Midterm Exam

15-316 Software Foundations of Security & Privacy
Frank Pfenning

October 10, 2024

Name: Sample Solution Andrew ID: **fp**

Instructions

- This 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 exam.
- There are 5 problems.
- The maximal exam score is 200.

	Sequent Calculus	Dynamic Logic	Loops & Invariants	Safety	Information Flow	
	Prob 1	Prob 2	Prob 3	Prob 4	Prob 5	Total
Score	**30**	**50**	**35**	**35**	**50**	**200**
Max	30	50	35	35	50	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 the sequent is not valid, give at least one countermodel. For reference, the rules are provided in Appendix A.

Task 1 (10 pts).

$$\dfrac{\dfrac{\dfrac{\dfrac{}{p \vdash q, p, r}\,\mathsf{id}}{\cdot \vdash p \to q, p, r}\,{\to}R \quad \dfrac{}{r \vdash p, r}\,\mathsf{id}}{(p \to q) \to r \vdash p, r}\,{\to}L}{(p \to q) \to r \vdash p \vee r}\,{\vee}R$$

Task 2 (10 pts).

$$\dfrac{\dfrac{\dfrac{}{p \vdash p, r}\,\mathsf{id} \quad \dfrac{\mathsf{XXX}}{p, q \vdash r}}{p, p \to q \vdash r}\,{\to}L \quad \dfrac{\dfrac{}{r \vdash p, r}\,\mathsf{id} \quad \dfrac{}{r, q \vdash r}\,\mathsf{id}}{r, p \to q \vdash r}\,{\to}L}{p \vee r, p \to q \vdash r}\,{\vee}L}{p \vee r \vdash (p \to q) \to r}\,{\to}R$$

Countermodel $p = q = \top$ and $r = \bot$.

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 **invertible** but **not sound**.

There are multiple solutions; here is one:

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

2 Dynamic Logic (50 points)

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

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

Consider an extension of SAFETINY with nondeterministic choice $\alpha \mid\mid \beta$. This program may arbitrarily execute either α or β, perhaps based on some hidden input such as a coin flip. If a nondeterministic choice is encountered multiple times (say, in a loop), the branch executed may be different each time. We define its semantics as

$$\omega[\![\alpha \mid\mid \beta]\!]\nu \quad \text{iff} \quad \omega[\![\alpha]\!]\nu \ \text{ or } \ \omega[\![\beta]\!]\nu$$

Also recall that in SAFETINY every construct is safe, so we define the meaning of $[\alpha]Q$ by

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

Appendix C and Appendix D summarize some other relevant definitions.

Task 4 (10 pts). Complete the following axiom for reasoning about nondeterministic choice.

$$[\alpha \mid\mid \beta]Q \quad \leftrightarrow \quad [\alpha]Q \wedge [\beta]Q$$

Task 5 (20 pts). Prove that the right-to-left implication of your axiom is valid.

$\omega \models [\alpha]Q \wedge [\beta]Q$	(1, assumption)
$\omega \models [\alpha]Q$ and $\omega \models [\beta]Q$	(2, from 1 by defn. of $\models$)
$\omega[\![\alpha \mid \beta]\!]\nu$	(3, assumption)
$\omega[\![\alpha]\!]\nu$ or $\omega[\![\beta]\!]\nu$	(4, from 3 by defn. of $[\![-]\!]$)
$\quad \omega[\![\alpha]\!]\nu$	(5, first case of 4)
$\quad \nu \models Q$	(6, from 5 and 2(a))
$\quad \omega[\![\beta]\!]\nu$	(7, second case of 4)
$\quad \nu \models Q$	(8, from 7 and 2(b))
$\nu \models Q$	(9, from 6 and 8 by cases on 4)
$\omega \models [\alpha \mid \beta]Q$	(9, from 3 and 9 by defn. of $\models$)

Task 6 (10 pts). Complete the new case in the definition of the weakest liberal precondition.

$$\mathsf{wlp}\ (\alpha \mid \beta)\ Q = \mathsf{wlp}\ \alpha\ Q \wedge \mathsf{wlp}\ \beta\ Q$$

Task 7 (10 pts). Complete the rule for symbolic evaluation by providing one or more premises.

$$\frac{\Gamma \Vdash [\alpha]\, S \quad \Gamma \Vdash [\beta]\, S}{\Gamma \Vdash [\alpha \mid \beta]\, S}\ [\mid]R$$

3 Loops and Invariants (35 points)

Consider adding a new form of loop to our language

$$\textbf{repeat } \alpha \textbf{ until } P$$

It should evaluate as follows:

1. Execute α
2. If P is true in the poststate we exit the loop
3. If P is false in the poststate we repeat the loop

The *loop invariant* J should be checked **just before the exit condition** P. For example,

$$sum := 0 \ ; i := 0 \ ; \textbf{repeat } i := i + 1 \ ; sum := sum + i \textbf{ until } i = n$$

should satisfy the postcondition $2 * sum = n * (n + 1)$. We should be able to prove this using the loop invariant $J = (2 * sum = i * (i + 1))$.

