [pc_2 \mapsto B])\}$

It can be seen that there is no reachable state where both threads are at location B.

Q2. Consider an arbitrary multithreaded program. Let State be the set of states of the program and $\to$ be its transition relation. Let *post* be the successor operator

$$\begin{aligned} \mathit{post} : \mathfrak{P}(\text{State}) &\to \mathfrak{P}(\text{State})\,, \\ \mathit{post}(Q) &= \{q' \mid \exists\, q \in Q \colon q \to q'\}\,. \end{aligned}$$

Let $f\colon\ \mathfrak{P}(\text{State}) \to \mathfrak{P}(\text{State}),\ f(Q) = \mathit{init} \cup \mathit{post}(Q)$.

Q2.1. Show that f is monotone with respect to inclusion.
Formally, show that that for all $Q, Q' \in \mathfrak{P}(\text{State})$ we have $Q \subseteq Q' \Rightarrow f(Q) \subseteq f(Q')$.

Let $Q \subseteq Q'$. We will show that $f(Q)$ is a subset of $f(Q')$. Let $q' \in f(Q)$. If $q' \in \mathit{init}$, then $q' \in f(Q')$. Otherwise, $q' \in \mathit{post}(Q)$, so there is $q \in Q$ such that $q \to q'$. Then, $q \in Q'$. $q' \in \mathit{post}(Q') \subseteq f(Q')$.

Q2.2. Does f have a fixpoint? Formally, is there $Q \in \mathfrak{P}(\text{State})$ such that $f(Q) = Q$?

Yes, since monotone maps on a complete lattice always have fixpoints. And, $(\mathfrak{P}(\text{State}), \subseteq)$ is a complete lattice.

Mini-test 9

In all the exercises, if you use abbreviations, define them upfront. ˜x denotes arithmetic negation.

Q1 Given a typing environment $T := \{abs := int \rightarrow int\}$, infer the type using the inference rules in the handout (construct the tree):

```
let val x = ~3 in abs x end
```

```
T |- ~ : int -> int     T |- 3 : int   T, {x : int} |- abs : int -> int    T, {x : int} |- x : int
------------------------------------   -------------------------------------------------------------
T |> val x = ~ 3 : T, {x : int}        T, {x : int} |- abs x : int
------------------------------------------------------------------
T |- let val x = ~3 in abs x end : int
```

Q2 Which sequence of typing environments is obtained by typing the following declarations starting with the empty typing environment:

(a) `val b = true`

{b : bool}

(b) `val x = let fun square x = x*x in square 5 end`

{b : bool, x : int}

Q3 Which sequence of value environments is obtained by evaluating the following declarations starting with the empty value environment:

(a) `val y = 3`

[y := 3]

(b) `fun max x = if x > y then true else false`

[y := 3, max := (fun max x = if x > y then true else false, [y := 3])]

(c) `val y = max y`

[y := false, max := (fun max x = if x > y then true else false, [y := 3])]

(d) `fun fact y = if y < 1 then 1 else y*fact(y - 1)`

[y := false, max := (fun max x = if x > y then true else false, [y := 3]), fact = (fun fact y = if y < 1 then 1 else y*fact(y - 1), [])]

Q4 Formalize as a refinement type:

f is a function that takes as input a positive integer x and returns an integer that is greater than or equals the value of x, but is smaller than the value of identifier y.

$f : (x : \{v : int|v > 0\} - > \{v : int|v \geq x \wedge v < y\})$

Q5 Given a value environment $V := [y := 10]$, evaluate using the inference rules in the handout (construct the tree):

```
let val z = y*y in if z>y then true else false end
```

```
                                  V, [z:=100] |= z :100     V, [z :=100] |= y : 10
                                  ------------------------------------------------
V |= y : 10    V |= y : 10             V, [z :=100] |= z > y : true       V, [z :=100] |= true : true
--------------------------------      ---------------------------------------------------------------
V |>> val z = y*y : V, [z :=100]      V, [z :=100] |= if z>y then true else false : true
-----------------------------------------------------
V |= let val z = y*y in if z>y then true else false end : true
```