

Logic Coverage

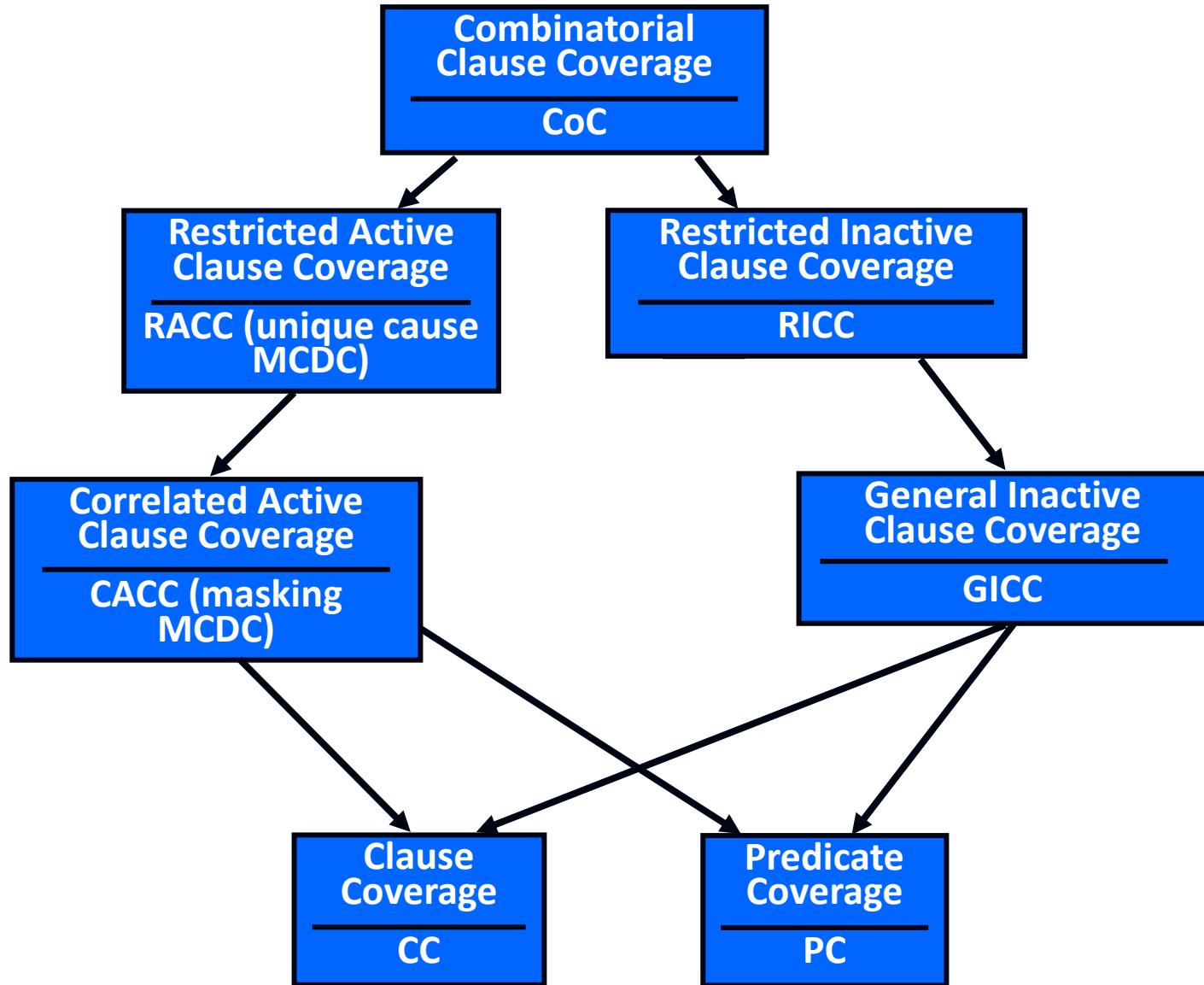
Moonzoo Kim
School of Computing
KAIST

*The original slides are taken from Chap. 8 of Intro. to SW Testing
2nd ed by Ammann and Offutt*

Covering Logic Expressions

- Logic expressions show up in many situations
- Covering logic expressions is required by the US Federal Aviation Administration for safety critical software
- Logical expressions can come from many sources
 - Decisions in programs
 - FSMs and statecharts
 - Requirements
- Tests are intended to choose some subset of the total # of truth assignments to the expressions

