EC4219: Software Engineering

Lecture 1 — Propositional Logic (1)
Syntax and Semantics

Sunbeom So
2025 Spring

Why Computational Logic in Software Engineering?

```
void testme(int x, int y) {
  z = 2 * y;
  if (z == x) {
    if (x > y+10) {
      /* Vulnerability */
    }
  }
}
```

Goal:
Find the bug

Why Computational Logic in Software Engineering?

```c
void testme(int x, int y) {
  z = 2 * y;
  if (z == x) {
    if (x > y+10) {
      /* Vulnerability */
    }
  }
}
```

x = 2*y

y > 10

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

- Test-cases for reaching line 5:
  - **(x,y)**: (22, 11), (24, 12), …, (100, 50)

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 }
```

x = 2*y

y > 10

Goal: Find the bug

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

Can we do better in systematic ways?

Why Computational Logic in Software Engineering?

```
void testme(int x, int y) {
  z = 2 * y;
  if (z == x) {
    if (x > y+10) {
      /* Vulnerability */
    }
  }
}
```

α

β

Goal:
Find the bug

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

Why Computational Logic in Software Engineering?

```
void testme(int x, int y) {
  z = 2 * y;
  if (z == x) {
    if (x > y+10) {
      /* Vulnerability */
    }
  }
}
```

α β

Goal: Find the bug

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

↓

SMT Solver (constraint solver)

↓

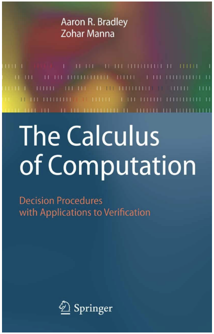
Error-triggering input

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

- Logic is the mathematical basis for systemically 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:

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

- $\boldsymbol{A}$: hypothesis (antecedent)
- $\boldsymbol{B}$: conclusion (consequent)
- Interpreted as: "if A is true then B is also true".
- Inference rules without hypotheses are called axioms (e.g., $\boldsymbol{B}$):

$$\frac{}{\boldsymbol{B}}$$

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

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

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 $\mathbb{N} = \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \cdots\}$ is inductively defined using inference rules:

$$\frac{}{\mathbf{0} \in \mathbb{N}} \qquad \frac{\boldsymbol{n} \in \mathbb{N}}{\boldsymbol{n} + \mathbf{1} \in \mathbb{N}}$$

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

- The inference rules can be expressed by production rules:

$$\boldsymbol{n} \to \mathbf{0} \mid \boldsymbol{n} + \mathbf{1}$$

  Interpreted as:
  (1) $\mathbf{0}$ is a natural number.
  (2) If $\boldsymbol{n}$ is a natural number then so is $\boldsymbol{n} + \mathbf{1}$.

Syntax of Propositional Logic

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

$$
\begin{array}{rcll}
F & \rightarrow & \bot & \\
& | & \top & \\
& | & P & \\
& | & \neg F & \text{negation ("not")} \\
& | & F_1 \land F_2 & \text{conjunction ("and")} \\
& | & F_1 \lor F_2 & \text{disjunction ("or")} \\
& | & F_1 \rightarrow F_2 & \text{implication ("implies")} \\
& | & F_1 \leftrightarrow F_2 & \text{iff ("if and only if")}
\end{array}
$$

Subformula

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

$$
\begin{array}{rcl}
\mathbf{sub}(\bot) & = & \{\bot\} \\
\mathbf{sub}(\top) & = & \{\top\} \\
\mathbf{sub}(P) & = & \{P\} \\
\mathbf{sub}(\neg F) & = & \{\neg F\} \cup \mathbf{sub}(F) \\
\mathbf{sub}(F_1 \wedge F_2) & = & \{F_1 \wedge F_2\} \cup \mathbf{sub}(F_1) \cup \mathbf{sub}(F_2) \\
\mathbf{sub}(F_1 \vee F_2) & = & \{F_1 \vee F_2\} \cup \mathbf{sub}(F_1) \cup \mathbf{sub}(F_2) \\
\mathbf{sub}(F_1 \rightarrow F_2) & = & \{F_1 \rightarrow F_2\} \cup \mathbf{sub}(F_1) \cup \mathbf{sub}(F_2) \\
\mathbf{sub}(F_1 \leftrightarrow F_2) & = & \{F_1 \leftrightarrow F_2\} \cup \mathbf{sub}(F_1) \cup \mathbf{sub}(F_2)
\end{array}
$$

- 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

$$F : (P \land Q) \to (P \lor \neg Q).$$

$$\begin{aligned}
\textbf{sub}(F) &= \{(P \land Q) \to (P \lor \neg Q)\} \cup \textbf{sub}(P \land Q) \cup \textbf{sub}(P \lor \neg Q) \\
&= \{(P \land Q) \to (P \lor \neg Q)\} \cup \\
&\quad \cup \{P \land Q\} \cup \textbf{sub}(P) \cup \textbf{sub}(Q) \\
&\quad \cup \{P \lor \neg Q\} \cup \textbf{sub}(P) \cup \textbf{sub}(\neg Q) \\
&= \{(P \land Q) \to (P \lor \neg Q)\} \\
&\quad \cup \{P \land Q\} \cup \{P\} \cup \{Q\} \\
&\quad \cup \{P \lor \neg Q\} \cup \{P\} \cup \{\neg Q\} \cup \textbf{sub}(Q) \\
&= \{(P \land Q) \to (P \lor \neg Q)\} \\
&\quad \cup \{P \land Q\} \cup \{P\} \cup \{Q\} \\
&\quad \cup \{P \lor \neg Q\} \cup \{P\} \cup \{\neg Q\} \cup \{Q\} \\
&= \{(P \land Q) \to (P \lor \neg Q), P \land Q, P \lor \neg Q, \neg Q, P, Q\}
\end{aligned}$$

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., $false$).

$$
\begin{array}{lll}
I \models \top & & \\
I \not\models \bot & & \\
I \models P & \text{iff} & I[P] = true \\
I \not\models P & \text{iff} & I[P] = false \\
I \models \neg F & \text{iff} & I \not\models F \\
I \models F_1 \land F_2 & \text{iff} & I \models F_1 \text{ and } I \models F_2 \\
I \models F_1 \lor F_2 & \text{iff} & I \models F_1 \text{ or } I \models F_2 \\
I \models F_1 \rightarrow F_2 & \text{iff} & I \not\models F_1 \text{ or } I \models F_2 \\
I \models F_1 \leftrightarrow F_2 & \text{iff} & (I \models F_1 \text{ and } I \models F_2) \text{ or } (I \not\models F_1 \text{ and } I \not\models F_2)
