EC4219: Software Engineering

Lecture 3 — Propositional Logic (1)
Syntax and Semantics

Sunbeom So
2024 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        /* Error */
6      }
7    }
8  }
```

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  }
```

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

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  }
```

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

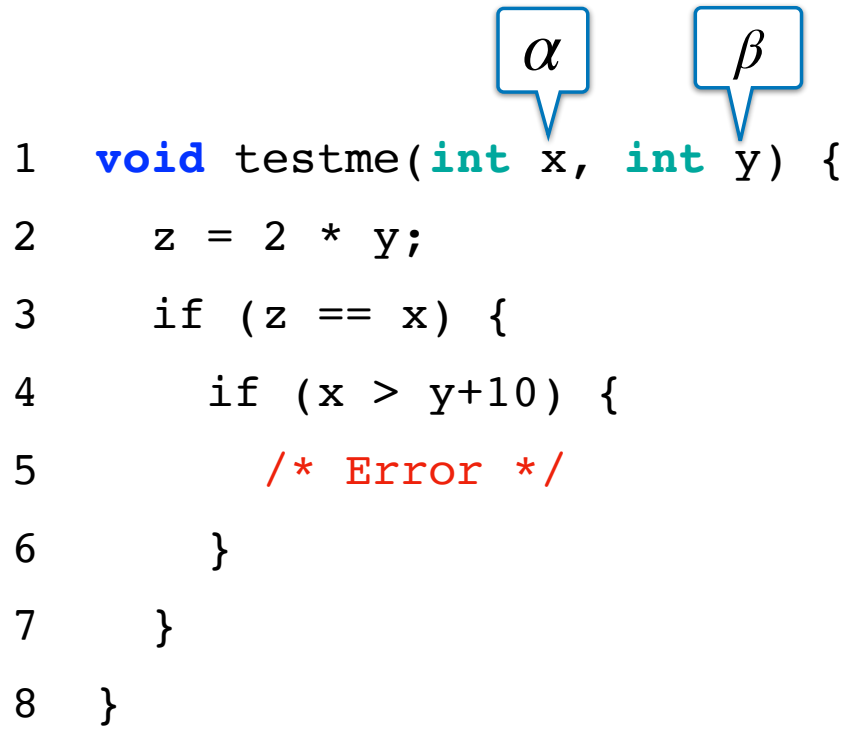
Why Computational Logic in Software Engineering?


```
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?

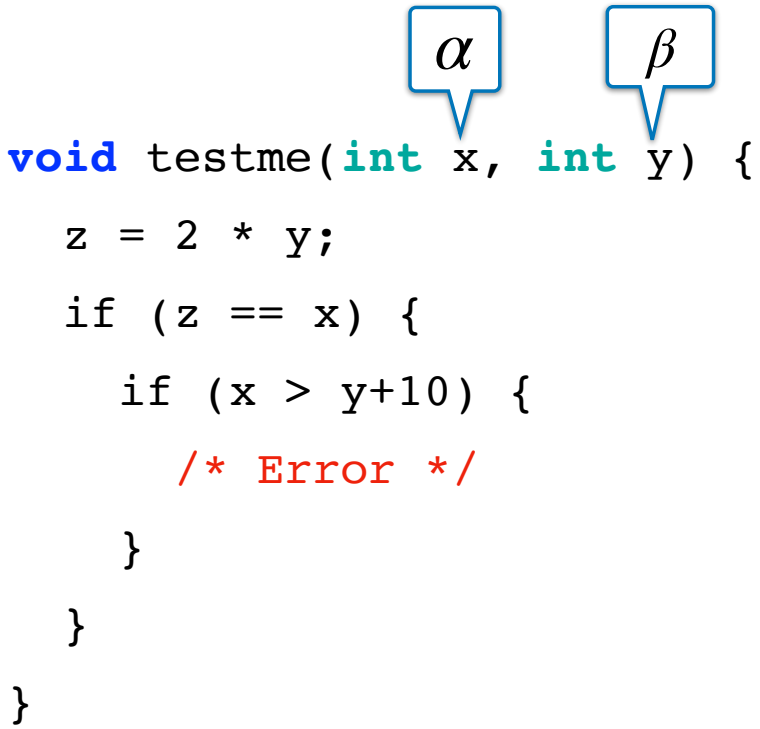
Why Computational Logic in Software Engineering?