Logic Coverage Criteria Subsumption



Logic Predicates and Clauses

- A *predicate* is an expression that evaluates to a **boolean** value
- Predicates can contain
 - **boolean variables**
 - non-boolean variables that contain $>$, $<$, $==$, $>=$, $<=$, $!=$
 - **boolean function calls**
- Internal structure is created by logical operators
 - \neg – the *negation* operator
 - \wedge – the *and* operator
 - \vee – the *or* operator
 - \rightarrow – the *implication* operator
 - \oplus – the *exclusive or* operator
 - \leftrightarrow – the *equivalence* operator
- A *clause* is a predicate with no logical operators

Examples

- $(a < b) \vee f(z) \wedge D \wedge (m \geq n^*o)$
- Four clauses:
 - $(a < b)$ – relational expression
 - $f(z)$ – boolean-valued function
 - D – boolean variable
 - $(m \geq n^*o)$ – relational expression
- Most predicates have few clauses
- Sources of predicates
 - Decisions in programs
 - Guards in finite state machines
 - Decisions in UML activity graphs
 - Requirements, both formal and informal
 - SQL queries

Testing and Covering Predicates

- We use predicates in testing as follows :
 - Developing a model of the software as one or more predicates
 - Requiring tests to satisfy some combination of clauses
- Abbreviations:
 - P is a set of predicates
 - p is a single predicate in P
 - C is a set of clauses in P
 - C_p is a set of clauses in predicate p
 - c is a single clause in C

Predicate and Clause Coverage

- The first (and simplest) two criteria require that each predicate and each clause be evaluated to both true and false

Predicate Coverage (PC) : For each p in P , TR contains two requirements: p evaluates to true, and p evaluates to false.

a.k.a. “decision coverage” in literature

- When predicates come from conditions on edges, this is equivalent to edge coverage
- PC does not evaluate all the clauses, so ...

Clause Coverage (CC) : For each c in C , TR contains two requirements: c evaluates to true, and c evaluates to false.