\end{array}
$$

Example: Semantics

Consider the formula

$$F : P \land Q \to P \lor \neg Q$$

and the interpretation

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

The truth value of F is computed as follows.

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

Satisfiability and Validity

- A formula F is **satisfiable** iff there exists an interpretation (somtimes 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]

$$F \text{ is valid iff } \neg F \text{ is unsatisfiable}$$

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

[1] In 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) $F : P \land Q \to Q \lor \neg Q$.

P	Q	$P \land Q$	$\neg Q$	$P \lor \neg Q$	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 interpreation).
  - 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:

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

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

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

$$\frac{I \models F \to G}{I \not\models F \mid I \models G} \qquad \frac{I \not\models F \to G}{I \models F, I \not\models G}$$

$$\frac{I \models F \leftrightarrow G}{I \models F \wedge G \mid I \models \neg F \wedge \neg G} \qquad \frac{I \not\models F \leftrightarrow G}{I \models F \wedge \neg G \mid I \models \neg F \wedge G}$$

$$\frac{I \models F \qquad I \not\models F}{I \models \bot}$$

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.

$$F : (P \to Q) \land (Q \to R) \to (P \to R)$$

Example 2: Semantic Argument Method

Prove that the following formula is valid.

$$F : (P \to Q) \land (Q \to R) \to (P \to R)$$

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

We have two cases from $\mathbf{6}$ and the rule of $\to$.

1. $\mathbf{8.}\ I \not\models P$: contradiction with 4
2. $\mathbf{9.}\ I \models Q$: At this point, consider other two cases from 7 and the rule of $\to$.
    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.

$$F : (P \lor Q) \to (P \land Q)$$

Example 3: Semantic Argument Method

Determine the validity of the following formula.

$$F : (P \lor Q) \to (P \land Q)$$

1.	$I \not\models (P \lor Q) \to (P \land Q)$	assumption
2.	$I \models P \lor Q$	by 1 and the rule of $\to$
3.	$I \not\models P \land Q$	by 1 and the rule of $\to$

We have two cases from $\mathbf{2}$ and the rule of $\lor$.

1. $\mathbf{4.}\ I \models P$: At this point, consider other two cases from 3 and the rule of $\land$
    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 $I : \{P \mapsto \top, Q \mapsto \bot\}$.

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**:

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

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

1.	$I \not\models F$	assumption
2.	$I \models (P \to Q) \land (Q \to R)$	by 1 and the rule of $\to$
3.	$I \not\models P \to R$	by 1 and the rule of $\to$
4.	$I \models P$	by 3 and the rule of $\to$
5.	$I \not\models R$	by 3 and the rule of $\to$
6.	$I \models P \to Q$	by 2 and the rule of $\land$
7.	$I \models Q \to R$	by 2 and the rule of $\land$
8.	$I \models Q$	by 4, 6, and modus ponens
9.	$I \models R$	by 7, 8, and modus ponens
10.	$I \models \bot$	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) $I \models P \land \neg\neg\neg P$

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

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

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

Substitution

- A substitution $\boldsymbol{\sigma}$ is a mapping from formulas to formulas:

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

- The domain of $\boldsymbol{\sigma}$, denoted **dom**, is

$$\mathbf{dom}(\boldsymbol{\sigma}) : \{\boldsymbol{F_1, \cdots, F_n}\}$$

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

$$\mathbf{range}(\boldsymbol{\sigma}) : \{\boldsymbol{G_1, \cdots, G_n}\}$$

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

Example: Substitution

Consider the formula F

$$F : P \land Q \to P \lor \neg Q$$

and the substitution σ

$$\sigma : \{P \mapsto R, P \land Q \mapsto P \to Q\}$$

Then,

$$F\sigma : (P \to Q) \to R \lor \neg Q$$

More Notations on Substitution

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

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

- For example, in the previous example, writing

$$F[P, P \land Q]\sigma : F[R, P \to Q]$$

emphasizes that P and $P \land Q$ are replaced by R and $P \to Q$, respectively.

Semantic Consequences of Substitution

Proposition (Substitution of Equivalent Formulas)

Consider a substitution $\sigma : \{F_1 \mapsto G_1, \cdots, F_n \mapsto G_n\}$ 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

$$F : (P \to Q) \leftrightarrow (\neg P \lor Q)$$

  is valid, every formula of the form $F_1 \to F_2$ is equivalent to $\neg F_1 \lor F_2$, 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 **range**(σ_1), and add the results to σ.
2. If F_i of $F_i \mapsto G_i$ appears in **dom**(σ_2) but not in **dom**(σ_1), add $F_i \mapsto G_i$ to σ.

For example,

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

Summary

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

Announcement: No class on March 12 (this Wednesday)

Next lecture (March 17): normal forms and decision procedures