Assignment 3
Proving Safety
Sample Solution

15-316: Software Foundations of Security & Privacy
Frank Pfenning

Due **Wednesday**, September 25, 2024
75 points

Note that this is a *sample solution*! There are often multiple correct ways to solve a problem and we do not try to comprehensive in any way.

1 Semantic Equivalence [20 points]

In the previous homework we proved program equivalence using the inference rules of the sequent calculus for dynamic logic. For programs that use **while** this is difficult or impossible with the tools we have so far, since our axiom for **while** is only an implication (not a bi-implication) and, consequently, only corresponds to a right rule ($[\mathbf{while}]R$) in the sequent calculus and not a left rule.

Task 1 (20 points) Consider the two formulas

$$[\mathbf{while}\ \top\ \mathbf{assert}\ \top]Q$$

and

$$[\mathbf{test}\ \bot]Q$$

Prove that they are *semantically* equivalent, that is, in any state ω, one is true if and only if the other is. For the reasons explained above, you should do this using the definition of $\omega \models P$ and other semantic definitions such as $\omega[\![\alpha]\!]\nu$ and $\omega[\![\alpha]\!]\lightning$.

Both of these are valid, no matter what Q is, as shown below. So both are equivalent to $\top$ and therefore to each other.

$\models [\mathbf{while}\ \top \mathbf{assert}\ \top]Q$

For any ω we have $\omega \models [\mathbf{while}\ \top\ \mathbf{assert}\ \top]Q$ because (1) there is no poststate ν such that $\omega[\![\mathbf{while}\ \top\ \mathbf{assert}\ \top]\!]\nu$ and (2) **not** $[\![\mathbf{while}\ \top \mathbf{assert}\ \top]\!]\lightning$.

To conclude (1), we prove by induction on n that $\omega[\![\mathbf{while}\ \top\ \mathbf{assert}\ \top]\!]^n\nu$ is impossible. In the case of $n = 0$, that is because $\omega \models \top$, but $\omega \not\models \top$ is required. In the case of $n > 0$ it follows by induction hypothesis on $\mu[\![\mathbf{while}\ \top\ \mathbf{assert}\ \top]\!]^{n-1}\nu$.

To conclude (2), we prove by induction on n that $\omega[\![\textbf{while } \top \textbf{ assert } \top]\!]^n \lightning$ is impossible. For $n = 0$ it is impossible by definition. For $n > 0$ there are two clauses (according to the definition on page L5.4). For the first, we find that $\omega[\![\textbf{assert } \top]\!] \lightning$ is impossible (by definition and $\omega \vdash \top$). For the second, we appeal to the induction hypothesis for $\mu[\![\textbf{while } P\ \alpha]\!]^{n-1} \lightning$

$\models [\textbf{test } \bot]Q$

For any ω we have $\omega \models [\textbf{test } \bot]Q$ because (1) there is no poststate ν such that $\omega[\![\textbf{test } \bot]\!]\nu$ and (2) $\omega[\![\textbf{test } \bot]\!] \lightning$ is false. Both of these follow immediately by definition (see page L6.5).

2 Safety of Output [55 points]

In this problem we consider a more complex safety policy than division by zero or array access out of bounds. We add three new commands to our language: one to open a stream for writing, one to print to a stream, and one to close a stream. To keep complexity manageable, we imagine there is just one stream (e.g., the terminal) that we can open, print to, and close:

$$\text{Programs} \quad \alpha, \beta \quad ::= \quad \ldots \mid \textbf{open} \mid \textbf{print } e \mid \textbf{close}$$

In this problem we assume this is the only source of unsafe behavior, that is, we exclude division and memory access from consideration.

As an example, here is a program that prints the numbers 0 to 9.

```
open ;
i := 0 ;
while (i < 10) {
  print i ;
  i := i + 1
} ;
close
```

We have the following safety conditions:

1. When a program starts to execute, we should assume the stream is closed.
2. When a program finishes, we should verify that the stream is closed.
3. Only when the stream is closed can we open it.
4. Only when the stream is open can we print to it.
5. Only when the stream is open can we close it.

In order to define the meaning of these programs, we introduce a new *ghost variable* "$status$" that is used to track the status of the stream. It is called a ghost variable because it is only introduced to reason about the safety of the program and may not appear directly in a source program. When $status = 0$ the stream is not open, and when $status \neq 0$ then the stream is open.

Task 2 (5 pts) The sample program above should be safe. State each of the following (by necessity using the ghost variable $status$):

(a) The precondition required for its safety (see item 1)

(b) The postcondition required for its safety (see item 2)

(c) The loop invariant needed to prove safety

(a) $status = 0$

(b) $status = 0$

(c) $status \neq 0$

In order to model the output of a program, the state now *should* contain a sequence of values that have been printed. However, since we are only interested in safety we are content to *approximate* the true meaning of a program and ignore the values actually printed.

