CS5430 Homework 5: Information Flow Control

General Instructions. You are expected to work alone on this assignment.
Due: April 17 at 11:59pm. No late assignments will be accepted.
Submit your solution using CMS. Prepare your solution as a single .pdf.

For this set of problems:

- use lattice $\langle\{\mathrm{L}, \mathrm{H}\}, \sqsubseteq\rangle$ of labels, where $\mathrm{L} \sqsubseteq \mathrm{H}$
- employ the following simpler definition of *GM-Noninterference* (NI):
    - A program `c` satisfies NI iff for any two finite executions of `c` that start with memories agreeing on values in low variables (i.e., variables tagged with L), both executions terminate with memories agreeing on values in low variables.

Problem 1

Consider the following program:

```
if x>0 then y:=z+1 else x:=w;
w:=z
```

with a fixed environment Γ, such that Γ(`w`)= H. Give the least restrictive labels for Γ(`x`), Γ(`y`), and Γ(`z`) such that the program is type correct, with initial context label ctx= L, based on the static type system introduced in class. Include the proof tree that establishes the type-correctness of this program for the chosen labels above.

Problem 2

Consider the syntax of a **for-loop** command

```
for x in 1:y do c
```

The operational semantics of this command is:

- If 1≤`y` holds, then `x` is initialized to 1 and there will be `y` iterations of `c`. At the beginning of the ith iteration, for 1≤i≤`y`, the value of `x` will be i.
- Otherwise, `c` does not execute.

Justify your answers to the following questions. Feel free to introduce variables and declare their labels.

i. Give a **for-loop** command that satisfies noninterference NI, as defined above. Body `c` should not be empty. Make sure to give labels for all variables that appear in the command you choose.
ii. Give a **for-loop** command that does NOT satisfy NI. Make sure to give labels for all variables that appear in the command you choose.
iii. Give a rule that extends the static type system described in class to handle **for-loop** commands. Explain why a **for-loop** command that is accepted by the proposed rule satisfies NI.

iv. Give an example that shows that your extended type-system is conservative because the **for-loop** rule fails for a program that satisfies NI.

Problem 3

We might extend the syntax of the programming language by adding a `break` command. When a `break` within a **while**-command is executed, execution jumps to the command following that **while**-command. Assume for simplicity that **while**-commands are not enclosed within other **while**-commands.

i. Give a program that does not satisfy NI (as defined above) as a result of using a `break` in a **while**-command, even though your program would satisfy NI if that `break` were replaced by `skip`. Justify your answer and make sure to give labels to all variables that appear in the command you choose.

ii. Assume that we extend the static type system introduced in class with the following type rule for `break`:

$$\frac{}{\Gamma, ctx \vdash \texttt{break}}$$

So, `break` is always type correct. Is the program proposed above accepted by the extended type system?

iii. Propose a type rule for `break` such that the extended type system is sound: type-correct programs satisfy NI. Is the program proposed in the first question now accepted by the extended type system?

Problem 4

Consider a dynamic enforcement mechanism (for information flow) that uses a fixed environment Γ. This enforcement mechanism works as follows:

When execution reaches assignment `x:=e` with context label ctx, check whether $ctx \sqcup \Gamma(\mathrm{e}) \sqsubseteq \Gamma(\mathrm{x})$ holds. If the check succeeds, then the assignment is performed; otherwise execution is blocked (without performing the assignment).

When execution reaches a conditional command, the context label is augmented with the label of the guard of that command, and the taken branch is then executed.

Note, there is no “on-the-fly” analysis of an untaken branch.

i. Can labels on variables encode sensitive information during execution under this dynamic mechanism?

ii. Information can be leaked by the decision of this dynamic enforcement mechanism. Give an example.

iii. What restriction on guards could be applied to prevent leaks though blocking by this dynamic mechanism?

Extra credit:

Justify your answers to each of the following.

i. We would like to modify the static type system introduced in class to handle a flow-sensitive environment Γ (instead of a fixed Γ). A flow-sensitive Γ might change its assignment of labels to variables after the analysis of each command and become Γ'. So, the judgment that seems to be needed would be of the form:

$$\Gamma, ctx \vdash \mathrm{c}\,, \Gamma'.$$

Give a type rule for assignment, **if**-command, and sequence of commands.

ii. Give a program for which the new flow-sensitive rules proposed above deduce conservative labels; variables could have been associated with less restrictive labels and the program would still satisfy NI.