Formal Software Development Methods:

- Propositional Logic
- Proof systems for PL
- SAT solvers
- Quantifier-free logic / simple assertions
- First order logic

Instructor: Madhusudan Parthasarathy

UIUC

Fall 2026

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Propositional Logic

- In this lecture, we will learn about propositional logic which is the foundation of first-order and higher-order logic

- The satisfiability of Boolean constraints as we will see here is a core component of program verification methods

Background read: Chapter 1 (Sec. 1.1, 1.2, 1.3) of
https://courses.grainger.illinois.edu/cs474/sp2026/Logic_in_CS_Madhu_Mahesh_2026.pdf

Propositional Logic

- Syntax
- Semantics (truth tables)
- Proofs

Propositional Logic

The Language of Propositional Logic

- Constants $\{T, F\}$, left "(" and right ")" parenthesis
- Countable set AP of propositional variables (x, y, z), a.k.a. propositional atoms/variables, a.k.a. atomic propositions
- Logical connectives: ∧ (and); ∨ (or); ¬ (not); ⇒ (implies);⇔ (if and only if)

Propositional Logic

We can write it as a grammar:

- $C ::= T \mid F$
- $AP ::= x \mid y \mid z \mid \ldots$
- $PROP ::= C \mid AP \mid (PROP) \mid \neg PROP$
  $\mid PROP \wedge PROP \mid PROP \vee PROP$
  $\mid PROP \Rightarrow PROP \mid PROP \Leftrightarrow PROP$

We can get various "sentences" in this language.

E.g. $x \wedge y, (x \wedge y) \Rightarrow (x \vee y), x \vee \neg x \Leftrightarrow T$

But what is their meaning?

Toward Propositional Logic Semantics

Model (valuation/interpretation) for Propositional Logic:

- Interpretation function giving meaning to propositional variables

Semantics (meaning) of logical formulas:

Defined by structural recursion on formulas

- Standard Model of Propositional Logic
  - **Boolean values** B = {true,false}
  - **Valuation** v : AP → B

Example valuation:

AP	B
x	true
y	false
z	true

Semantics of Propositional Logic

Interpretation $I_v(\alpha)$ is defined by structural induction on α , extending from Prop to all formulas:

- $I_v\,(T) = T$ and $I_v\,(F) = F$
- If $\alpha = p \in AP$ then $I_v\,(\alpha) = v(p)$
- If $\alpha = \neg\beta$, then
  - $I_v\,(\alpha) = T$ if $I_v\,(\beta) = F$
    $= F$ otherwise
- If $\alpha = \beta_1 \vee \beta_2$
  $I_v\,(\alpha) = T$ if $I_v\,(\beta_1) = T$ *or* $Iv\,(\beta_2) = T$
  $= F$ *otherwise*
- If $\alpha = \beta_1 \wedge \beta_2$
  $I_v\,(\alpha) = T$ if $I_v\,(\beta_1) = T$ *and* $I_v\,(\beta_2) = T$
  $= F$ *otherwise*
- If $\alpha = \beta_1 \Rightarrow \beta_2$
  $I_v\,(\alpha) = T$ if $I_v\,(\beta_1) = F$ *or* $Iv\,(\beta_2) = T$
  $= F$ *otherwise*
- If $\alpha = \beta_1 \Leftrightarrow \beta_2$
  $I_v\,(\alpha) = T$ if $I_v\,(\beta_1) = Iv\,(\beta_2)$
  $= F$ *otherwise*

Example

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \Rightarrow q$	$p \Leftrightarrow q$
true	true	false	true	true	true	true
true	false	false	false	true	false	false
false	true	true	false	true	true	false
false	false	true	false	false	true	true

$$I_v(\alpha) \quad is\ computable\ given\ v \quad (in\ linear\ time)$$

Compute by recursion, following the recursive definition

Called the circuit value problem (CVP), and is actually PTIME-complete.

Semantics of Propositional Logic

- **Satisfaction relation $\models$:** Given valuation v and proposition α we write $v \models \alpha$ iff $I_v(\alpha) = T$
  - We say valuation v **satisfies** p, or v **models** p
  - Write $v \not\models p$ if $I_v(p) = \text{false}$

- α is **satisfiable** if there <u>exists</u> a valuation v such that $v \models \alpha$

  Type equation here.