Task 3 (15 pts) Give definitions for $\omega[\![\alpha]\!]\nu$ and $\omega[\![\alpha]\!]\lightning$ for each of the three new constructs.

Make sure to model the prescribed safety conditions accurately. In other words $\omega[\![\alpha]\!]\lightning$ should be true exactly when the program is unsafe, and no unsafe program should have a poststate.

$$
\begin{array}{lll}
\omega[\![\textbf{open}]\!]\nu & \text{iff} & \omega(status) = 0 \text{ and } \nu = \omega[status \mapsto 1] \\
\omega[\![\textbf{print } e]\!]\nu & \text{iff} & \omega(status) \neq 0 \text{ and } \nu = \omega \\
\omega[\![\textbf{close}]\!]\nu & \text{iff} & \omega(status) \neq 0 \text{ and } \nu = \omega[status \mapsto 0]
\end{array}
$$

$$
\begin{array}{lll}
\omega[\![\textbf{open}]\!]\lightning & \text{iff} & \omega(status) \neq 0 \\
\omega[\![\textbf{print } e]\!]\lightning & \text{iff} & \omega(status) = 0 \\
\omega[\![\textbf{close}]\!]\lightning & \text{iff} & \omega(status) = 0
\end{array}
$$

Task 4 (15 pts) Give *right* rules in the sequent calculus for **open**, **print**, and **close**.

You do not need to show left rules or axioms, and you do not need to prove their soundness.

Because the open, print, and close commands don't explicitly refer to the variable $status$, we cannot quite use the same solution we used for assignment and substitute a fresh $status'$ for $status$ to represent the *new* state of the variable. Since we still need to make sure that prior information about $status$ does not conflict with its new value we rename any reference to the *old* version of the variable which may be in Γ or Δ.

We abbreviate $status$ as s and write s' for a fresh variable to keep the width of the rules manageable.

$$\frac{\Gamma(s) \vdash s = 0, \Delta(s) \quad \Gamma(s'), s \neq 0 \vdash Q, \Delta(s')}{\Gamma(s) \vdash [\mathbf{open}]Q, \Delta(s)}\ [\mathbf{open}]R^{s'}$$

$$\frac{\Gamma \vdash s \neq 0, \Delta \quad \Gamma \vdash Q, \Delta}{\Gamma \vdash [\mathbf{print}\ e]Q, \Delta}\ [\mathbf{print}]R$$

$$\frac{\Gamma(s) \vdash s \neq 0, \Delta(s) \quad \Gamma(s'), s = 0 \vdash Q, \Delta(s')}{\Gamma(s) \vdash [\mathbf{close}]Q, \Delta(s)}\ [\mathbf{close}]R^{s'}$$

Task 5 (5 pts) Give a biconditional (iff) axiom for reasoning about **print** e of the form

$$[\mathbf{print}\ e]Q \leftrightarrow ??$$

$$[\mathbf{print}\ e]Q \leftrightarrow status \neq 0 \wedge Q$$

Task 6 (10 pts) Prove the validity of your axiom with respect to the semantics in Task 3.

$\omega \models [\mathbf{print}\ e]Q$	(assumption)
not $\omega[\![\mathbf{print}\ e]\!]\lightning$	(from assumption)
$\omega(status) \neq 0$	(by definition of unsafe semantics)
$\omega \models status \neq 0$	($\omega[\![status]\!] \neq \omega[\![0]\!]$)
$\omega[\![\mathbf{print}\ e]\!]\omega$	(by definition of safe semantics)
$\omega \models Q$	(by picking $\nu = \omega$ in first assumption)
$\omega \models status \neq 0 \wedge Q$	(to show; by combining prior results)

$\omega \models status \neq 0 \wedge Q$	(assumption)
$\omega[\![\mathbf{print}\ e]\!]\nu$ for some ν	(assumption)
$\nu = \omega$	(by definition of safe semantics)
$\nu \models Q$	(to show; by combining $\omega \models Q$ and $\nu = \omega$)
not $\omega[\![\mathbf{print}\ e]\!]\lightning$	(since $\omega \models status \neq 0$, so we know $\omega(status) \neq 0$)

Consider the following program schema (with $P(i)$ and $R(i)$ some formulas of pure arithmetic that may refer to i):

while $P(i)$ {
  if $R(i)$ **then close else open** ;

```
  print i ;
  if R(i) then open else close ;
  i := i + 1
}
```

Task 7 (5 pts) Assume suitable pre- and post-conditions to satisfy item 1 and item 2 in the informal definition of safety.

State conditions relating $P(i)$ and $R(i)$ that would guarantee safety of this program. They should be as general as possible (yes, we realize that $P(i) = \bot$ guarantees safety regardless of $R(i)$).

> For all i, $P(i)$ implies not $R(i)$, that is $P(i) \to \neg R(i)$ is valid.