Task 8 (15 pts). Give a right rule for **repeat** in the sequent calculus using J as an invariant.

$$\frac{\Gamma \vdash [\alpha]J, \Delta \quad J, \neg P \vdash [\alpha]\, J \quad J, P \vdash Q}{\Gamma \vdash [\textbf{repeat } \alpha \textbf{ until } P]\, Q, \Delta}\ [\textbf{repeat}]R$$

Task 9 (10 pts). Show how to define **repeat** α **until** P using only the existing language constructs (including **while**).

$$\textbf{repeat } \alpha \textbf{ until } P \quad \triangleq \quad \alpha \ ; \textbf{while } \neg P \ \alpha$$

Task 10 (10 pts). Now imagine we had taken **repeat** loops as our primitive instead of **while** loops. Show how to define **while** loops using **repeat**, or briefly explain why you think this is not possible. Your definition may use all the language constructs (except **while**, of course).

$$\textbf{while } P\ \alpha \quad \triangleq \quad \textbf{if } \neg P \textbf{ then skip else repeat } \alpha \textbf{ until } \neg P$$

4 Safety (35 points)

In general, we analyze programs with uninitialized variables, which are considered inputs. However, before we actually run a program, it is important that every variable is defined by the time we use its value. The def/use analysis from Lab 1 ensures that this is always the case, but it is limited. For example,

$$\alpha_0 = (\textbf{if } \bot \textbf{ then } y := 0 \textbf{ else } x := 1)$$

definitely defines the variable x, but our def/use analysis will not determine that.

Task 11 (5 pts). Show the computation of $\mathsf{def}\ \alpha_0$. You may refer to the definition of this function in Appendix D.

$$\begin{aligned} & \mathsf{def}\ (\textbf{if } \bot \textbf{ then } y := 0 \textbf{ else } x := 1) \\ = \ & \mathsf{def}\ (y := 0) \cap \mathsf{def}\ (x := 1) \\ = \ & \{y\} \cap \{x\} \\ = \ & \{\,\} \end{aligned}$$

In order to improve on def/use analysis we define a new formula $\mathsf{defd}\ x$ which is true in a poststate if the variable x must have been assigned to (or it is already true) in the prestate. We then extend symbolic evaluation

$$P \Vdash [\alpha]Q$$

such that precondition P and postcondition Q may contain formulas of the form $\mathsf{defd}\ x$. P declares the variables we assume to be defined *before* α executes, and Q checks the variables we would like to be defined *after* α executes. For example, we should be able to derive

$$\mathsf{defd}\ z \Vdash [\textbf{if } \bot \textbf{ then } y := 0 \textbf{ else } x := 1]\,(\mathsf{defd}\ z \wedge \mathsf{defd}\ x)$$

Task 12 (10 pts). Complete the following extended rule for assignment so it also accounts for the defd predicate. You can find the original rules in Appendix D.

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

where x' does not occur in Γ, e, or $S(x')$.

The other rules remain unchanged, where loops are unrolled just once, that is, each while loop is annotated as $\textbf{while}^1$.

Task 13 (10 pts). Using your rule from Task 12 and the remaining rules for symbolic evaluation, derive the example above. You may assume any valid arithmetic sequents are proved by an oracle.

$$\dfrac{\dfrac{\dfrac{\vdots}{\mathsf{defd}\ z, \bot \vdash \bot}}{\mathsf{defd}\ z, \bot \Vdash [y := 0]\,(\mathsf{defd}\ z \wedge \mathsf{defd}\ x)}\ \mathsf{infeasible} \quad \dfrac{\dfrac{\dfrac{\vdots}{\mathsf{defd}\ z, \neg\bot, x' = 0, \mathsf{defd}\ x' \vdash \mathsf{defd}\ z \wedge \mathsf{defd}\ x'}}{\mathsf{defd}\ z, \neg\bot, x' = 0, \mathsf{defd}\ x' \Vdash \mathsf{defd}\ z \wedge \mathsf{defd}\ x'}\ \mathsf{arith}}{\mathsf{defd}\ z, \neg\bot \Vdash [x := 0]\,(\mathsf{defd}\ z \wedge \mathsf{defd}\ x)}\ [:=]R^{x'}}{\mathsf{defd}\ z \Vdash [\mathbf{if}\ \bot\ \mathbf{then}\ y := 0\ \mathbf{else}\ x := 1]\,(\mathsf{defd}\ z \wedge \mathsf{defd}\ x)}\ [\mathbf{if}]R$$

Task 14 (10 pts). Assuming an arithmetic oracle proves every valid sequent of pure arithmetic, is it still possible that a variable is definitely defined but symbolic evaluation cannot prove it? Either give such an example (you don't need to show how it fails) or briefly explain why you believe no such example exists.

$$(\mathbf{while}^1\ (x > 0)\ x := x - 1)\ ;\ \mathbf{if}\ x \leq 0\ \mathbf{then}\ y := 0\ \mathbf{else}\ \mathbf{skip}$$

This program definitely defines y, but symbolic evaluation will not uncover this because it can not assume the loop guard is false after one iteration.