- α is **valid**, a.k.a. a **tautology** if <u>for every</u> valuation v we have $v \models \alpha$

  Type equation here.

- α is **logically equivalent** to β , $\alpha \equiv \beta$ if for every valuation v

  $v \models \alpha$ iff $v \models \beta$.

  Logical equivalence is an equivalence relation

Example Tautology

$$A \Rightarrow ((A \Rightarrow B) \Rightarrow B)$$

A	B	$A \Rightarrow B$	$(A \Rightarrow B) \Rightarrow B$	$A \Rightarrow ((A \Rightarrow B) \Rightarrow B)$
true	true	true	true	true
true	false	false	true	true
false	true	true	true	true
false	false	true	false	true

- The truth table method does not scale to larger formulas
- Next, we introduce a deductive method called semantic argument

Syllogism and Logic of the Greeks

Axiomatic Proof Systems

- Axioms and Proof rules (both can be infinite, but finitely presented)

- A proof/derivation is a finite sequence of statements

  $s_0, s_1, s_2, \ldots s_n$

  where each s_i *is an axiom or*

  is derived from previous statements using a proof rule

  s_n *is the conclusion of the proof.*

Whether a sequence of statements is a proof is checkable easily as proof rules are syntactic.

A proof system for propositional logic

(A1) $\alpha \Rightarrow (\beta \Rightarrow \alpha)$

(A2) $\big(\alpha \Rightarrow (\beta \Rightarrow \gamma)\big) \Rightarrow ((\alpha \Rightarrow \beta) \Rightarrow (\alpha \Rightarrow \gamma))$

(A3) $(\neg\beta \Rightarrow \neg\alpha) \Rightarrow ((\neg\beta \Rightarrow \alpha) \Rightarrow \beta)$

(MP) $\dfrac{\alpha \qquad \alpha \Rightarrow \beta}{\beta}$

Example proof

1.	$(p \supset ((p \supset p) \supset p)) \supset ((p \supset (p \supset p)) \supset (p \supset p))$	Instance of (A2)
2.	$p \supset ((p \supset p) \supset p)$	Instance of (A1)
3.	$(p \supset (p \supset p)) \supset (p \supset p)$	From 1 and 2 by MP
4.	$p \supset (p \supset p)$	Instance of (A1)
5.	$p \supset p$	From 3 and 4 by MP

In the above, read $\supset$ as $\Rightarrow$

Above proves the conclusion $p \Rightarrow p$ from axioms and applications of proof rule

Soundness and Completeness for Prop Logic

A proof system for prop logic is *sound* if
all conclusions are valid (tautologies)

A proof system for prop logic is *complete* if
every valid formula has a proof

The proof system we saw (3 axioms + MP) is sound and complete!

Soundness: Verify all axioms are tautologies and MP is sound,
i.e., if α is valid and $\alpha \Rightarrow \beta$ is valid, then β is valid.
Then argue by induction that all proved statements are valid.

Completeness: Harder, but true!

Notions of Logical Consequence

Semantic entailment: $\Gamma \vDash \alpha$

- for every valuation v, whenever we have $\mathrm{v} \vDash \beta$ for each $\beta \in \Gamma$ then we have $\mathrm{v} \vDash \alpha$

Syntactic entailment: $\Gamma \vdash \alpha$ is true iff there is a proof from the formulas in Γ to the formula α

- ***Deductive system***: a list of rules that express which formulas can legally follow which.
- ***Proof (aka derivation):*** a sequence of formulas that follow the rules of the deductive apparatus

