$\oplus$	a	b	a $\oplus$ b
	T	T	F
	T	F	T
	F	T	T
	F	F	F

Week-5

L1 Logic-Basics needed for ST

- The fragment of logic that we consider is → predicate logic or 1st order logic.
- An atomic proposition is a term that is either T or F. Propositional logic deals with combining propositions using logical connectives to form formulas. $\vee$ (or), $\wedge$ (and), $\neg$ (not), $\supset$ / $\rightarrow$ (implies), $\equiv$ / $\leftrightarrow$ (iff / equivalence).
- Propositions are basic building blocks of logic. It is a declarative sentence is either T or F but not both.
- $P = \{p_0, p_1, \dots\}$ is a countably ∞ set of propositions. The set Φ of formulas of propositional logic is the smallest set.
- A valuation is a func^n $v : P \rightarrow \{T, F\}$
- Truth tables are simple way of calculating the semantic of a given propositional logic formula. We split a formula into its constituent sub-formulas repeatedly till we reach propositions.

$\neg \Rightarrow$	p	$\neg$ p
	T	F
	F	T

$\wedge$	p	q	p $\wedge$ q
	T	T	T
	T	F	F
	F	T	F
	F	F	F

$\vee \Rightarrow$	p	q	p $\vee$ q
	T	T	T
	T	F	T
	F	T	T
	F	F	F

$\supset \Rightarrow$	p	q	p $\supset$ q
	T	T	T
	T	F	F
	F	T	T
	F	F	T

$\equiv \Rightarrow$	p	q	p $\equiv$ q
	T	T	T
	T	F	F
	F	T	F
	F	F	T

- A formula α is said to be satisfiable if there exists a valuation v s.t. $v(\alpha) = T$. $v \models \alpha$
- A formula α is said to be valid / tautology if for every valuation v, we have $v(\alpha) = T$.
- To check if a formula α is satisfiable, construct the truth table for α & check if there is an entry that makes α T. It takes exp. time in the len of α. Satisfiability of propositional logic is NP-Comp.

- A predicate is an expression that evaluates to a Boolean value. It can have vars. that have func. Relational operators ($>, <, =, \leq, \geq$) constitutes predicates
  Eg. $(x \geq y) \vee \neg flag \vee f(x)$ → predicate
- A clause is a predicate that doesn't contain logical operator. It is a boolean atomic entity which evaluates to T or F.
- Satisfiability for predicate logic is undecidable.

L2 Logic: Coverage Criteria - 1

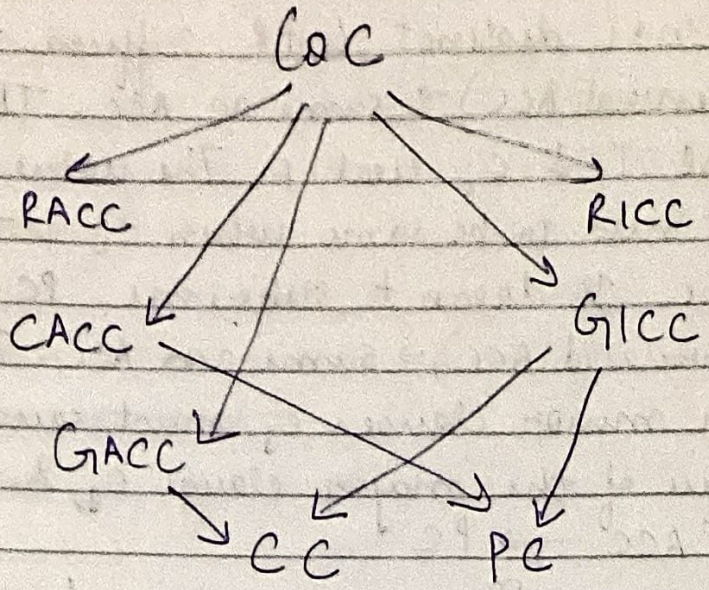
- P → set of predicates, C → set of clauses in the predicates in P
  C_p → clauses in p
  $C_p = \{c \mid c \in P\} \Rightarrow C = \bigcup_{p \in P} C_p$.
- Predicate Coverage (PC) → For each $p \in P$, TR has 2 req.
  ↳ p evaluates to T & p eval. to F.
  same as edge coverage.
- Clause Coverage (CC) → For each $c \in C$, TR has 2 req.
  ↳ c eval. to T & c eval. to F.
  CC doesn't subsume PC.
- Combinatorial Coverage (CoC) / Multiple cond^n cov. → For each $p \in P$, TR contains test req. for the clauses in C_p to evaluate to each possible combin^n of truth values. Total 2^n possible (n clauses). Sometimes, not feasible.
- Active Clause Coverage (ACC) → For each $p \in P$ & each major clause $c_i \in C_p$, choose minor clauses $c_j (i \neq j)$, so that c_i determins p. TR for each c_i,
  $c_i \Rightarrow T$, $c_i \Rightarrow F$.
  $a \vee b \Rightarrow (T,F), (F,T), (F,F)$