a.k.a. “condition coverage” in literature

Predicate Coverage Example

$$((a < b) \vee D) \wedge (m \geq n^*o)$$

predicate coverage

Predicate = true

$$a = 5, b = 10, D = \text{true}, m = 1, n = 1, o = 1$$

$$= (5 < 10) \vee \text{true} \wedge (1 \geq 1^*1)$$

$$= \text{true} \vee \text{true} \wedge \text{TRUE}$$

$$= \text{true}$$

Predicate = false

$$a = 10, b = 5, D = \text{false}, m = 1, n = 1, o = 1$$

$$= (10 < 5) \vee \text{false} \wedge (1 \geq 1^*1)$$

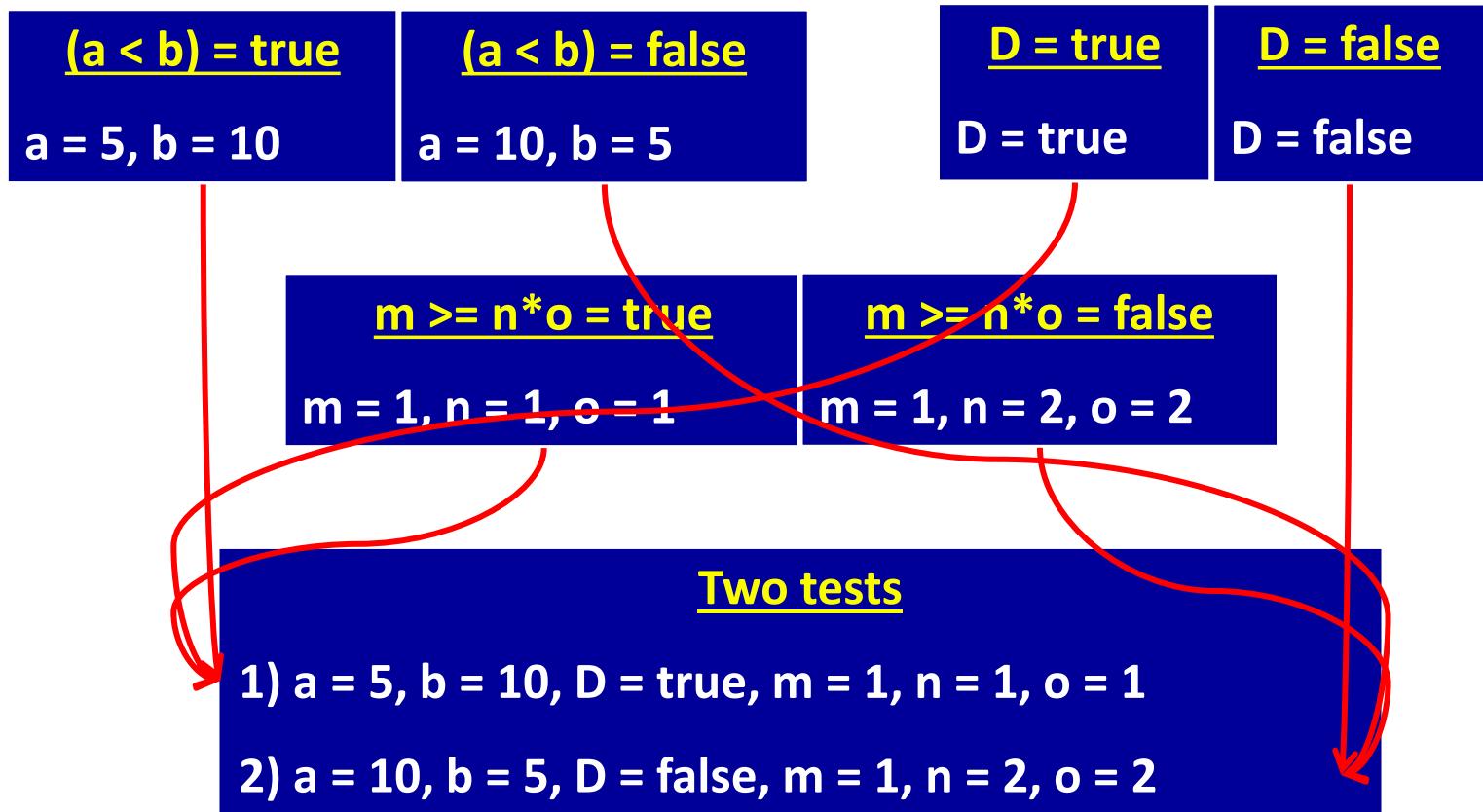
$$= \text{false} \vee \text{false} \wedge \text{TRUE}$$

$$= \text{false}$$

Clause Coverage Example

$$((a < b) \vee D) \wedge (m \geq n^*o)$$

Clause coverage



Problems with PC and CC

- PC does not fully exercise all the clauses, especially in the presence of short circuit evaluation
- CC does not always ensure PC
 - That is, we can satisfy CC without causing the predicate to be both true and false
 - Ex. $x > 3 \rightarrow x > 1$
 - Two test cases { $x=4, x=0$ } satisfy CC but not PC
- Condition/decision coverage is a hybrid metric composed by CC union PC

Combinatorial Coverage

- CoC requires every possible combination
- Sometimes called **Multiple Condition Coverage**

Combinatorial Coverage (CoC) : For each p in P , TR has test requirements for the clauses in C_p to evaluate to each possible combination of truth values.

	$a < b$	D	$m \geq n^*o$	$((a < b) \vee D) \wedge (m \geq n^*o)$
1	T	T	T	T
2	T	T	F	F
3	T	F	T	T
4	T	F	F	F
5	F	T	T	T
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

Combinatorial Coverage

- This is simple, neat, clean, and comprehensive ...
- But quite expensive!
- 2^N tests, where N is the number of clauses
 - Impractical for predicates with more than 3 or 4 clauses
- The literature has lots of suggestions – some confusing
- The general idea is simple:

Test each clause **independently** from the other clauses

- Getting the details right is hard
- What exactly does “independently” mean ?
- The book presents this idea as “***making clauses active***” ...

