

EC4219: Software Engineering

Lecture 1 — Propositional Logic (1) *Syntax and Semantics*

Sunbeam So
2025 Spring

Why Computational Logic in Software Engineering?

```
1 void testme(int x, int y) {  
2     z = 2 * y;  
3     if (z == x) {  
4         if (x > y+10) {  
5             /* Vulnerability */  
6         }  
7     }  
8 }
```

Goal:
Find the bug

Why Computational Logic in Software Engineering?

```
1 void testme(int x, int y) {  
2     z = 2 * y;  
3     if (z == x) {    x = 2*y  
4         if (x > y+10) { y > 10  
5             /* Vulnerability */  
6         }  
7     }  
8 }
```

Goal:
Find the bug

Why Computational Logic in Software Engineering?

```
1 void testme(int x, int y) {      • Test-cases for reaching line 5:  
2     z = 2 * y;  
3     if (z == x) {    x = 2*y  
4         if (x > y+10) {    y > 10  
5             /* Vulnerability */  
6         }  
7     }  
8 }
```

- (x,y) : (22, 11), (24, 12), ..., (100, 50)



Goal:
Find the bug

Why Computational Logic in Software Engineering?

```
1 void testme(int x, int y) {  
2     z = 2 * y;  
3     if (z == x) {    x = 2*y  
4         if (x > y+10) {  y > 10  
5             /* Vulnerability */  
6         }  
7     }  
8 }
```

- Test-cases for reaching line 5:
 - (x,y) : (22, 11), (24, 12), ..., (100, 50)
- Success probability of random testing (assumption: $1 \leq x, y \leq 100$)
 - **0.4% (= 40/10000)**

Goal:
Find the bug

Can we do better in systematic ways?

Why Computational Logic in Software Engineering?

```
1 void testme(int x, int y) {  
2     z = 2 * y;           α      β  
3     if (z == x) {  
4         if (x > y+10) {  
5             /* Vulnerability */  
6         }  
7     }  
8 }
```

$$(x = \alpha) \wedge (y = \beta) \wedge (z = 2 * y) \wedge (z = x) \wedge (x > y + 10)$$

Goal:
Find the bug

Why Computational Logic in Software Engineering?

```
1 void testme(int x, int y) {  
2     z = 2 * y;           α    β  
3     if (z == x) {  
4         if (x > y+10) {  
5             /* Vulnerability */  
6         }  
7     }  
8 }
```

Goal:
Find the bug

$$(x = \alpha) \wedge (y = \beta) \wedge (z = 2 * y) \wedge (z = x) \wedge (x > y + 10)$$

SMT Solver (constraint solver)

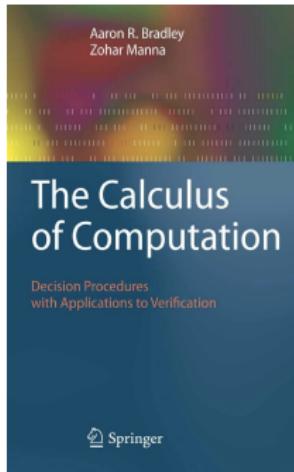
Error-triggering input

$$x \mapsto 22, y \mapsto 11, z \mapsto 22, \boxed{\alpha \mapsto 22, \beta \mapsto 11}$$

- Logic is the mathematical basis for systematically analyzing software.
- Many software engineering tools are built on top of logic.
 - Symbolic execution, formal verification, program synthesis, etc.

Reference Book

- “The Calculus of Computation” by Aaron R. Bradley and Zohar Manna
- See chapters 1, 2, and 5 for this course.
 - ▶ Chapter 1: Propositional Logic
 - ▶ Chapter 2: First-order Logic
 - ▶ Chapter 5: Program Verification



Preliminary 1: Inference Rule

One way to define the set is to use inference rules. An inference rule is of the form:

$$\frac{A}{B}$$

- A : hypothesis (antecedent)
- B : conclusion (consequent)
- Interpreted as: “if A is true then B is also true”.
- Inference rules without hypotheses are called axioms (e.g., B):

$$\overline{B}$$

- The hypothesis may contain multiple statements, e.g.,

$$\frac{A \quad B}{C}$$

Interpreted as: “If both A and B are true then so is C ”.

Preliminary 2: Production Rule

Another way to define the set is to use production rules.

