Practice Midterm Exam

15-316 Software Foundations of Security & Privacy
Frank Pfenning

October 10, 2024

Name: Andrew ID:

Instructions

- This practice exam is closed-book, closed-notes.
- There are several appendices for reference.
- Reference pages will not be scanned (you may tear them off).
- Try to keep your answers inside the answer boxes to ensure proper scanning.
- You have 80 minutes to complete the practice exam.
- There are 5 problems.
- The maximal practice exam score is 200.

	Sequent Calculus	Dynamic Logic	Loop Invariants	Safety	Information Flow	
	Prob 1	Prob 2	Prob 3	Prob 4	Prob 5	Total
Score						
Max	30	50	35	45	40	200

1 Sequent Calculus (30 pts)

Either prove or refute the sequents in Tasks 1 and 2 by constructing a derivation where all leaves consist only of propositional variables. In case it is not valid, give at least one countermodel. For reference, the rules are provided in Appendix A.

Task 1 (10 pts).

$$\neg(p \to q) \vdash p \wedge \neg q$$

Task 2 (10 pts).

$$\neg q \vdash \neg(p \to q)$$

Task 3 (10 pts). Complete the following rule with one or more premises using only Γ, Δ, F and G such that the resulting rule is **sound** but **not invertible**. You do not need to justify your answer.

$$\frac{\qquad\qquad}{\Gamma \vdash F \to G, \Delta}$$

2 Dynamic Logic (50 points)

For the remainder of the exam, we fix our base language SAFETINY to following programs.

$$\begin{array}{rrcl} \text{programs} & \alpha, \beta & ::= & x := e \mid \alpha \,;\, \beta \mid \textbf{skip} \mid \textbf{if } P \textbf{ then } \alpha \textbf{ else } \beta \mid \textbf{while } P\ \alpha \\ & & \mid & \textbf{test } P \end{array}$$

See Appendix B and Appendix C for the semantic definitions and proof rules for SAFETINY. Imagine we want to add a conditional without an **else** clause. For example

$$\textbf{when } x < 0 \textbf{ do } x := -x$$

should set x to its absolute value.

Task 4 (5 pts). Complete the semantic definition for the new program construct.

$\omega[\![\textbf{when } P \textbf{ do } \alpha]\!]\nu$ iff

Task 5 (5 pts). Complete the following axiom for reasoning about the new construct.

$[\textbf{when } P \textbf{ do } \alpha]Q \quad \leftrightarrow$

Task 6 (20 pts). Prove that the right-to-left implication from your axiom is valid with respect to your semantics in Task 4. We have started the proof for you

Assume $\omega \models$

$\omega \models [\textbf{when } P \textbf{ do } \alpha]Q$ (to show)

Task 7 (10 pts). Complete the new case in the definition of the weakest liberal precondition.

$\mathsf{wlp}\ (\mathbf{when}\ P\ \mathbf{do}\ \alpha)\ Q =$

Task 8 (10 pts). Complete the rule for symbolic evaluation by providing one or more premises.

$$\frac{}{\Gamma \Vdash [\mathbf{when}\ P\ \mathbf{do}\ \alpha]\ S}\ [\mathbf{when}]R$$

3 Loop Invariants (35 points)

In the definition of evaluation eval $\omega\ \alpha = \nu$ we took ω and ν to be *partial maps* from variables to values. The def/use analysis then guaranteed all variables that may be used during evaluation have been defined before. This relies on def α consisting of all variables that *must be defined* by executing α. For other purposes, it is useful to calculate the set of variables that *may be defined* by a program, written maydef α.

Task 9 (10 pts). Complete the definition of the following function. You may use the usual set operation like union, intersection, etc.

maydef $(x := e)$ $=$

maydef $(\alpha\ ;\ \beta)$ $=$

maydef $(\mathbf{skip})$ $=$

maydef $(\mathbf{if}\ P\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta)$ $=$

maydef $(\mathbf{while}\ P\ \alpha)$ $=$

maydef $(\mathbf{test}\ P)$ $=$

Task 10 (10 pts). The maydef set is also relevant to information flow. Complete the following theorem (you don't need to prove it). Your theorem should be helpful in deriving consequences of information flow security, using the information about maydef α.