Soundness: If $\Gamma \vdash \alpha$ then $\Gamma \vDash \alpha$

Completeness: If $\Gamma \vDash \beta$ then $\Gamma \vdash \alpha$

Some Useful Logical Equivalences

$\neg\neg A \equiv A$	$\neg \boldsymbol{T} \equiv \boldsymbol{F}$
$(A \vee A) \equiv A$	$(A \vee B) \vee C \equiv A \vee (B \vee C)$
$(A \wedge A) \equiv A$	$(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$
$(A \vee B) \equiv (B \vee A)$	$\neg(A \vee B) \equiv (\neg A) \wedge (\neg B)$
$(A \wedge B) \equiv (B \wedge A)$	$\neg(A \wedge B) \equiv (\neg A) \vee (\neg B)$
$(A \wedge \neg A) \equiv \boldsymbol{F}$	$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$
$(A \vee \neg A) \equiv \boldsymbol{T}$	$(A \wedge B) \vee C \equiv (A \vee C) \wedge (B \vee C)$
$(\boldsymbol{T} \wedge A) \equiv A$	$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
$(\boldsymbol{T} \vee A) \equiv \boldsymbol{T}$	$(A \wedge B) \vee C \equiv (A \wedge C) \vee (B \wedge C)$
$(\boldsymbol{F} \wedge A) \equiv \boldsymbol{F}$	$(F \vee A) \equiv A$

Desired Properties of a Proof System

- Soundness: if the system proves a formula, it is valid
- Completeness: if some formula is valid, the system proves it
- Both the Truth-table and Semantic argument method are sound and complete

Boolean Satisfiability

Literal	Propositional variables, e.g., A
Clause	Disjunction of literals
Conjunctive normal form (CNF)	$\bigwedge_{i \in I} \bigvee_{j \in J_i} AP_{i,j}$
Disjunctive normal form (DNF)	$\bigvee_{i \in I} \bigwedge_{j \in J_i} AP_{i,j}$

where I and J are index sets

Does there exist a valuation v to the literals such that the formula is satisfied?

What is the computational complexity of finding a satisfying assignment of variables?

Tool Support: Boolean Satisfiability

SAT Solver

- Takes a logical formula
- Returns a satisfying valuation (or unsat)
- Difficulty: Answering if a CNF formula is satisfiable is NP-complete

Power-horses of many of today's analysis

- Finding the solution is NP-hard in general
- There are many good heuristics for common formulas arising from analysis of programs

RESOURCE: See ``*The Quest for Efficient Boolean Satisfiability Solvers" by Sharad Malik:*

Slides: https://www.princeton.edu/~sharad/CMUSATSeminar.pdf

Paper: https://www.princeton.edu/~chaff/publication/cade_cav_2002.pdf

Basic Solution Algorithm

- DPLL (Davis–Putnam–Logemann–Loveland), 1961: many modern algorithms derive from it in some way
- Operates on the formula in CNF
- Core – Backtracking:
  - Recursively, choose a literal, set a truth value, check if the simplified formula was satisfied;
  - if not invert the truth value of the literal, and try again
- Aggressive simplification:
  - Unit clauses: contains only a single unassigned literal – it can be set in only one way to make the clause true!
  - Pure literal elimination: only x (or $\neg$x) occur in all clauses – assign it to make all clauses that contain it true

DPLL algorithm

```
Algorithm DPLL
    Input: A set of clauses Φ in CNF.
    Output: A Truth Value.

function DPLL(Φ)
    if Φ is a consistent set of literals then
        return true;
    if Φ contains an empty clause then
        return false;
    for every unit clause {l} in Φ do
        Φ ← unit-propagate(l, Φ);
    for every literal l that occurs pure in Φ do
        Φ ← pure-literal-assign(l, Φ);
    l ← choose-literal(Φ);
    return DPLL(Φ ∧ {l}) or DPLL(Φ ∧ {not(l)});
```


From Wikipedia: https://en.wikipedia.org/wiki/DPLL_algorithm

DPLL Algorithm

$$(A \vee B) \wedge (B \vee \neg C \vee D) \wedge (\neg A \vee \neg B) \wedge (\neg A \vee C \vee \neg D) \wedge A$$

Assignment	Formula
$A \rightarrow True$	$(B \vee \neg C \vee D) \wedge (\neg A \vee \neg B) \wedge (\neg A \vee C \vee \neg D)$
$A \rightarrow True, B \rightarrow False$	$(B \vee \neg C \vee D) \wedge (\neg A \vee C \vee \neg D)$
$A \rightarrow True, B \rightarrow False$ $C \rightarrow True$	$(B \vee \neg C \vee D)$
$A \rightarrow True, B \rightarrow False$ $C \rightarrow True, D \rightarrow True$	