- The set of natural numbers $\mathbb{N} = \{0, 1, 2, \dots\}$ is inductively defined using inference rules:

$$\frac{}{0 \in \mathbb{N}} \quad \frac{n \in \mathbb{N}}{n + 1 \in \mathbb{N}}$$

For notational brevity, sometimes we just write: $\overline{0} \quad \frac{n}{n + 1}$

- The inference rules can be expressed by production rules:

$$n \rightarrow 0 \mid n + 1$$

Interpreted as:

- (1) 0 is a natural number.
- (2) If n is a natural number then so is $n + 1$.

Syntax of Propositional Logic

- **Atom:** basic elements
 - ▶ truth symbols: \perp (*false*), \top (*true*)
 - ▶ propositional variables P, Q, R, \dots
- **Literal:** an atom α or its negation $\neg\alpha$
- **Formula:** a literal, or the application of a logical(boolean) connective to formulas

F	\rightarrow	\perp
		\top
		P
		$\neg F$
		negation (“not”)
		$F_1 \wedge F_2$
		conjunction (“and”)
		$F_1 \vee F_2$
		disjunction (“or”)
		$F_1 \rightarrow F_2$
		implication (“implies”)
		$F_1 \leftrightarrow F_2$
		iff (“if and only if”)

Subformula

- Formula G is a **subformula** of formula F if it syntactically occurs within G .

$$\mathbf{sub}(\perp) = \{\perp\}$$

$$\mathbf{sub}(\top) = \{\top\}$$

$$\mathbf{sub}(P) = \{P\}$$

$$\mathbf{sub}(\neg F) = \{\neg F\} \cup \mathbf{sub}(F)$$

$$\mathbf{sub}(F_1 \wedge F_2) = \{F_1 \wedge F_2\} \cup \mathbf{sub}(F_1) \cup \mathbf{sub}(F_2)$$

$$\mathbf{sub}(F_1 \vee F_2) = \{F_1 \vee F_2\} \cup \mathbf{sub}(F_1) \cup \mathbf{sub}(F_2)$$

$$\mathbf{sub}(F_1 \rightarrow F_2) = \{F_1 \rightarrow F_2\} \cup \mathbf{sub}(F_1) \cup \mathbf{sub}(F_2)$$

$$\mathbf{sub}(F_1 \leftrightarrow F_2) = \{F_1 \leftrightarrow F_2\} \cup \mathbf{sub}(F_1) \cup \mathbf{sub}(F_2)$$

- The strict subformulas of a formula are all its subformulas except itself.

$$\mathbf{strict}(F) = \mathbf{sub}(F) \setminus \{F\}$$

Example: Subformula

Find subformulas of the formula

$$F : (P \wedge Q) \rightarrow (P \vee \neg Q).$$

$$\begin{aligned}\mathbf{sub}(F) &= \{(P \wedge Q) \rightarrow (P \vee \neg Q)\} \cup \mathbf{sub}(P \wedge Q) \cup \mathbf{sub}(P \vee \neg Q) \\ &= \{(P \wedge Q) \rightarrow (P \vee \neg Q)\} \cup \\ &\quad \cup \{P \wedge Q\} \cup \mathbf{sub}(P) \cup \mathbf{sub}(Q) \\ &\quad \cup \{P \vee \neg Q\} \cup \mathbf{sub}(P) \cup \mathbf{sub}(\neg Q) \\ &= \{(P \wedge Q) \rightarrow (P \vee \neg Q)\} \\ &\quad \cup \{P \wedge Q\} \cup \{P\} \cup \{Q\} \\ &\quad \cup \{P \vee \neg Q\} \cup \{P\} \cup \{\neg Q\} \cup \mathbf{sub}(Q) \\ &= \{(P \wedge Q) \rightarrow (P \vee \neg Q)\} \\ &\quad \cup \{P \wedge Q\} \cup \{P\} \cup \{Q\} \\ &\quad \cup \{P \vee \neg Q\} \cup \{P\} \cup \{\neg Q\} \cup \{Q\} \\ &= \{(P \wedge Q) \rightarrow (P \vee \neg Q), P \wedge Q, P \vee \neg Q, \neg Q, P, Q\}\end{aligned}$$

Semantics

- The semantics of a logic provides its meaning. The meaning of a PL formula is either *true* or *false*.
- The semantics of a formula is defined with an **interpretation** that assigns truth values to propositional variables.
- Semantics is inductively defined, where we write $I \models F$ (resp., $I \not\models F$) iff F evaluates to *true* (resp., *false*).

