Software Foundations of Security & Privacy

Digest Notes

Mohamed Waiel Shikfa

These notes are a condensed companion to the official course lecture notes. Each lecture is summarized as: a goal, one-page map, core definitions, key proof patterns/recipes, common pitfalls, and a short "check yourself" set. Full rule lists and semantic definitions are collected in the appendices and referenced from the relevant lecture.

Last updated: January 29, 2026

Lecture 2 (Jan 11, 2026): Propositional Logic and Proof

Goal: Use sequent calculus to (1) precisely define validity of formulas/sequents and (2) build/check formal derivations; understand soundness/completeness/decidability via invertibility + termination.

One-page map

- **Objects:** propositional formulas F built from variables and connectives; sequents $\Gamma \vdash \Delta$.
- **Semantics:** $\Gamma \vdash \Delta$ is valid iff "all Γ true implies some Δ true".
- **Proof system:** right rules decompose goals; left rules decompose assumptions; identity closes branches.
- **Meta-properties:** soundness (derivable $\Rightarrow$ valid), completeness (valid $\Rightarrow$ derivable), decidability (procedure to decide validity).

Core definitions

- **Definition.** *Formula*: built from p using $\wedge, \vee, \rightarrow, \neg, \top, \bot$.
- **Definition.** *Valid formula*: true under every truth assignment.
- **Definition.** *Satisfiable formula*: true under some truth assignment.
- **Definition.** *Sequent* $\Gamma \vdash \Delta$: assumptions Γ and goals Δ (multi-succedent, disjunctive).
- **Definition.** *Valid sequent*: if all formulas in Γ are true then at least one formula in Δ is true.
- **Definition.** *Derivation*: tree built bottom-up using inference rules; leaves typically close by identity.

Key rules / theorems (high level)

- **Key idea.** Right/Left rules eliminate a top-level connective from a goal/assumption.
- **Key idea.** Disjunction needs multi-succedent sequents for completeness.
- **Theorem (informal).** Soundness: if $\Gamma \vdash \Delta$ is derivable then it is valid.
- **Theorem (informal).** Completeness + decidability: all rules are invertible and reductive, so proof search terminates and succeeds iff sequent is valid.

Proof-search recipe (sequent calculus)

- **Step 1:** Apply invertible rules bottom-up whenever possible (decompose connectives).
- **Step 2:** Stop at variable-only leaves $p_1, \ldots, p_n \vdash q_1, \ldots, q_m$.
- **Step 3:** Leaf closes iff some $p_i = q_j$ (identity). Otherwise it is a countermodel seed.

Pitfalls

- **Common pitfall.** Confusing formula implication $F \to G$ with sequent entailment $F \vdash G$: they coincide in validity but are different syntactic objects.
- **Common pitfall.** Using a two-rule disjunction right-introduction ($\vee R_1 / \vee R_2$) breaks completeness for classical truth-table validity. In general, avoid applying more than one rule at a time.

Check yourself

1. **Check yourself.** Prove $p \vee (p \to q)$ is valid using sequent calculus.
2. **Check yourself.** Given a failed leaf $p, q \vdash r$, what truth assignment witnesses invalidity?

Reference: Full propositional sequent-calculus rules are in Appendix A.

Lecture 3 (Jan 13, 2026): Dynamic Logic

Goal: Extend sequent reasoning to talk about programs; express safety-style properties as postconditions; derive rules for program constructs (if/assignment/sequence/while).

One-page map

- **New syntax:** programs α (assign/seq/if/while) and modalities $[\alpha]Q$ (box) and $\langle\alpha\rangle Q$ (diamond).
- **Intuition:** $[\alpha]Q$ means "every terminating run of α ends in a state satisfying Q".
- **Theme:** Rules should reduce properties of compound programs to properties of smaller programs.

Core definitions

- **Definition.** *Trace*: (possibly infinite) sequence of states/events during execution.
- **Definition.** *Safety property*: "nothing bad happens" (prefix-closed).
- **Definition.** *Liveness property*: "something good eventually happens".
- **Definition.** *Box/diamond*: $[\alpha]Q$ (all terminating runs end in Q) and $\langle\alpha\rangle Q$ (some terminating run ends in Q). Since our language has no nondeterminism, $\langle\alpha\rangle Q$ is equivalent to "α terminates and $[\alpha]Q$".

Key rules (as "recipes")

- **Key idea. If:** split on the guard.
- **Key idea. Sequential Composition:** rewrite $[\alpha;\beta]Q$ as $[\alpha]([\beta]Q)$.
- **Key idea. Assignment:** do *not* assume $x = e$ with the same x on both sides of the state change; introduce fresh x' (post-state value) and substitute.
- **Key idea. While:** unfolding is sound but not reductive; loop invariants are the scalable rule.

