

# EC4219: Software Engineering

## Lecture 3 — Propositional Logic (1) *Syntax and Semantics*

Sunbeom So  
2024 Spring

# Why Computational Logic in Software Engineering?

88, 10, 3, ...

0, 12, 7, ...

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

# Why Computational Logic in Software Engineering?

88, 10, 3, ...

0, 12, 7, ...

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

- [Q1] Test-cases for reaching line 5?

# Why Computational Logic in Software Engineering?

88, 10, 3, ...

0, 12, 7, ...

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

• **[Q1]** Test-cases for reaching line 5?

- (22, 11), (24, 12), (100, 50), ...

# Why Computational Logic in Software Engineering?

88, 10, 3, ...

0, 12, 7, ...

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

- **[Q1]** Test-cases for reaching line 5?

- (22, 11), (24, 12), (100, 50), ...

- **[Q2]** Probability of generating such tests?  
(assumption:  $0 \leq x, y \leq 100$ )

# Why Computational Logic in Software Engineering?

88, 10, 3, ...

0, 12, 7, ...

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

- **[Q1]** Test-cases for reaching line 5?

- (22, 11), (24, 12), (100, 50), ...

- **[Q2]** Probability of generating such tests?  
(assumption:  $0 \leq x, y \leq 100$ )

- **0.4%**

# Why Computational Logic in Software Engineering?

88, 10, 3, ...

0, 12, 7, ...

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

- **[Q1]** Test-cases for reaching line 5?

- (22, 11), (24, 12), (100, 50), ...

- **[Q2]** Probability of generating such tests?  
(assumption:  $0 \leq x, y \leq 100$ )

- **0.4%**

Can we do better in systematic ways?

# 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              /* Error */  
6          }  
7      }  
8  }
```

$\alpha$

$\beta$

Constraint for reaching line 5

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



# 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             /* Error */  
6         }  
7     }  
8 }
```

$\alpha$

$\beta$

Constraint for reaching line 5

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



SMT Solver (theorem prover)

# 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             /* Error */  
6         }  
7     }  
8 }
```

$\alpha$

$\beta$

Constraint for reaching line 5

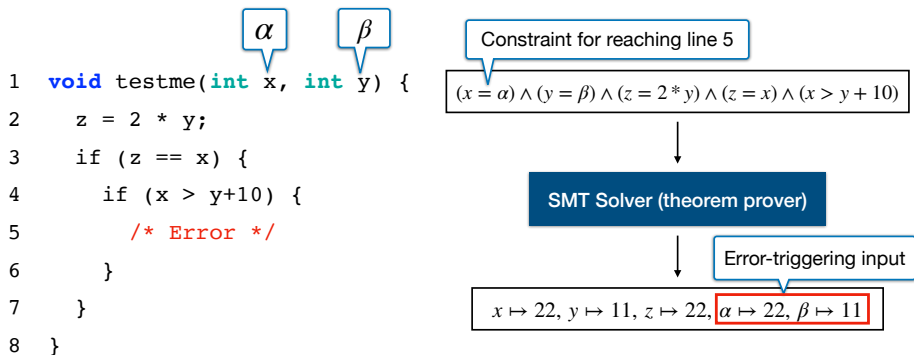
$(x = \alpha) \wedge (y = \beta) \wedge (z = 2 * y) \wedge (z = x) \wedge (x > y + 10)$

SMT Solver (theorem prover)

Error-triggering input

$x \mapsto 22, y \mapsto 11, z \mapsto 22, \alpha \mapsto 22, \beta \mapsto 11$

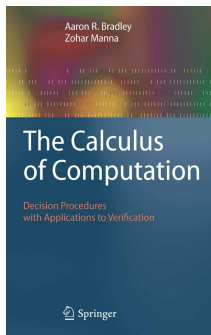
# Why Computational Logic in Software Engineering?



- 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:

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

- $A$ : 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: Context-free Grammar

Another way to define the set is to use context-free grammar.

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

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

- The inference rules can be expressed by the context-free grammar:

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

- Interpreted as:
  - $0$  is a natural number.
  - 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  $\alpha$  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  $G$ .

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

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

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

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

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

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

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

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

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

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



## Example: Subformula

Find subformulas of the formula

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

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

- 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 values.
- 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.  $I \models P \vee \neg Q$             by 1 and semantics of  $\vee$
6.  $I \models F$                       by 4 and semantics of  $\rightarrow$

# Satisfiability and Validity

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

$F$  is valid iff  $\neg F$  is unsatisfiable

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

# 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.
  - ▶ Case 1) If every branch of the proof derives a contradiction, then  $F$  is valid.
  - ▶ Case 2) If some branch of the proof never derives a contradiction, then  $F$  is invalid. This branch describes a falsifying interpretation of  $F$ .
- SAT solvers use both search and deduction.

# Deduction Rules for Propositional Logic

Proof rules used in the semantic argument method:

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

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

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

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

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

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

## Example 1: Semantic Argument Method

Prove that the following formula is valid using the semantic argument method.

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

## Example 1: Semantic Argument Method

Prove that the following formula is valid using the semantic argument method.

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

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



## Example 2: Semantic Argument Method

Prove that the following formula is valid using the semantic argument method.

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

# 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  $I$ ,  $I \models F_1 \rightarrow F_2$ .

- Note 1)  $F_1 \iff F_2$  and  $F_1 \implies F_2$  are not formulas. They are semantic assertions!
- We can check equivalence and implication by checking validity.

# Substitution

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

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

- The domain of  $\sigma$ , 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  $\sigma$  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 **dom**( $\sigma$ ) and  $F_k$  is a strict subformula of  $F_j$ , then  $F_k$  is replaced first.

## Example: Substitution

Consider the formula  $F$

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

and the substitution  $\sigma$

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

Then,

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

# More Notations on Substitution

- A variable substitution is a substitution in which the 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

## Lemma (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,  $F \iff F\sigma$ .

## Lemma (Valid Template)

If  $F$  is valid and  $G = F\sigma$  for some variable substitution  $\sigma$ , 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 any formulas  $F_1$  and  $F_2$ .

# Composition of Substitutions

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

- 1 Apply  $\sigma_2$  to each formula of the range of  $\sigma_1$ , and add the results to  $\sigma$ .
- 2 If  $F_i$  of  $F_i \mapsto G_i$  appears in the domain of  $\sigma_2$  but not in the domain of  $\sigma_1$ , add  $F_i \mapsto G_i$  to  $\sigma$ .

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 systemically analyzing software.
- Syntax and semantics of propositional logic
- Satisfiability and validity
- Basic notations (inference rule, context-free grammar, substitution)

Next lecture: normal forms and decision procedures