$$I \models \top$$

$$I \not\models \perp$$

$$I \models P \quad \text{iff} \quad I[P] = \text{true}$$

$$I \not\models P \quad \text{iff} \quad I[P] = \text{false}$$

$$I \models \neg F \quad \text{iff} \quad I \not\models F$$

$$I \models F_1 \wedge F_2 \quad \text{iff} \quad I \models F_1 \text{ and } I \models F_2$$

$$I \models F_1 \vee F_2 \quad \text{iff} \quad I \models F_1 \text{ or } I \models F_2$$

$$I \models F_1 \rightarrow F_2 \quad \text{iff} \quad I \not\models F_1 \text{ or } I \models F_2$$

$$I \models F_1 \leftrightarrow F_2 \quad \text{iff} \quad (I \models F_1 \text{ and } I \models F_2) \text{ or } (I \not\models F_1 \text{ and } I \not\models F_2)$$

Example: Semantics

Consider the formula

$$F : P \wedge Q \rightarrow P \vee \neg Q$$

and the interpretation

$$I : \{P \mapsto \text{true}, Q \mapsto \text{false}\}.$$

The truth value of F is computed as follows.

1. $I \models P$ since $I[P] = \text{true}$
2. $I \not\models Q$ since $I[Q] = \text{false}$
3. $I \models \neg Q$ by 2 and semantics of \neg
4. $I \not\models P \wedge Q$ by 2 and semantics of \wedge
5. $I \models F$ by 4 and semantics of \rightarrow

Satisfiability and Validity

- A formula F is **satisfiable** iff there exists an interpretation (sometimes called assignment) I such that $I \models F$.
- A formula F is **valid** iff for all interpretations I , $I \models F$.
- Satisfiability and validity are dual.¹

F is valid iff $\neg F$ is unsatisfiable

- We can check satisfiability by deciding validity, and vice versa.

¹In logic, functions (or relations) A and B are dual if $A(x) = \neg B(\neg x)$.

Determining Validity and Satisfiability

There are two approaches to show F is valid.

- **Truth table method:** performs exhaustive search.
 - ▶ Ex) $F : P \wedge Q \rightarrow Q \vee \neg Q$.

P	Q	$P \wedge Q$	$\neg Q$	$P \vee \neg Q$	F
0	0	0	1	1	1
0	1	0	0	0	1
1	0	0	1	1	1
1	1	1	0	0	1

- **Semantic argument method:** uses deduction (proof by contradiction).
 - ▶ Assume F is invalid: $I \not\models F$ for some I (falsifying interpretation).
 - ▶ Apply deduction rules (proof rules) to derive a contradiction.
 - ★ If every branch of the proof derives a contradiction, then F is valid.
 - ★ If some branch of the proof never derives a contradiction, then F is invalid. This branch describes a falsifying interpretation of F .
- Modern SAT solvers combine search and deduction.

Deduction Rules for Propositional Logic

Proof rules used in the semantic argument method:

$$\frac{I \models \neg F}{I \not\models F} \quad \frac{I \not\models \neg F}{I \models F}$$

$$\frac{I \models F \wedge G}{I \models F, I \models G} \quad \frac{I \not\models F \wedge G}{I \not\models F \mid I \not\models G}$$

$$\frac{I \models F \vee G}{I \models F \mid I \models G} \quad \frac{I \not\models F \vee G}{I \not\models F, I \not\models G}$$

$$\frac{I \models F \rightarrow G}{I \not\models F \mid I \models G} \quad \frac{I \not\models F \rightarrow G}{I \models F, I \not\models G}$$

$$\frac{I \models F \leftrightarrow G}{I \models F \wedge G \mid I \models \neg F \wedge \neg G} \quad \frac{I \not\models F \leftrightarrow G}{I \models F \wedge \neg G \mid I \models \neg F \wedge G}$$
$$\frac{\begin{array}{c} I \models F \\ I \not\models F \end{array}}{I \models \perp}$$

Example 1: Semantic Argument Method

Prove that the following formula is valid.

$$F : P \wedge Q \rightarrow P \vee \neg Q$$

1. $I \not\models P \wedge Q \rightarrow P \vee \neg Q$ assumption
2. $I \models P \wedge Q$ by 1 and the rule of \rightarrow
3. $I \not\models P \vee \neg Q$ by 1 and the rule of \rightarrow
4. $I \models P$ by 2 and the rule of \wedge
5. $I \not\models P$ by 3 and the rule of \vee
6. $I \models \perp$ 4 and 5 are contradictory