Active Clauses

- Clause coverage has a weakness
 - The values do **not** always make a difference to a whole predicate
- To really test the results of a clause, the clause should be the **determining factor** in the value of the predicate

Determination :

A clause c_i in predicate p , called the **major clause**,
determines p if and only if the values of the remaining
minor clauses c_j are such that changing c_i changes the
value of p

- This is considered to make the clause c_i **active**

Determining Predicates

$$\underline{P = A \vee B}$$

if $B = \text{true}$, p is always true.

so if $B = \text{false}$, A determines p .

if $A = \text{false}$, B determines p .

$$\underline{P = A \wedge B}$$

if $B = \text{false}$, p is always false.

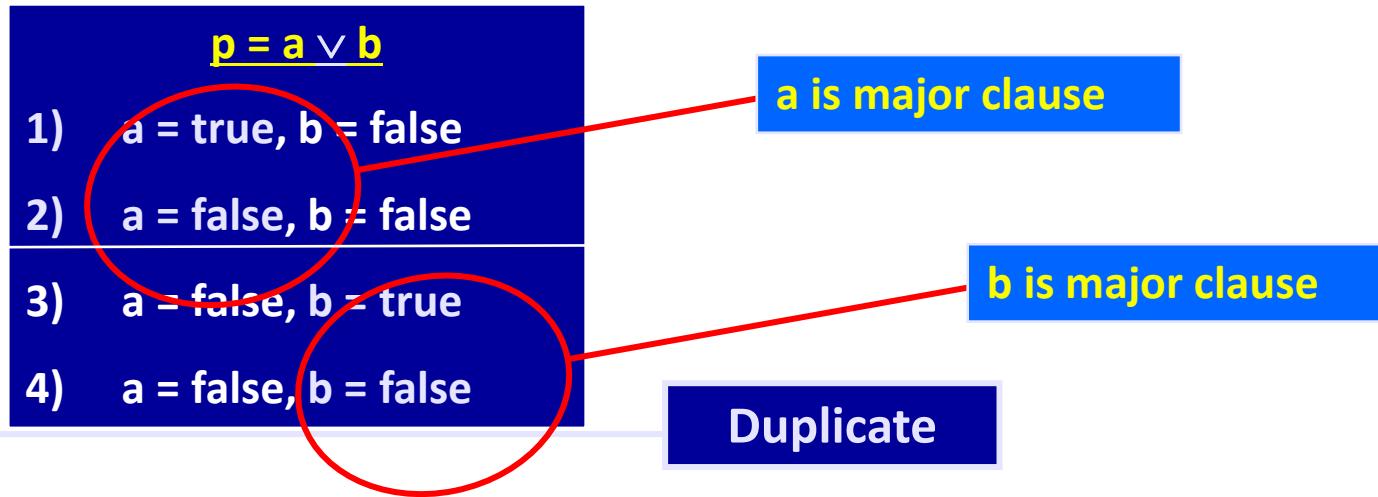
so if $B = \text{true}$, A determines p .

if $A = \text{true}$, B determines p .

- Goal : Find tests for **each** clause when the clause determines the value of the predicate
- This is formalized in **several criteria** that have subtle, but very important, differences

Active Clause Coverage

Active Clause Coverage (ACC) : For each p in P and each major clause c_i in C_p , choose minor clauses c_j , $j \neq i$, so that c_i determines p . TR has two requirements for each c_i : c_i evaluates to true and c_i evaluates to false.



- This is a form of **MCDC**, which is required by the Federal Avionics Administration (FAA) for safety critical software
- Ambiguity : Do the minor clauses have to have the **same values** when the major clause is true and false?

Resolving the Ambiguity

$$p = a \vee (b \wedge c)$$

Major clause : a

a = true, b = false, c = true

a = false, b = false, c = ~~false~~
c = false

Is this allowed ?

- This question caused confusion among testers for years
- Considering this carefully leads to three separate criteria :
 - Minor clauses do need to be the same (RACC)
 - Minor clauses do not need to be the same but force the predicate to become both true and false (CACC)

Restricted Active Clause Coverage

Restricted Active Clause Coverage (RACC) : For each p in P and each major clause c_i in Cp , choose minor clauses c_j , $j \neq i$, so that c_i determines p . TR has two requirements for each c_i : c_i evaluates to true and c_i evaluates to false.

The values chosen for the minor clauses c_j must be the same when c_i is true as when c_i is false, that is, it is required that $c_j(c_i = \text{true}) = c_j(c_i = \text{false})$ for all c_j .

