Question 1 Prove that assigning Volpano’s information flow type of class 13 to a program is undecidable (and so any type system like Volpano’s information flow type system will reject infinitely many programs with no security leak but which cannot be proved to be typable by the type system).

Answer 1 Assume by reductio ad absurdum that any program is typable by Volpano’s information flow typing of class 13 if and only if it has no security leak. Then let $P$ be any program and $x$ and $y$ be two variables never appearing in $P$. Assume that $x$ is public and $y$ is private. Consider the following program

$$P' \triangleq P; \quad x := y;$$

Then we would have “$P$ is not typable by Volpano’s information flow typing of class 13” if and only if “$P'$ has an information leak” if and only if “$P$ does terminate”, which is undecidable, a contradiction.

Question 2 Provide an example of a very simple program which is secure but

- fails to be proved secure by the Dennis flow analysis of class 12;
- fails to be typed secure by the Volpano-Irvine-Smith type system of class 13;
- is provably secure by the security static analysis of class 14.

Answer 2 Consider confidentiality for the following program with two integer variables:

```plaintext
public := secret - secret;
```

Dennis flow analysis and the Volpano-Irvine-Smith type system evaluate expression "secret - secret" to the security class secret and refuse the assignment to "public" which would be a flow down (no write down). On the contrary, the security static analysis evaluate expression "secret - secret" to "0" of class public which can be assigned without problem to "public".
**Question 3** Consider the following erroneous rule for constants (L is low, the infimum of the security lattice) for the static program flow analysis of class 12.

\[
\begin{align*}
\{1:\} & \ a := \text{cte} \\
\{2:\} & \ PC_2 = PC_1 \\
& \ I_2(a) = L
\end{align*}
\]

Provide a counter-example to this rule. □

**Answer 3** This rule is an under-approximation, since if it is in a test such as

\[
\text{if (private } == 0) \ a = 0; \text{ else } a = 1;
\]

then private information has been propagated to \(a\) so in the rule this information must be propagated to \(PC\) and then to \(a\) as follows

\[
\begin{align*}
\{1:\} & \ a := \text{cte} \\
\{2:\} & \ PC_2 = PC_1 \\
& \ I_2(a) = PC_1
\end{align*}
\]

**Question 4** Consider the rule for conditional in the static program flow analysis of class 12.

\[
\begin{align*}
\{1:\} & \ \text{if } B(o_1, \ldots, o_n) \text{ then} \\
& \{2:\} \ S \{3:\} \\
& \{4:\} \ S' \{5:\} \\
& \{6:\} \ fi
\end{align*}
\]

\[
\begin{align*}
PC_2 & = PC_1 \oplus I_1(o_1) \oplus \ldots \oplus I_1(o_n) \\
I_2(o_i) & = PC_2, \ i = 1, \ldots, n \\
PC_4 & = PC_1 \oplus I_1(o_1) \oplus \ldots \oplus I_1(o_n) \\
I_4(o_i) & = PC_4, \ i = 1, \ldots, n \\
PC_6 & = PC_1 \\
I_6(o_i) & = I_3(o_i) \oplus I_5(o_i), \ i = 1, \ldots, n
\end{align*}
\]

Explain why the information leak, if any, in the test \(B(o_1, \ldots, o_n)\) can propagate beyond program \(6:\) despite the fact that \(PC_6 = PC_1\) stating that the information leaked after the test is the same as that leaked before the test. □

**Answer 4** The leak of information can only be through assignment to variables in the “then” or “else” branches of the conditional. Without such assignments which branch as been taken is not be recorded at all. The information in \(B(o_1, \ldots, o_n)\) will propagate to \(PC_2/PC_4\) then to some \(I_2(o_i)/I_4(o_j)\) hence to some \(I_6(o_k) = I_3(o_k) \oplus I_5(o_k)\) and so will be recorded at point \(6:\) in some variables \(o_k\) though a series of successive assignments (and tests). □

**Question 5** Consider again the static program flow analysis of class 12. Propose a rule for the local variable declaration which scope/visibility is the following block of declaration.

\[
\begin{align*}
\{1:\} & \ a := \text{cte} \\
\{2:\} & \ PC_2 = PC_1 \\
& \ I_2(a) = L
\end{align*}
\]
Note that if one of the variables $o_1, \ldots, o_n$ is $x$ in the initialization of the declaration $\text{var } x := f(o_1, \ldots, o_n)$; then it cannot be the new fresh $x$ local to the block but it must be one of the global variables $o_1, \ldots, o_n$ visible at program point $\{1:\}$. However in statement $S$, any instance of $x$ is that of the local variable (hiding the global one amongst the $o_1, \ldots, o_n$, if any, which is not visible in the block beyond point $\{2:\}$ after the declaration of the fresh variable $x$).

**Answer 5**

$\{1:\}$ begin
\[
\begin{align*}
\text{var } x & := f(o_1, \ldots, o_n); \\
\end{align*}
\]
\{2:\} $\quad$ \begin{align*}
PC_2 &= PC_1 \\
I_2(x) &= I_1(o_i) \oplus \ldots \oplus I_1(o_n) \\
I_2(o_i) &= I_1(o_i), \ i = 1, \ldots, n, \ o_i \neq x
\end{align*}
$\quad$ \begin{align*}
S \quad \{3:\}
\end{align*}
\{3:\} $\quad$ \begin{align*}
PC_4 &= PC_1 \\
I_4(o_i) &= I_3(o_i), \ i = 1, \ldots, n, \ o_i \neq x \\
I_4(x) &= I_1(x) \quad \text{(if } x \text{ is visible at } \{1:\} \text{ and } \{4:\})
\end{align*}
\}