5 Information Flow (50 points)

Consider the following program

$$\alpha_0 = (y := x - x)$$

with the security policy $\Sigma_0 = (x : \mathsf{H}, y : \mathsf{L})$ where $\mathsf{H} \sqsupset \mathsf{L}$.

Task 15 (15 pts). Complete the proof that this program is **semantically secure**, that is, $\Sigma_0 \models (y := x - x)$ secure.

$\Sigma_0 \vdash \omega_1 \approx_{\mathsf{L}} \omega_2$	(assumption)
$\mathsf{eval}\ \omega_1\ (y := x - x) = \nu_1$	(assumption)
$\mathsf{eval}\ \omega_2\ (y := x - x) = \nu_2$	(assumption)
$\nu_1 = \omega_1[y \mapsto 0]$	(by defn. of eval)
$\nu_2 = \omega_2[y \mapsto 0]$	(by defn. of eval)
$\nu_1(y) = \nu_2(y)$	(by prev. two lines)
$\nu_1(z) = \nu_2(z)$ for all $z \neq y$ and $\Sigma_0(z) = \mathsf{L}$	(since $\Sigma_0 \vdash \omega_1 \approx_{\mathsf{L}} \omega_2$)
$\Sigma_0 \vdash \nu_1 \approx_{\mathsf{L}} \nu_2$	(by previous two lines and $\Sigma_0(y) = \mathsf{L}$)

Task 16 (10 pts). Show that this program is **not well-typed** in Σ_0 according to our information flow type system by filling in this derivation to show it must fail. The rule for subtraction is the same as for addition in Appendix E.

$$\dfrac{\dfrac{\dfrac{}{\Sigma_0 \vdash x : \mathsf{H}}\,\mathsf{var} \quad \dfrac{}{\Sigma_0 \vdash x : \mathsf{H}}\,\mathsf{var}}{\Sigma_0 \vdash x - x : \mathsf{H}}\,{-F} \qquad \mathsf{H} = \Sigma_0(pc) \sqcup \mathsf{H} \quad \overset{\mathsf{XXX}}{\mathsf{H} \sqsubseteq \Sigma_0(y) = \mathsf{L}}}{\Sigma_0 \vdash y := x - x \text{ secure}}\,{:=F}$$

The security level of pc is irrelevant here, because the last premise of $:=F$ fails either way. It is okay to assume it is L.

Task 17 (10 pts). Give an encoding of information flow security for this example as a sequent in dynamic logic. Validity of the sequent should imply the program is secure, according to our policy.

$$y = y' \to [y = x - x \,;\, y' = x' - x']\,(y = y')$$

Task 18 (15 pts). Using the weakest liberal precondition, calculate a proposition whose validity implies that the program α_0 satisfies our security policy Σ_0. Is it valid?

$$\begin{array}{rl} & y = y' \to \mathsf{wlp}\ (y := x - x \,;\, y' := x' - x')\ (y = y') \\ = & y = y' \to \mathsf{wlp}\ (y := x - x)\ (\mathsf{wlp}\ (y' := x' - x')\ (y = y')) \\ = & y = y' \to \mathsf{wlp}\ (y := x - x)\ (y = x' - x') \\ = & y = y' \to x - x = x' - x' \end{array}$$

And, yes, this is valid.

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

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

$$\begin{array}{lll} \omega[\![\textbf{while } P \ \alpha]\!]^{n+1}\nu & \text{iff} & \omega \models P \text{ and } \omega[\![\alpha]\!]\mu \text{ and } \mu[\![\textbf{while } P \ \alpha]\!]^n\nu \\ \omega[\![\textbf{while } P \ \alpha]\!]^0\nu & \text{iff} & \omega \not\models P \text{ and } \omega = \nu \\ \omega[\![\textbf{test } P]\!]\nu & \text{iff} & \omega \models P \text{ and } \nu = \omega \end{array}$$

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

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

D Algorithms

Def/Use

The set $\mathsf{def}\ \alpha$ consists of all variables that α **must** define. We omit use since it is not needed in this exam.

$$
\begin{array}{lcl}
\mathsf{def}\ (x := e) & = & \{x\} \\
\mathsf{def}\ (\alpha\ ;\ \beta) & = & \mathsf{def}\ \alpha \cup \mathsf{def}\ \beta \\
\mathsf{def}\ (\mathsf{skip}) & = & \{\,\} \\
\mathsf{def}\ (\mathbf{if}\ P\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta) & = & \mathsf{def}\ \alpha \cap \mathsf{def}\ \beta \\
\mathsf{def}\ (\mathbf{while}\ P\ \alpha) & = & \{\,\} \\
\mathsf{def}\ (\mathbf{test}\ P) & = & \{\,\}
\end{array}
$$

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

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

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

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

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

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

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

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

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

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

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