- This has been a common interpretation of MCDC by aviation developers
 - Often called “unique-cause MCDC”
- RACC often leads to infeasible test requirements

Correlated Active Clause Coverage

Correlated Active Clause Coverage (CACC) : For each p in P and each major clause c_i in Cp , choose minor clauses $c_j, j \neq i$, so that c_i determines p . TR has two requirements for each c_i : c_i evaluates to true and c_i evaluates to false.

The values chosen for the minor clauses c_j must cause p to be true for one value of the major clause c_i and false for the other, that is, it is required that $p(c_i = \text{true}) \neq p(c_i = \text{false})$.

- A more recent interpretation
 - Also known as “Masking MCDC”
- Implicitly allows minor clauses to have different values
 - But still the major clause should be **the only clause** that affects the predicate
- Explicitly satisfies (subsumes) predicate coverage

CACC and RACC

	a	b	c	$a \wedge (b \vee c)$
1	a	T	T	T
2	T	T	F	T
3	T	F	T	T
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F

major clause

CACC can be satisfied by choosing
any of rows 1, 2, 3 AND any of rows
5, 6, 7 – a total of nine pairs

	a	b	c	$a \wedge (b \vee c)$
1	a	T	T	T
5	F	T	T	F
2	T	T	F	T
6	F	T	F	F
3	T	F	T	T
7	F	F	T	F

major clause

RACC can only be satisfied by one of
the three pairs above

Note that only **a** affects the predicate
since the value of $(b \vee c)$ does not change

Modified condition/decision coverage (MCDC)

- Standard requirement for safety critical systems such as avionics and automotive (e.g., DO 178B/C, ISO26262)
- Modified condition/decision coverage (MCDC) requires
 - Satisfying CC and DC, and
 - every condition in a decision should be shown to independently affect that decision's outcome
- Example: $C = A \mid\mid B$
 - Which test cases are necessary to satisfy
 - Condition coverage
 - Decision coverage
 - Condition/decision coverage
 - MCDC coverage

	A	B	C
TC1	T	T	T
TC2	T	F	T
TC3	F	T	T
TC4	F	F	F

Minimum Testing to Achieve MCDC [Chilenski and Miller'94]

For $C = A \& B$,

- All conditions (i.e., A and B) should be true so that decision (i.e., C) becomes true
 - 1 test case required
- Each and every input should be exclusively false so that decision becomes false.
 - 2 (or n for n-ary and) test cases required

	A	B	C
TC1	T	T	T
TC2	T	F	F
TC3	F	T	F
TC4	F	F	F

For $C = A \mid\mid B$

- All conditions (i.e., A and B) should be false so that decision (i.e., C) becomes false
 - 1 test case required
- Each and every input should be exclusively true so that decision becomes true.
 - 2 (or n for n-ary or) test cases required

	A	B	C
TC1	T	T	T
TC2	T	F	T
TC3	F	T	T
TC4	F	F	F

A Few Notes for Masking MC/DC

- The masking MC/DC allows more than one condition to change in an independence pair, **as long as** the condition of interest (i.e., major clause) is the only condition that affects the value of the decision outcome.
 - **Masking** refers to the approach where specific conditions can mask the effects of other conditions.
- Example. If (A and B) or (C and D) then X; else Y;
 - The following 2 test inputs show that A can independently affect the outcome of the decision.

A	B	C	D	Outcome
True	True	False	True	True
False	True	True	False	False

<https://www.rapitasystems.com/blog/masking-mcdc-whats-all-about-then#:~:text=Masking%20refers%20to%20the%20approach,value%20of%20the%20decision%20outcome.>

Inactive Clause Coverage

- The active clause coverage criteria ensure that “major” clauses do affect the predicates
- Inactive clause coverage takes the opposite approach – major clauses do NOT affect the predicates

Inactive Clause Coverage (ICC) : For each p in P and each major clause c_i in Cp , choose minor clauses c_j , $j \neq i$, so that c_i does not determine p . TR has four requirements for each c_i :

- (1) c_i evaluates to true with p true
- (2) c_i evaluates to false with p true
- (3) c_i evaluates to true with p false, and
- (4) c_i evaluates to false with p false.