Theorem.

If $\Sigma \vdash \alpha$ secure and $x \notin$ maydef α

and $\Sigma \vdash \omega_1 \approx_\ell \omega_2$

and eval $\omega_1\ \alpha = \nu_1$ and eval $\omega_2\ \alpha = \nu_2$

and [ ]

then $\Sigma \vdash \nu_1 \approx_\ell \nu_2$

and [ ]

Task 11 (15 pts). The maydef and use sets can be used to infer or strengthen some loop invariants. Complete the following with conditions regarding the maydef and use set so that the rule is sound. Your conditions should **not** use $[\beta]R$ for any β or R. Remember that in SAFETINY all programs are safe.

$$\frac{\Gamma \vdash K, \Delta \qquad\qquad K \vdash Q}{\Gamma \vdash [\textbf{while}\ P\ \alpha]\, Q, \Delta}$$

4 Safety (45 points)

Consider an extension of our language by a single *stack*. We have operations **push** e and $x := \mathbf{pop}$, where **pop** is *unsafe* if the stack is empty. We consider the size of the stack to be unlimited. At the beginning of the program the stack is empty.

We introduce a *ghost variable size* that we intend to contain the current size of the stack at any point during execution of the program.

Task 12 (10 pts). Define a transformation of the program to *sandbox* all push and pop operations. The resulting program should abort instead of allowing an unsafe operation. We call this function sandbox. The ghost variable should appear in the resulting sandboxed code in case it contains push or pop commands. You may assume that, at the top level, the sandboxed program is prefixed by $size := 0$.

sandbox $(x := e)$	$=$	$x := e$
sandbox $(\alpha \,;\, \beta)$	$=$	(sandbox α) ; (sandbox β)
sandbox $(\mathbf{skip})$	$=$	$\mathbf{skip}$
sandbox $(\mathbf{if}\ P\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta)$	$=$	
sandbox $(\mathbf{while}\ P\ \alpha)$	$=$	
sandbox $(\mathbf{test}\ P)$	$=$	$\mathbf{test}\ P$
sandbox $(\mathbf{push}\ e)$	$=$	
sandbox $(x := \mathbf{pop})$	$=$	

We now add a new command to our language, **abort**, which always safely(!) terminates the program without a poststate.

Task 13 (5 pts). Complete the semantic definitions of **abort**.

$\omega[\![\mathbf{abort}]\!]\nu$ iff

$\omega[\![\mathbf{abort}]\!]\lightning$ iff

Task 14 (5 pts). Provide a definition of **abort** in the SAFETINY language.

$\mathbf{abort} \triangleq$

Task 15 (5 pts). Now imagine we had used **abort** as a primitive instead of **test**. Define **test** in terms of **abort** (and the remaining language constructs)

$\mathbf{test}\ P \quad \triangleq$

Task 16 (5 pts). Give an axiom for **abort**.

$[\mathbf{abort}]Q \quad \leftrightarrow$

Next we define **impossible** which is **always** unsafe.

Task 17 (5 pts). Complete the semantic definitions.

$\omega[\![\mathbf{impossible}]\!]\nu$ iff

$\omega[\![\mathbf{impossible}]\!]\lightning$ iff

Task 18 (5 pts). Recall the command **assert** P which is unsafe if P is false. Provide a definition of **assert** P using **impossible** and the remaining constructs in the SAFETINY language.

$\mathbf{assert}\ P \quad \triangleq$

Task 19 (5 pts). Complete the following axiom.