Example 2: Semantic Argument Method

Prove that the following formula is valid.

$$F : (P \rightarrow Q) \wedge (Q \rightarrow R) \rightarrow (P \rightarrow R)$$

Example 2: Semantic Argument Method

Prove that the following formula is valid.

$$F : (P \rightarrow Q) \wedge (Q \rightarrow R) \rightarrow (P \rightarrow R)$$

1. $I \not\models F$ assumption
2. $I \models (P \rightarrow Q) \wedge (Q \rightarrow R)$ by 1 and the rule of \wedge
3. $I \not\models P \rightarrow R$ by 1 and the rule of \rightarrow
4. $I \models P$ by 3 and the rule of \rightarrow
5. $I \not\models R$ by 3 and the rule of \rightarrow
6. $I \models P \rightarrow Q$ by 2 and the rule of \wedge
7. $I \models Q \rightarrow R$ by 2 and the rule of \wedge

We have two cases from 6 and the rule of \rightarrow .

- ① 8. $I \not\models P$: contradiction with 4
- ② 9. $I \models Q$: At this point, consider other two cases from 7 and the rule of \rightarrow .
 - ① $I \not\models Q$: contradiction with 9
 - ② $I \models R$: contradiction with 5

Example 3: Semantic Argument Method

Determine the validity of the following formula.

$$F : (P \vee Q) \rightarrow (P \wedge Q)$$

Example 3: Semantic Argument Method

Determine the validity of the following formula.

$$F : (P \vee Q) \rightarrow (P \wedge Q)$$

1. $I \not\models (P \vee Q) \rightarrow (P \wedge Q)$ assumption
2. $I \models P \vee Q$ by 1 and the rule of \rightarrow
3. $I \not\models P \wedge Q$ by 1 and the rule of \rightarrow

We have two cases from 2 and the rule of \vee .

- ① 4. $I \models P$: At this point, consider other two cases from 3 and the rule of \wedge
 - ① $I \not\models P$: contradiction with 4
 - ② $I \not\models Q$: no more proof rules are applicable, and no contradictions are derived. Thus F is invalid. The falsifying interpretation is
 $I : \{P \mapsto \top, Q \mapsto \perp\}$.

Derived Rules

The proof rules are sufficient, but **derived rules** can make proofs more concise (e.g., proofs can be made without branches). Examples include the rule of **modus ponens**:

$$\frac{I \models F \quad I \models F \rightarrow G}{I \models G}$$

Revisit the formula $F : (P \rightarrow Q) \wedge (Q \rightarrow R) \rightarrow (P \rightarrow R)$ to show its validity using the above rule.

- | | | |
|-----|--|------------------------------------|
| 1. | $I \not\models F$ | assumption |
| 2. | $I \models (P \rightarrow Q) \wedge (Q \rightarrow R)$ | by 1 and the rule of \rightarrow |
| 3. | $I \not\models P \rightarrow R$ | by 1 and the rule of \rightarrow |
| 4. | $I \models P$ | by 3 and the rule of \rightarrow |
| 5. | $I \not\models R$ | by 3 and the rule of \rightarrow |
| 6. | $I \models P \rightarrow Q$ | by 2 and the rule of \wedge |
| 7. | $I \models Q \rightarrow R$ | by 2 and the rule of \wedge |
| 8. | $I \models Q$ | by 4, 6, and modus ponens |
| 9. | $I \models R$ | by 7, 8, and modus ponens |
| 10. | $I \models \perp$ | 5 and 9 are contradictory |

Equivalence and Implication

- Two formulas F_1 and F_2 are equivalent:

$$F_1 \iff F_2$$

iff $F_1 \leftrightarrow F_2$ is valid, i.e., for all interpretations I , $I \models F_1 \leftrightarrow F_2$.

- Formula F_1 implies F_2

$$F_1 \implies F_2$$

iff $F_1 \rightarrow F_2$ is valid, i.e., for all interpretations I , $I \models F_1 \rightarrow F_2$.

- We can check equivalence and implication by checking validity.

cf) $F_1 \iff F_2$ and $F_1 \implies F_2$ are not formulas. They are semantic assertions!

Example: Equivalence

Prove that

$$P \iff \neg\neg P$$

Proof) Assume $I \not\models P \leftrightarrow \neg\neg P$ for some I . We have two cases from the rule of \leftrightarrow .