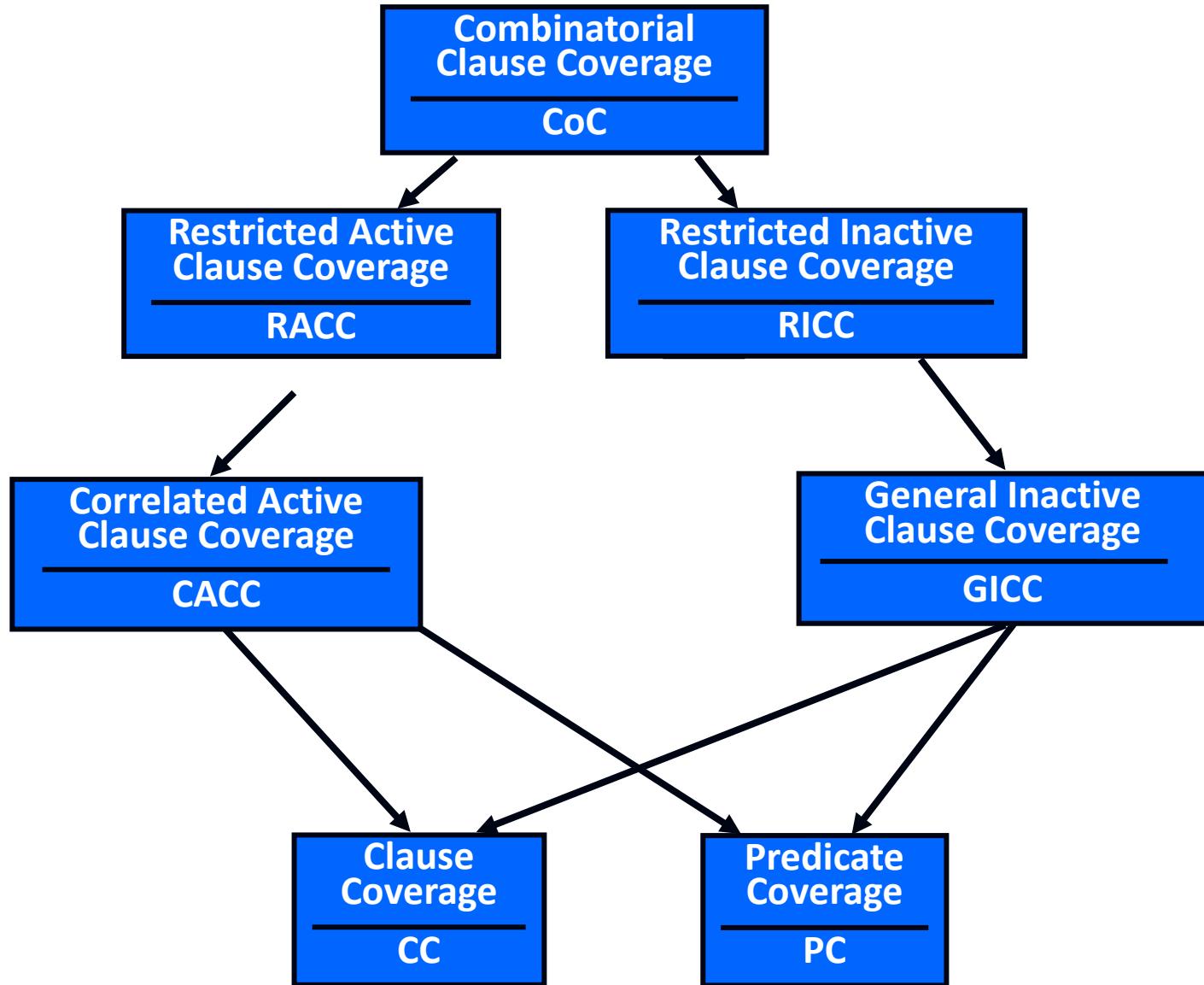
General and Restricted ICC

- Unlike ACC, the notion of correlation is not relevant
 - c_i does not determine p , so cannot correlate with p
- Predicate coverage is always guaranteed

General Inactive Clause Coverage (GICC) : For each p in P and each major clause c_i in C_p , choose minor clauses $c_j, j \neq i$, so that c_i does not determine p . The values chosen for the minor clauses c_j do not need to be the same when c_i is true as when c_i is false, that is, $c_j(c_i = \text{true}) = c_j(c_i = \text{false})$ for all c_j OR $c_j(c_i = \text{true}) \neq c_j(c_i = \text{false})$ for all c_j .