$[\mathbf{impossible}]Q \quad \leftrightarrow$

5 Information Flow (40 points)

In this problem we explore a version of *taint analysis*. It is intended to compute the lowest security level a variable can have that is equal to or higher than the initial level and consistent with all *direct flows* in the program. We define it as a function $\mathsf{taint}\ \Sigma\ \alpha = \Sigma'$ where Σ' may raise the security levels of some variables in Σ. Here are some examples:

$$\begin{array}{lcl} \mathsf{taint}\ (x : \mathsf{H}, y : \mathsf{L})\ (y := x + 1) & = & (x : \mathsf{H}, y : \mathsf{H}) \\ \mathsf{taint}\ (x : \mathsf{H}, y : \mathsf{L})\ (\textbf{if}\ x = 0\ \textbf{then}\ y := 1\ \textbf{else}\ y := 0) & = & (x : \mathsf{H}, y : \mathsf{L}) \end{array}$$

The level of y is not raised in the second example because the level of the program counter (pc) which is necessary for indirect flows is not modeled in this form of taint analysis.

We require that Σ is defined on all variables V occurring in the program.

$$\begin{array}{lcll} \Sigma \sqsubseteq \Sigma' & \text{iff} & \Sigma(x) \sqsubseteq \Sigma'(x) & \text{for every } x \in V \\ (\Sigma \sqcup \Sigma')(x) & = & \Sigma(x) \sqcup \Sigma(x') & \text{for } x \in V \end{array}$$

You may use conditions $\Sigma \vdash e : \ell$ and $\Sigma \vdash P : \ell$ because we can easily turns these into functions to compute ℓ given Σ and e or P.

Task 20 (20 pts). Fill in the missing cases in the following definition.

$$\begin{array}{lcl} \mathsf{taint}\ \Sigma\ (x := e) & = & \\ \\ \mathsf{taint}\ \Sigma\ (\alpha\ ;\ \beta) & = & \\ \\ \mathsf{taint}\ \Sigma\ (\textbf{skip}) & = & \\ \\ \mathsf{taint}\ \Sigma\ (\textbf{if}\ P\ \textbf{then}\ \alpha\ \textbf{else}\ \beta) & = & \\ \\ \mathsf{taint}\ \Sigma\ (\textbf{while}\ P\ \alpha) & = & \\ \\ \mathsf{taint}\ \Sigma\ (\textbf{test}\ P) & = & \Sigma \end{array}$$

Task 21 (10 pts). Use your definition from Task 20 to calculate:

$$\mathsf{taint}\ (x : \mathsf{H}, y : \mathsf{L}, z : \mathsf{L})\ (y := x + 1\ ;\ z := x - 3)$$

$=$

Task 22 (10 pts). Use the information flow type system to derive or fail to derive, where $\Sigma_0 = (x : \mathsf{H}, y : \mathsf{L}, z : \mathsf{L}, pc : \mathsf{L})$ the following. You may omit derivations for the levels of expressions and formulas.

$$\Sigma_0 \vdash \textbf{while}\ x > 0\ (y := y + 1\ ;\ x := x - 1)\ \mathsf{secure}$$

A Propositional Sequent Calculus

$$\frac{}{\Gamma, F \vdash F, \Delta}\ \mathsf{id}$$

$$\frac{\Gamma \vdash F, \Delta \quad \Gamma \vdash G, \Delta}{\Gamma \vdash F \wedge G, \Delta}\ \wedge R \qquad \frac{\Gamma, F, G \vdash \Delta}{\Gamma, F \wedge G \vdash \Delta}\ \wedge L$$

$$\frac{\Gamma, F \vdash G, \Delta}{\Gamma \vdash F \to G, \Delta}\ \to R \qquad \frac{\Gamma \vdash F, \Delta \quad \Gamma, G \vdash \Delta}{\Gamma, F \to G \vdash \Delta}\ \to L$$

$$\frac{\Gamma \vdash F, G, \Delta}{\Gamma \vdash F \vee G, \Delta}\ \vee R \qquad \frac{\Gamma, F \vdash \Delta \quad \Gamma, G \vdash \Delta}{\Gamma, F \vee G \vdash \Delta}\ \vee L$$

$$\frac{\Gamma, F \vdash \Delta}{\Gamma \vdash \neg F, \Delta}\ \neg R \qquad \frac{\Gamma \vdash F, \Delta}{\Gamma, \neg F \vdash \Delta}\ \neg L$$

