Exercise 1: Properties of post#  
Give a counterexample for those of the following propositions that are wrong.

(a) $\text{post#}((\varphi, \rho_1 \circ \rho_2)) \subseteq \text{post#}((\varphi, \rho_2), \rho_1)$
(b) $\text{post#}((\varphi, \rho_1 \circ \rho_2)) \supseteq \text{post#}((\varphi, \rho_2), \rho_1)$
(c) $\text{post#}((\varphi, \rho_1 \lor \rho_2)) \subseteq \text{post#}((\varphi, \rho_1) \lor \text{post#}((\varphi, \rho_2))$
(d) $\text{post#}((\varphi, \rho_1 \lor \rho_2)) \supseteq \text{post#}((\varphi, \rho_1) \lor \text{post#}((\varphi, \rho_2))$
(e) $\text{post#}(\varphi_1 \lor \varphi_2, \rho) \subseteq \text{post#}(\varphi_1, \rho_1) \lor \text{post#}(\varphi_2, \rho)$
(f) $\text{post#}(\varphi_1 \lor \varphi_2, \rho) \supseteq \text{post#}(\varphi_1, \rho_1) \lor \text{post#}(\varphi_2, \rho)$

Exercise 2: AbstReach  
Consider the following program.

```c
int x, y, z, w;
void foo() {
    do {
        z := 0;
        x := y;
        if (w == 17){
            x++; 
            z := 1;
        }
    } while (x != y);
    assert (z != 1);
}
```

Solve the following tasks without explicitly executing the procedure AbstREACH (unless you have a lot of free time and paper).

(a) Is the program safe? Give an intuitive argument.

(b) Give three predicates (in addition to the predicates on the program counter) such that the corresponding abstraction is sufficient to prove safety. Give the corresponding abstract reachability graph (in an informal representation where the edges are labeled by line numbers).
(c) Give the abstract reachability graph that corresponds to the abstraction for the set of predicates \( \text{Pred}_0 \) which contains only the predicates on the program counter. Take the shortest counterexample path. Add one predicate \( p_1 \) to eliminate this first counterexample.

(d) Give the abstract reachability graph that corresponds to the abstraction for the set of predicates \( \text{Pred}_1 := \text{Pred}_0 \cup \{p_1\} \). Take again the shortest counterexample path. Add two predicates \( p_2 \) and \( p_3 \) to eliminate this counterexample. (Did you get the three predicates from (b)?)

**Exercise 3: State space explosion**

Consider the procedure \( \text{AbstReach} \). Let \( n := |\text{Preds}| \) be the number of predicates. Let \( m := |\mathcal{R}| \) be the number of transitions of the program.

(a) How many abstract reachable states (elements of \( \text{ReachStates}^\# \)) are there in the worst case?

(b) How many times do we check validity of an implication \( \varphi \models p \) in the worst case?

(c) Let us roughly estimate the maximal number of predicates a tool can deal with (in the worst case). Consider the following setting: We have an implementation of \( \text{AbstReach} \) that may use up to 4 gibibyte, one abstract state needs 32 byte and we neglect the memory necessary for all other data (e.g., the \text{Parent} relation). What is the maximal number of predicates \( n_{\text{max}} \) such that our implementation of \( \text{AbstReach} \) does not run out of memory? Consider the worst case scenario from part (a).

(d) Let us roughly estimate the runtime of \( \text{AbstReach} \) for \( n_{\text{max}} \) predicates. Consider the following setting: We have \( m = 1000 \) relations. The theorem prover always needs exactly one millisecond to decide validity of an implication \( \varphi \models p \). If we neglect the runtime of all components but the theorem prover, how much time does it take in the worst case to compute the set of all reachable abstract states? Consider the worst case scenario from part (b).

(e) Suggest an optimization for the \( \text{AbstRefineLoop} \) algorithm that can reduce the number of abstract states.