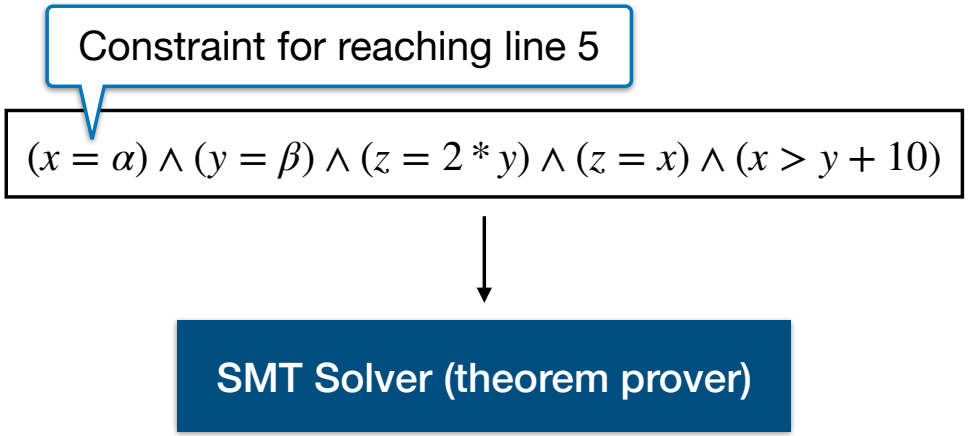


Constraint for reaching line 5

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

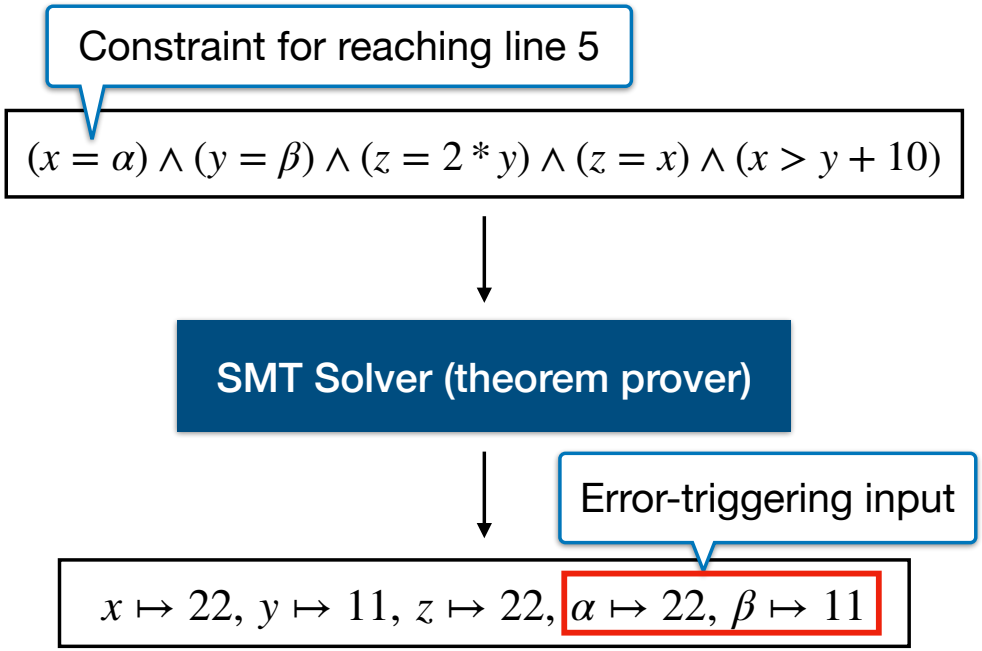
Why Computational Logic in Software Engineering?