B Dynamic Logic, Semantics

Expressions

$$\begin{array}{lcl} \omega[\![c]\!] & = & c \\ \omega[\![x]\!] & = & \omega(x) \\ \omega[\![e_1 + e_2]\!] & = & \omega[\![e_1]\!] + \omega[\![e_2]\!] \end{array}$$

Formulas

$$\begin{array}{lll} \omega \models e_1 \leq e_2 & \text{iff} & \omega[\![e_1]\!] \leq \omega[\![e_2]\!] \\ \omega \models e_1 = e_2 & \text{iff} & \omega[\![e_1]\!] = \omega[\![e_2]\!] \end{array}$$

$$\begin{array}{lll} \omega \models P \wedge Q & \text{iff} & \omega \models P \quad \text{and} \quad \omega \models Q \\ \omega \models P \vee Q & \text{iff} & \omega \models P \quad \text{or} \quad \omega \models Q \\ \omega \models P \rightarrow Q & \text{iff} & \omega \models P \quad \text{implies} \quad \omega \models Q \\ \omega \models \neg P & \text{iff} & \omega \not\models P \\ \omega \models P \leftrightarrow Q & \text{iff} & \omega \models P \quad \text{iff} \quad \omega \models Q \end{array}$$

$$\omega \models [\alpha]Q \qquad \text{iff} \quad \text{for every } \nu \text{ with } \omega[\![\alpha]\!]\nu \text{ we have } \nu \models Q$$

Programs

$$\begin{array}{lll} \omega[\![x := e]\!]\nu & \text{iff} & \omega[x \mapsto c] = \nu \text{ where } \omega[\![e]\!] = c \\ \omega[\![\alpha \,;\, \beta]\!]\nu & \text{iff} & \omega[\![\alpha]\!]\mu \text{ and } \mu[\![\beta]\!]\nu \text{ for some state } \mu \\ \omega[\![\textbf{skip}]\!]\nu & \text{iff} & \nu = \omega \\ \omega[\![\textbf{if } P \textbf{ then } \alpha \textbf{ else } \beta]\!]\nu & \text{iff} & \omega \models P \text{ and } \omega[\![\alpha]\!]\nu \quad \text{or} \\ & & \omega \not\models P \text{ and } \omega[\![\beta]\!]\nu \\ \omega[\![\textbf{while } P \ \alpha]\!]\nu & \text{iff} & \omega[\![\textbf{while } P \ \alpha]\!]^n\nu \quad \text{for some } n \in \mathbb{N} \\ \omega[\![\textbf{while } P \ \alpha]\!]^{n+1}\nu & \text{iff} & \omega \models P \text{ and } \omega[\![\alpha]\!]\mu \text{ and } \mu[\![\textbf{while } P \ \alpha]\!]^n\nu \\ \omega[\![\textbf{while } P \ \alpha]\!]^0\nu & \text{iff} & \omega \not\models P \text{ and } \omega = \nu \\ \omega[\![\textbf{test } P]\!]\nu & \text{iff} & \omega \models P \text{ and } \nu = \omega \end{array}$$

C Dynamic Logic, Proofs

Sequent Calculus (Right Rules Only)

$$\frac{\Gamma, x' = e \vdash Q(x'), \Delta}{\Gamma \vdash [x := e]Q(x), \Delta}\ [:=]R^{x'} \qquad \frac{\Gamma, P \vdash [\alpha]Q, \Delta \quad \Gamma, \neg P \vdash [\beta]Q, \Delta}{\Gamma \vdash [\textbf{if } P \textbf{ then } \alpha \textbf{ else } \beta]Q, \Delta}\ [\textbf{if}]R$$

$$\frac{\Gamma \vdash [\alpha]([\beta]Q), \Delta}{\Gamma \vdash [\alpha\ ;\ \beta]Q, \Delta}\ [;]R \qquad \frac{\Gamma \vdash Q, \Delta}{\Gamma \vdash [\textbf{skip}]Q, \Delta}\ [\textbf{skip}]R$$

$$\frac{\Gamma \vdash J, \Delta \quad J, P \vdash [\alpha]J \quad J, \neg P \vdash Q}{\Gamma \vdash [\textbf{while}_J\ P\ \alpha]Q, \Delta}\ [\textbf{while}]R \qquad \frac{\Gamma, P \vdash Q, \Delta}{\Gamma \vdash [\textbf{test}\ P]Q, \Delta}\ [\textbf{test}]R$$