modified condition coverage

- For ACC, $n+1$ distinct testR suffices. → MCDC
- GACC (General ACC) → same as ACC. TR for each c_i, c_i eval. T & c_i eval. F. The values chosen for c_j do not need to be same when c_i is true as when c_i is false. It doesn't subsume PC.
- CACC (Correlated ACC) → same as ACC. The values chosen for minor clauses c_j must cause p to be true, for 1 value of the major clause c_i & false for the other. CACC ⟶ PC
- RACC → same. The values chosen for minor clauses c_j must be the same when c_i is true as when c_i is false.

L3 Part-2

- Complementary criterion to ACC, ensures that the major clause doesn't affect the predicate.
- ICC - For each $p \in P$ & each major clause $c_i \in C_p$, choose minor clauses c_j, $j \neq i$, so that c_i doesn't determine p.
- GICC → same as ICC. TR has 4 req. for c_i -
  (i) c_i is T with p T.
  (ii) c_i is F with p T.
  (iii) c_i is T with p F.
  (iv) c_i is F with p T.
- Logic criteria may also be infeasible. RACC can also be infeasible.



Logic Subsumpⁿ

L4 Logic Coverage Criteria: Make Clause determine Predicate

- Test cases for satisfying TRs for predicate, clause & CoC are easy to define. We need to identify the correct predicate and pass them to a SAT/SMT solver to check if the predicate is satisfiable.
- For simple cases, we can solve them manually, in general, symbolic execn can be used to solve this problem. For ACC criteria, we have to make clause chosen as major clause determining a predicate.
- 1. choose c (clause) of p (predicate)
  2. p_c = true $\oplus^{(xor)}$ p_c = false
     ↳ c determines p ↳ p is indep. of c.

Eg.1 $p = a \vee b \Rightarrow p_a = p_{a=T} \oplus p_{a=F}$
$= T \vee b \oplus F \vee b = \text{true} \oplus b = \neg b$
for a to determine p, b must be false.

Eg.2 $p = a \wedge b \Rightarrow p_a = T \wedge b \oplus F \wedge b = b \oplus \text{false} = b$
a to deter p, b must be true.

Eg.3 $p = a \equiv b \Rightarrow p_a = T \equiv b \oplus F \equiv b = b \oplus \neg b = \text{true}$
a determines p, regardless of b's value.

- if c is true, ICC becomes infeasible, if c is false, ACC becomes infeasible.

Eg4 $p = a \wedge (b \vee c) \Rightarrow p_a = T \wedge (b \vee c) \oplus F \wedge (b \vee c) = (b \vee c) \oplus \text{false}$
$= (b \vee c)$
a determines p, if (b∨c) is true.
↳ $p_b = a \vee \neg c$

Eg. $p = (a \vee b) \wedge c$ $p_a = \neg b \wedge c$ $p_c = a \vee b$
↳ truth table $p_b = \neg a \wedge c$

GACC → each major clause be T & F, minor clause such that major determines predicate

	a	b	c	p	
$a=T \wedge p_a$	T	F	T	T	✓
$a=F \wedge p_a$	F	F	T	F	✓
$b=T \wedge p_b$	F	T	T	T	
$b=F \wedge p_b$	F	F	T	F	✓
$c=T \wedge p_c$	T	F	T	T	✓
$c=F \wedge p_a$	F	T	F	F	

} 4 unique

LS Applied to Test Code

- Predicates are derived from decision statements. In programs most predicates have < 4 clauses. If a predicate has only 1 clause, CoC, ACC, ICC & CC all becomes PC
- Var. in the predicates that are not inputs to the program are c/d internal variables.
- Thermostat eg.

$p = (a \,||\, (b \,\&\&\, c)) \,\&\&\, d$

$p_a = (T \vee (b \wedge c)) \wedge d \;\oplus\; (F \vee (b \wedge c)) \wedge d$

$= T \wedge d \oplus (b \wedge c) \wedge d$

$= d \oplus (b \wedge c) \wedge d \quad = (\neg b \wedge \neg c) \vee d$

Same find p_b, p_c, p_d. Truth table for p.

→ CACC(TR)