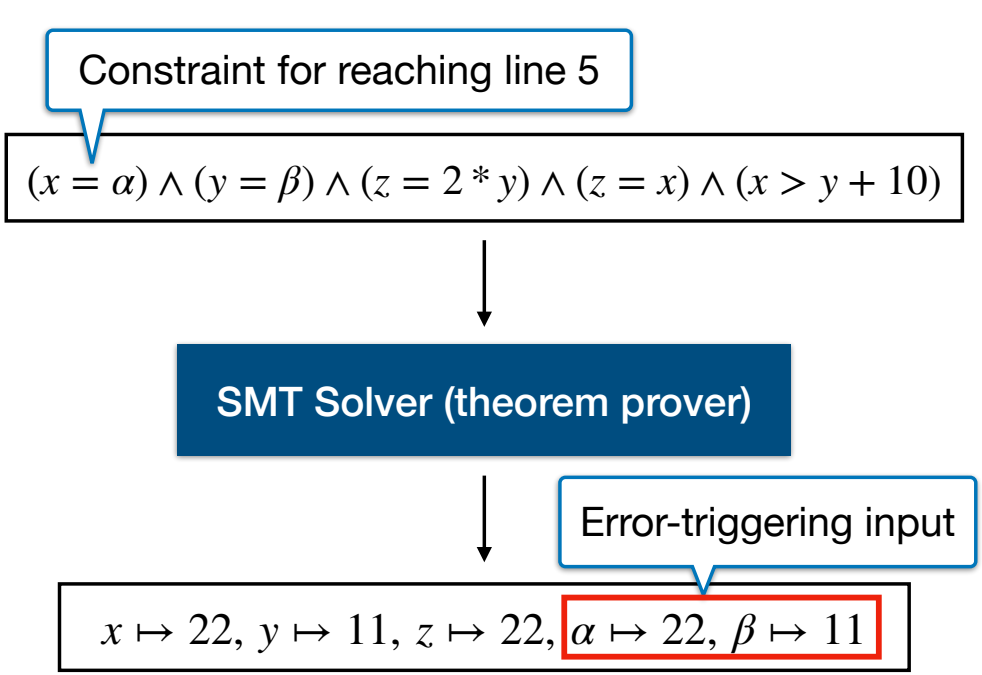
Why Computational Logic in Software Engineering?

```
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  }
```



Why Computational Logic in Software Engineering?

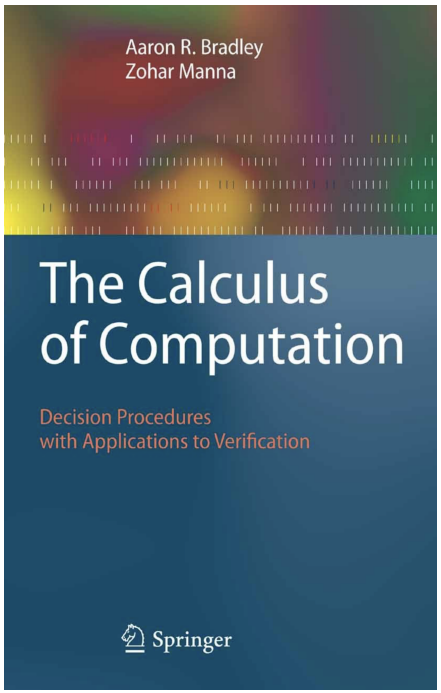
```
1  void testme(int x, int y) {
2    z = 2 * y;
3    if (z == x) {
4      if (x > y+10) {
5        /* Error */
6      }
7    }
8  }
```



- 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):

$$\frac{}{B}$$

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

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

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

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

- 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 $\boldsymbol{P}$, $\boldsymbol{Q}$, $\boldsymbol{R}$, $\cdots$
- **Literal:** an atom $\boldsymbol{\alpha}$ or its negation $\neg\boldsymbol{\alpha}$
- **Formula:** a literal, or the application of a logical(boolean) connective to formulas