Axioms

$$\begin{array}{rlcl}
[:=]A & [x := e]Q(x) & \leftrightarrow & \forall x'.x' = e \rightarrow Q(x') \quad (x' \text{ not in } e \text{ or } Q(x)) \\
[;]A & [\alpha\ ;\ \beta]Q & \leftrightarrow & [\alpha]([\beta]Q) \\
[\textbf{skip}]A & [\textbf{skip}]Q & \leftrightarrow & Q \\
[\textbf{if}]A & [\textbf{if } P \textbf{ then } \alpha \textbf{ else } \beta]Q & \leftrightarrow & (P \rightarrow [\alpha]Q) \wedge (\neg P \rightarrow [\beta]Q) \\
[\textbf{while}]A & [\textbf{while}_J\ P\ \alpha]Q & \leftrightarrow & J \\
 & & & \wedge \Box(J \wedge P \rightarrow [\alpha]J) \\
 & & & \wedge \Box(J \wedge \neg P \rightarrow Q) \\
[\textbf{test}]A & [\textbf{test}\ P]Q & \leftrightarrow & (P \rightarrow Q)
\end{array}$$

D Algorithms

Weakest Liberal Precondition $\mathsf{wlp}\ \alpha\ Q$

$$
\begin{array}{lcl}
\mathsf{wlp}\ (x := e)\ Q(x) & = & Q(e) \\
\mathsf{wlp}\ (\alpha\ ;\ \beta)\ Q & = & \mathsf{wlp}\ \alpha\ (\mathsf{wlp}\ \beta\ Q) \\
\mathsf{wlp}\ (\mathbf{skip})\ Q & = & Q \\
\mathsf{wlp}\ (\mathbf{if}\ P\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta)\ Q & = & (P \to \mathsf{wlp}\ \alpha\ Q) \wedge (\neg P \to \mathsf{wlp}\ \beta\ Q) \\
\mathsf{wlp}\ (\mathbf{while}_J\ P\ \alpha)\ Q & = & J \\
 & & \wedge\ \Box(J \wedge P \to \mathsf{wlp}\ \alpha\ J) \\
 & & \wedge\ \Box(J \wedge \neg P \to Q) \\
\mathsf{wlp}\ (\mathbf{test}\ P)\ Q & = & P \to Q
\end{array}
$$

Symbolic Evaluation $\Gamma \Vdash [\alpha]S$

$$\frac{\Gamma \vdash Q \quad Q \text{ pure}}{\Gamma \Vdash Q}\ \mathsf{arith} \qquad \frac{\Gamma \vdash \bot}{\Gamma \Vdash S}\ \mathsf{infeasible}$$

$$\frac{\Gamma, x' = e \Vdash S(x') \quad x' \text{ fresh}}{\Gamma \Vdash [x := e]S(x)}\ [:=]R^{x'}$$

$$\frac{\Gamma \Vdash [\alpha]([\beta]S)}{\Gamma \Vdash [\alpha\ ;\ \beta]S}\ [;]R \qquad \frac{\Gamma \Vdash S}{\Gamma \Vdash [\mathbf{skip}]S}\ [\mathbf{skip}]R$$

$$\frac{\Gamma, P \Vdash [\alpha]S \quad \Gamma, \neg P \Vdash [\beta]S}{\Gamma \Vdash [\mathbf{if}\ P\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta]S}\ [\mathbf{if}]R$$

$$\frac{\Gamma \vdash J \quad J, P \Vdash [\alpha]J \quad J, \neg P \Vdash S}{\Gamma \Vdash [\mathbf{while}_J\ P\ \alpha]S}\ [\mathbf{while}]R \qquad \frac{\Gamma, P \Vdash S}{\Gamma \Vdash [\mathbf{test}\ P]S}\ [\mathbf{test}]R$$

$$\frac{\Gamma, P \Vdash [\alpha]([\mathbf{while}^n\ P\ \alpha]S) \quad \Gamma, \neg P \Vdash S}{\Gamma \Vdash [\mathbf{while}^{n+1}\ P\ \alpha]S}\ \mathsf{unfold}^{n+1} \qquad \frac{\Gamma \Vdash S}{\Gamma \Vdash [\mathbf{while}^0\ P\ \alpha]S}\ \mathsf{unfold}^0$$