Assignment gotcha (why freshness matters)

- **Common pitfall.** The tempting rule "assume $x = e$ after $x := e$" is *unsound* because it confuses the value of x *before* and *after* the assignment.
- **Lecture example:** with the unsound rule you could (wrongly) justify $x = 2 \vdash [x := 1]\, x = 3$ by reducing it to the premise $x = 2, x = 1 \vdash x = 3$, which is valid only because the antecedents are contradictory.
- **Key idea.** Fix (as in the notes): introduce a fresh post-state variable x' and replace the postcondition $Q(x)$ with $Q(x')$. Rule shape:
  $\Gamma \vdash [x := e]Q(x), \Delta$ reduces to $\Gamma, x' = e \vdash Q(x'), \Delta$ (with x' fresh).

Loop-invariant recipe

- **Pick an invariant J and prove these sequents (turnstile, not implication):**
  - **Init:** $\Gamma \vdash J$
  - **Preserved:** $J, P \vdash [\alpha]J$
  - **Post:** $J, \neg P \vdash Q$
- **Important:** Γ (facts true only initially) is intentionally dropped from the preserved/post obligations.

Check yourself

1. **Check yourself.** Explain why $[\alpha]Q$ is vacuously true when α does not terminate. What property is this capturing?
2. **Check yourself.** For the swap program (using + and -), what must the precondition include to avoid undefined behavior in bounded-integer languages?

Reference: Full dynamic-logic rule summary is in Appendix C.

Lecture 4 (Jan 20, 2026): Semantics of Programs

Goal: Pin down the meaning of expressions, programs, and formulas so rule soundness proofs become "expand definitions and chase the quantifiers".

One-page map

- **State:** $\omega(x)$ is the value of variable x.
- **Expression meaning:** $\omega[\![e]\!] \in \mathbb{Z}$.
- **Program meaning:** relation $\omega[\![\alpha]\!]\nu$ between pre/post states.
- **Formula meaning:** $\omega \models P$.
- **Reusable move:** prove equivalences (e.g. $[\alpha;\beta]Q \leftrightarrow [\alpha]([\beta]Q)$) and use them as rewrite rules.

Core clauses (keep only the high-yield ones)

- **Definition.** Assignment updates state: $\nu = \omega[x \mapsto \omega[\![e]\!]]$.
- **Definition.** Sequence composes via an intermediate state: $\exists\mu$.
- **Definition.** While termination is "there exists an n iterations" (bounded-iteration semantics).

Reusable proof pattern: "unwind definitions"

- **To show** $P \leftrightarrow Q$ is valid: fix an arbitrary state ω, assume $\omega \models P$, expand definitions until you can derive $\omega \models Q$ (and vice versa).
- **Key idea.** This justifies using valid equivalences as both left- and right-rules (sound + invertible).

Pitfalls

- **Common pitfall.** Quantification over program states is subtle: naive "substitute a constant for x" can fail when x is read across many loop states; semantics uses state update $\omega[x \mapsto c]$.

Check yourself

1. **Check yourself.** Prove (informally, by definitions) $\models [\alpha;\beta]Q \leftrightarrow [\alpha]([\beta]Q)$.
2. **Check yourself.** Give a simple program α and postcondition Q where $\omega \models [\alpha]Q$ holds because α does not terminate.

Reference: Full semantics (expressions, programs, formulas) are summarized in Appendix D.

Appendices

A Propositional sequent calculus (full rule list)

Sequents: $\Gamma \vdash \Delta$ where Γ, Δ are (multi)sets of formulas. **Validity:** all Γ true implies some Δ true.

Identity

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

Logical connectives

True	$\dfrac{}{\Gamma \vdash \top, \Delta}\ \top R$	$\dfrac{\Gamma \vdash \Delta}{\Gamma, \top \vdash \Delta}\ \top L$
False	$\dfrac{\Gamma \vdash \Delta}{\Gamma \vdash \bot, \Delta}\ \bot R$	$\dfrac{}{\Gamma, \bot \vdash \Delta}\ \bot L$
Negation	$\dfrac{\Gamma, F \vdash \Delta}{\Gamma \vdash \neg F, \Delta}\ \neg R$	$\dfrac{\Gamma \vdash F, \Delta}{\Gamma, \neg F \vdash \Delta}\ \neg L$
Conjunction	$\dfrac{\Gamma \vdash F, \Delta \quad \Gamma \vdash G, \Delta}{\Gamma \vdash F \wedge G, \Delta}\ \wedge R$	$\dfrac{\Gamma, F, G \vdash \Delta}{\Gamma, F \wedge G \vdash \Delta}\ \wedge L$
Disjunction	$\dfrac{\Gamma \vdash F, G, \Delta}{\Gamma \vdash F \vee G, \Delta}\ \vee R$	$\dfrac{\Gamma, F \vdash \Delta \quad \Gamma, G \vdash \Delta}{\Gamma, F \vee G \vdash \Delta}\ \vee L$
Implication	$\dfrac{\Gamma, F \vdash G, \Delta}{\Gamma \vdash F \to G, \Delta}\ {\to}R$	$\dfrac{\Gamma \vdash F, \Delta \quad \Gamma, G \vdash \Delta}{\Gamma, F \to G \vdash \Delta}\ {\to}L$
NOR (↓)	$\dfrac{\Gamma, F \vdash \Delta \quad \Gamma, G \vdash \Delta}{\Gamma \vdash F \downarrow G, \Delta}\ {\downarrow}R$	$\dfrac{\Gamma \vdash F, G, \Delta}{\Gamma, F \downarrow G \vdash \Delta}\ {\downarrow}L$