$\boldsymbol{F}$	$\rightarrow$	$\bot$	
	$\mid$	$\top$	
	$\mid$	$\boldsymbol{P}$	
	$\mid$	$\neg\boldsymbol{F}$	negation ("not")
	$\mid$	$\boldsymbol{F_1} \wedge \boldsymbol{F_2}$	conjunction ("and")
	$\mid$	$\boldsymbol{F_1} \vee \boldsymbol{F_2}$	disjunction ("or")
	$\mid$	$\boldsymbol{F_1} \rightarrow \boldsymbol{F_2}$	implication ("implies")
	$\mid$	$\boldsymbol{F_1} \leftrightarrow \boldsymbol{F_2}$	iff ("if and only if")

Subformula

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

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

- 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

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

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

Semantics

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

$\boldsymbol{I} \models \top$

$\boldsymbol{I} \not\models \bot$

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

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

$\boldsymbol{I} \models \neg \boldsymbol{F}$ iff $\boldsymbol{I} \not\models \boldsymbol{F}$

$\boldsymbol{I} \models \boldsymbol{F_1} \wedge \boldsymbol{F_2}$ iff $\boldsymbol{I} \models \boldsymbol{F_1}$ and $\boldsymbol{I} \models \boldsymbol{F_2}$

$\boldsymbol{I} \models \boldsymbol{F_1} \vee \boldsymbol{F_2}$ iff $\boldsymbol{I} \models \boldsymbol{F_1}$ or $\boldsymbol{I} \models \boldsymbol{F_2}$

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

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

Example: Semantics

Consider the formula

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

and the interpretation

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

The truth value of F is computed as follows.

1.	$I \models P$	since $I[P] = true$
2.	$I \not\models Q$	since $I[Q] = false$
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 $\boldsymbol{F}$ is *satisfiable* iff there exists an interpretation $\boldsymbol{I}$ such that $\boldsymbol{I} \models \boldsymbol{F}$.
- A formula $\boldsymbol{F}$ is *valid* iff for all interpretations $\boldsymbol{I}$, $\boldsymbol{I} \models \boldsymbol{F}$.
- Satisfiability and validity are *dual*:

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

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

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 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.

Deduction Rules for Propositional Logic

Proof rules used in the semantic argument method:

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

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

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

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

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

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

Example 1: Semantic Argument Method

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

$$\boldsymbol{F : P \wedge Q \rightarrow P \vee \neg Q}$$

Example 1: Semantic Argument Method

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

$$\boldsymbol{F : P \wedge Q \rightarrow P \vee \neg Q}$$

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

Example 2: Semantic Argument Method

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

$$\boldsymbol{F : (P \to Q) \wedge (Q \to R) \to (P \to R)}$$

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

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

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

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

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

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

- 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 and $\boldsymbol{F_k}$ is a strict subformula of $\boldsymbol{F_j}$, then $\boldsymbol{F_j}$ is replaced first.

Example: Substitution

Consider the formula $\boldsymbol{F}$

$$\boldsymbol{F : P \wedge Q \rightarrow P \vee \neg Q}$$

and the substitution $\boldsymbol{\sigma}$

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

Then,

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

More Notations on Substitution

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

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

- For example, in the previous example, writing

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

emphasizes that $\boldsymbol{P}$ and $\boldsymbol{P} \wedge \boldsymbol{Q}$ are replaced by $\boldsymbol{R}$ and $\boldsymbol{P} \rightarrow \boldsymbol{Q}$, 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

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

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

Composition of Substitutions

Given substitutions $\boldsymbol{\sigma_1}$ and $\boldsymbol{\sigma_2}$, their composition $\boldsymbol{\sigma = \sigma_1\sigma_2}$ ("apply $\boldsymbol{\sigma_1}$ and then $\boldsymbol{\sigma_2}$") is computed as follows.

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

For example,

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

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, context-free grammar, substitution)

Next lecture: normal Forms and decision procedures