E Information Flow

Semantic Definition

Definition [Equivalence at Security Level ℓ]
$\Sigma \vdash \omega_1 \approx_\ell \omega_2$ iff $\omega_1(x) = \omega_2(x)$ for all x such that $\Sigma(x) \sqsubseteq \ell$.

Definition [Noninterference]
$\Sigma \models \alpha$ secure iff for all ω_1, ω_2, ν_1, ν_2, and ℓ
$\Sigma \vdash \omega_1 \approx_\ell \omega_2$, eval $\omega_1\ \alpha = \nu_1$, and eval $\omega_2\ \alpha = \nu_2$ implies $\Sigma \vdash \nu_1 \approx_\ell \nu_2$.

Information Flow Types

$\boxed{\Sigma \vdash e : \ell}$

$$\frac{\Sigma(x) = \ell}{\Sigma \vdash x : \ell}\ \mathsf{var}F \qquad \frac{}{\Sigma \vdash c : \bot}\ \mathsf{const}F \qquad \frac{\Sigma \vdash e_1 : \ell_1 \quad \Sigma \vdash e_2 : \ell_2 \quad \ell = \ell_1 \sqcup \ell_2}{\Sigma \vdash e_1 + e_2 : \ell}\ +F$$

$\boxed{\Sigma \vdash P : \ell}$

$$\frac{\Sigma \vdash e_1 : \ell_1 \quad \Sigma \vdash e_2 : \ell_2}{\Sigma \vdash e_1 \le e_2 : \ell_1 \sqcup \ell_2}\ \le F \qquad \frac{}{\Sigma \vdash \top : \bot}\ \top F \qquad \frac{\Sigma \vdash P : \ell_1 \quad \Sigma \vdash Q : \ell_2}{\Sigma \vdash P \wedge Q : \ell_1 \sqcup \ell_2}\ \wedge F$$

$\boxed{\Sigma \vdash \alpha\ \mathsf{secure}}$

$$\frac{\Sigma \vdash e : \ell \quad \ell' = \Sigma(pc) \sqcup \ell \quad \ell' \sqsubseteq \Sigma(x)}{\Sigma \vdash x := e\ \mathsf{secure}}\ :=F$$

$$\frac{\Sigma \vdash \alpha\ \mathsf{secure} \quad \Sigma \vdash \beta\ \mathsf{secure}}{\Sigma \vdash \alpha\ ;\ \beta\ \mathsf{secure}}\ ;F \qquad \frac{}{\Sigma \vdash \mathbf{skip}\ \mathsf{secure}}\ \mathbf{skip}F$$

$$\frac{\Sigma \vdash P : \ell \quad \ell' = \Sigma(pc) \sqcup \ell \quad \Sigma' = \Sigma[pc \mapsto \ell'] \quad \Sigma' \vdash \alpha\ \mathsf{secure} \quad \Sigma' \vdash \beta\ \mathsf{secure}}{\Sigma \vdash \mathbf{if}\ P\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta\ \mathsf{secure}}\ \mathbf{if}F$$

$$\frac{\Sigma \vdash P : \ell \quad \ell' = \Sigma(pc) \sqcup \ell \quad \Sigma' = \Sigma[pc \mapsto \ell'] \quad \Sigma' \vdash \alpha\ \mathsf{secure}}{\Sigma \vdash \mathbf{while}\ P\ \alpha\ \mathsf{secure}}\ \mathbf{while}F$$

$$\frac{}{\Sigma \vdash \mathbf{test}\ P\ \mathsf{secure}}\ \mathbf{test}F$$