Case 1) $I \models P \wedge \neg\neg\neg P$

- $I \models P$
- $I \models \neg\neg\neg P \rightsquigarrow I \not\models \neg\neg P \rightsquigarrow I \models \neg P \rightsquigarrow I \not\models P$ (contradiction!)

Case 2) $I \models \neg P \wedge \neg\neg P$

- $I \models \neg P \rightsquigarrow I \not\models P$
- $I \models \neg\neg P \rightsquigarrow I \not\models \neg P \rightsquigarrow I \models P$ (contradiction!)

Substitution

- A substitution σ is a mapping from formulas to formulas:

$$\sigma : \{F_1 \mapsto G_1, \dots, F_n \mapsto G_n\}$$

- The domain of σ , denoted **dom**, is

$$\text{dom}(\sigma) : \{F_1, \dots, F_n\}$$

while the range, denoted **range**, is

$$\text{range}(\sigma) : \{G_1, \dots, G_n\}$$

- The application of a substitution σ to a formula F , $F\sigma$, replaces each occurrence of F_i with G_i . Replacements occur all at once.
- When two subformulas F_j and F_k are in $\text{dom}(\sigma)$ and F_k is a strict subformula of F_j , then F_j is replaced first.

Example: Substitution

Consider the formula F

$$F : P \wedge Q \rightarrow P \vee \neg Q$$

and the substitution σ

$$\sigma : \{P \mapsto R, P \wedge Q \mapsto P \rightarrow Q\}$$

Then,

$$F\sigma : (P \rightarrow Q) \rightarrow R \vee \neg Q$$

More Notations on Substitution

- A variable substitution is a substitution whose domain consists only of propositional variables.
- When we write $F[F_1, \dots, F_n]$, we mean that formula F can have formulas F_1, \dots, F_n as subformulas.
- If $\sigma = \{F_1 \mapsto G_1, \dots, F_n \mapsto G_n\}$, then

$$F[F_1, \dots, F_n]\sigma : F[G_1, \dots, G_n]$$

- For example, in the previous example, writing

$$F[P, P \wedge Q]\sigma : F[R, P \rightarrow Q]$$

emphasizes that P and $P \wedge Q$ are replaced by R and $P \rightarrow Q$, respectively.

Semantic Consequences of Substitution

Proposition (Substitution of Equivalent Formulas)

Consider a substitution $\sigma : \{F_1 \mapsto G_1, \dots, F_n \mapsto G_n\}$ such that $F_i \iff G_i$ for each i . Then, $F \iff F\sigma$.

Proposition (Valid Template)

If F is valid and $G = F\sigma$ for some variable substitution σ , then G is valid.

- For example, since

$$F : (P \rightarrow Q) \leftrightarrow (\neg P \vee Q)$$

is valid, every formula of the form $F_1 \rightarrow F_2$ is equivalent to $\neg F_1 \vee F_2$, for arbitrary formulas F_1 and F_2 .

- Proving the validity of F actually proves the validity of an infinite set of formulas that can be derived from F by variable substitutions.

Composition of Substitutions

Given substitutions σ_1 and σ_2 , their composition $\sigma = \sigma_1\sigma_2$ (“apply σ_1 and then σ_2 ”) is computed as follows.

- ① Apply σ_2 to each formula of $\text{range}(\sigma_1)$, and add the results to σ .
- ② If F_i of $F_i \mapsto G_i$ appears in $\text{dom}(\sigma_2)$ but not in $\text{dom}(\sigma_1)$, add $F_i \mapsto G_i$ to σ .

For example,

$$\begin{aligned}\sigma_1\sigma_2 &: \{P \mapsto R, P \wedge Q \mapsto P \rightarrow Q\} \{P \mapsto S, S \mapsto Q\} \\&= \{P \mapsto R\sigma_2, P \wedge Q \mapsto (P \rightarrow Q)\sigma_2, S \mapsto Q\} \\&= \{P \mapsto R, P \wedge Q \mapsto S \rightarrow Q, S \mapsto Q\}\end{aligned}$$

Summary

- Q. Why computational logic in software engineering?
 - A. Mathematical basis for systematically analyzing software.
- Syntax and semantics of propositional logic
- Satisfiability and validity
- Basic notations (inference rule, context-free grammar, substitution)

Announcement: No class on March 26 (this Wednesday)

Next lecture (March 31): normal forms and decision procedures