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, …

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

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 <= x, y <= 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 <= x, y <= 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 <= x, y <= 100)
  - **0.4%**

Can we do better in systematic ways?

Why Computational Logic in Software Engineering?

α β

```
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 }
```

Constraint for reaching line 5

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

Why Computational Logic in Software Engineering?

α β

```
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  }
```

Constraint for reaching line 5

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

$\downarrow$

SMT Solver (theorem prover)

Why Computational Logic in Software Engineering?

α (x), β (y)

```
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  }
```

Constraint for reaching line 5

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

↓

SMT Solver (theorem prover)

↓

Error-triggering input

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

Why Computational Logic in Software Engineering?

α β

```
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  }
```

Constraint for reaching line 5

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

$\downarrow$

SMT Solver (theorem prover)

$\downarrow$

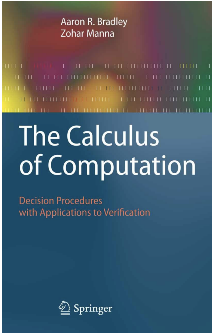
Error-triggering input

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

- 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., $\boldsymbol{B}$):

$$\overline{\boldsymbol{B}}$$

- 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: Context-free Grammar

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

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

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

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

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

- 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: $\perp$ (*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

$$
\begin{array}{rcll}
F & \rightarrow & \perp & \\
 & \mid & \top & \\
 & \mid & P & \\
 & \mid & \neg F & \text{negation (``not'')} \\
 & \mid & F_1 \wedge F_2 & \text{conjunction (``and'')} \\
 & \mid & F_1 \vee F_2 & \text{disjunction (``or'')} \\
 & \mid & F_1 \rightarrow F_2 & \text{implication (``implies'')} \\
 & \mid & F_1 \leftrightarrow F_2 & \text{iff (``if and only if'')}
\end{array}
$$

Subformula

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

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

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

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

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 values.
- Semantics is inductively defined, where we write $I \models F$ (resp., $I \not\models F$) iff F evaluates to true (resp., false).

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

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.	$I \models P \lor \neg Q$	by 1 and semantics of $\lor$
6.	$I \models F$	by 4 and semantics of $\to$

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*:

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

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

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.
  - 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.

Proof rules used in the semantic argument method:

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

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

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

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

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

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

Example 1: Semantic Argument Method

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

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

Example 1: Semantic Argument Method

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

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

1.	$I \not\models P \land Q \to P \lor \neg Q$	assumption
2.	$I \models P \land Q$	by 1 and the rule $\to$
3.	$I \not\models P \lor \neg Q$	by 1 and the rule $\to$
4.	$I \models P$	by 2 and the rule $\land$
5.	$I \not\models P$	by 3 and the rule $\lor$
6.	$I \models \bot$	4 and 5 are contradictory

Example 2: Semantic Argument Method

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

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

Equivalence and Implication

- Two formulas $\boldsymbol{F_1}$ and $\boldsymbol{F_2}$ are equivalent:

$$\boldsymbol{F_1} \iff \boldsymbol{F_2}$$

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

- Formula $\boldsymbol{F_1}$ implies $\boldsymbol{F_2}$

$$\boldsymbol{F_1} \implies \boldsymbol{F_2}$$

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

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

Substitution

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

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

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

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

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

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

- 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 $\mathbf{dom}(\sigma)$ and F_k is a strict subformula of F_j, then F_j is replaced first.

Example: Substitution

Consider the formula F

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

and the substitution σ

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

Then,

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

More Notations on Substitution

- A variable substitution is a substitution in which the domain consists only of propositional variables.
- When we write $F[F_1, \cdots, F_n]$, 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

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

Lemma (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 any formulas F_1 and F_2.

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 the range of σ_1, and add the results to σ.
2. If F_i of $F_i \mapsto G_i$ appears in the domain of σ_2 but not in the domain of σ_1, add $F_i \mapsto G_i$ to σ.

For example,

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

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