B Dynamic logic: Language summary

Variables	x, y, z		
Constants	c	$::=$	$\ldots, -1, 0, 1, \ldots$
Expressions	e	$::=$	$c \mid x \mid e_1 + e_2 \mid e_1 * e_2 \mid \ldots$
Programs	α, β	$::=$	$x := e \mid \alpha \,;\, \beta \mid \textbf{if } P \textbf{ then } \alpha \textbf{ else } \beta \mid \textbf{while } P\ \alpha$
Formulas	P, Q	$::=$	$e_1 \leq e_2 \mid e_1 = e_2 \mid \ldots$
		$\mid$	$P \wedge Q \mid P \vee Q \mid P \to Q \mid P \leftrightarrow Q \mid \neg P \mid \top \mid \bot$
		$\mid$	$[\alpha]Q \mid \langle\alpha\rangle Q$

C Dynamic logic: rule summary (box modality)

Reading: $[\alpha]Q$ is partial correctness (all *terminating* runs end in Q).

$$\frac{\Gamma, P \vdash [\alpha]Q, \Delta \quad \Gamma, \neg P \vdash [\beta]Q, \Delta}{\Gamma \vdash [\textbf{if } P \textbf{ then } \alpha \textbf{ else } \beta]Q, \Delta} \ [\textbf{if}]R$$

$$\frac{\Gamma, x' = e \vdash Q(x'), \Delta}{\Gamma \vdash [x := e]Q(x), \Delta} \ [:=]R^{x'}$$

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

$$\frac{\Gamma, P \vdash [\alpha]([\textbf{while } P \ \alpha]Q), \Delta \quad \Gamma, \neg P \vdash Q, \Delta}{\Gamma \vdash [\textbf{while } P \ \alpha]Q, \Delta} \ [\textbf{unfold}]R$$

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

D Semantics cheat sheet

States: total maps $\omega : \{x, y, \ldots\} \to \mathbb{Z}$.

Expressions

$$\omega[\![c]\!] = c \qquad \omega[\![x]\!] = \omega(x) \qquad \omega[\![e_1 + e_2]\!] = \omega[\![e_1]\!] + \omega[\![e_2]\!]$$

(and similarly for other operators).

State update

$$(\omega[x \mapsto c])(x) = c \qquad (\omega[x \mapsto c])(y) = \omega(y) \text{ for } y \neq x$$

Programs as relations Write $\omega[\![\alpha]\!]\nu$ for "executing α can take prestate ω to poststate ν".

$$\omega[\![x := e]\!]\nu \text{ iff } \nu = \omega[x \mapsto \omega[\![e]\!]]$$

$$\omega[\![\alpha \ ; \beta]\!]\nu \text{ iff } \exists \mu. \ \omega[\![\alpha]\!]\mu \wedge \mu[\![\beta]\!]\nu$$

$$\omega[\![\textbf{if } P \textbf{ then } \alpha \textbf{ else } \beta]\!]\nu \text{ iff } (\omega \models P \wedge \omega[\![\alpha]\!]\nu) \text{ or } (\omega \not\models P \wedge \omega[\![\beta]\!]\nu)$$

$$\omega[\![\textbf{while } P \ \alpha]\!]\nu \text{ iff } \omega[\![\textbf{while } P \ \alpha]\!]^n\nu \text{ for some } n \in \mathbb{N}$$

with the bounded-iteration clauses:

$$\omega[\![\textbf{while } P \ \alpha]\!]^0\nu \text{ iff } (\omega \not\models P \wedge \omega = \nu)$$

$$\omega[\![\textbf{while } P \ \alpha]\!]^{n+1}\nu \text{ iff } (\omega \models P \wedge \exists \mu. \ \omega[\![\alpha]\!]\mu \wedge \mu[\![\textbf{while } P \ \alpha]\!]^n\nu)$$

Formulas

$$\omega \models [\alpha]Q \text{ iff } \forall \nu. \ (\omega[\![\alpha]\!]\nu \Rightarrow \nu \models Q) \qquad \omega \models \langle\alpha\rangle Q \text{ iff } \exists \nu. \ (\omega[\![\alpha]\!]\nu \wedge \nu \models Q)$$