Restricted Inactive Clause Coverage (RICC) : For each p in P and each major clause c_i in C_p , choose minor clauses $c_j, j \neq i$, so that c_i does not determine p . The values chosen for the minor clauses c_j must be the same when c_i is true as when c_i is false, that is, it is required that $c_j(c_i = \text{true}) = c_j(c_i = \text{false})$ for all c_j .

Logic Coverage Criteria Subsumption



Making Clauses Determine a Predicate

- Finding values for minor clauses c_j is easy for simple predicates
- But how to find values for more complicated predicates ?
- Definitional approach:
 - $p_{c=true}$ is predicate p with every occurrence of c replaced by *true*
 - $p_{c=false}$ is predicate p with every occurrence of c replaced by *false*
- To find values for the minor clauses, connect $p_{c=true}$ and $p_{c=false}$ with exclusive *OR*

$$p_c = p_{c=true} \oplus p_{c=false}$$

- After solving, p_c describes exactly the values needed for c to determine p
- Note that we have to calculate $\neg p_c \wedge p=\text{true}$ and/or $\neg p_c \wedge p=\text{false}$ to get values for minor clauses for Inactive Coverage Criteria

Examples

$$\underline{p = a \vee b}$$

$$\begin{aligned} p_a &= p_{a=\text{true}} \oplus p_{a=\text{false}} \\ &= (\text{true} \vee b) \text{ XOR } (\text{false} \vee b) \\ &= \text{true XOR } b \\ &= \neg b \end{aligned}$$

$$\underline{p = a \wedge b}$$

$$\begin{aligned} p_a &= p_{a=\text{true}} \oplus p_{a=\text{false}} \\ &= (\text{true} \wedge b) \oplus (\text{false} \wedge b) \\ &= b \oplus \text{false} \\ &= b \end{aligned}$$

$$\underline{p = a \vee (b \wedge c)}$$

$$\begin{aligned} p_a &= p_{a=\text{true}} \oplus p_{a=\text{false}} \\ &= (\text{true} \vee (b \wedge c)) \oplus (\text{false} \vee (b \wedge c)) \\ &= \text{true} \oplus (b \wedge c) \\ &= \neg (b \wedge c) \\ &= \neg b \vee \neg c \end{aligned}$$

- “*NOT b* \vee *NOT c*” means either *b* or *c* can be false
- RACC requires the same choice for both values of *a*, CACC does not

A More Subtle Example

$$p = (a \wedge b) \vee (a \wedge \neg b)$$

$$\begin{aligned} p_a &= p_{a=\text{true}} \oplus p_{a=\text{false}} \\ &= ((\text{true} \wedge b) \vee (\text{true} \wedge \neg b)) \oplus ((\text{false} \wedge b) \vee (\text{false} \wedge \neg b)) \\ &= (b \vee \neg b) \oplus \text{false} \\ &= \text{true} \oplus \text{false} \\ &= \text{true} \end{aligned}$$

$$p = (a \wedge b) \vee (a \wedge \neg b)$$

$$\begin{aligned} p_b &= p_{b=\text{true}} \oplus p_{b=\text{false}} \\ &= ((a \wedge \text{true}) \vee (a \wedge \neg \text{true})) \oplus ((a \wedge \text{false}) \vee (a \wedge \neg \text{false})) \\ &= (a \vee \text{false}) \oplus (\text{false} \vee a) \\ &= a \oplus a \\ &= \text{false} \end{aligned}$$

- *a* always determines the value of this predicate
- *b* never determines the value – *b* is irrelevant !

Infeasible Test Requirements

- Consider the predicate:

$$(a > b \wedge b > c) \vee c > a$$

- $(a > b) = \text{true}$, $(b > c) = \text{true}$, $(c > a) = \text{true}$ is **infeasible**
- As with graph-based criteria, infeasible test requirements have to be recognized and ignored
- Recognizing infeasible test requirements is hard, and in general, **undecidable**

Example

$$p = a \wedge (\neg b \vee c)$$

	a	b	c	p	p_a	p_b	p_c
1	T	T	T	T	T	F	T
2	T	T	F	F	F	T	T
3	T	F	T	T	T	F	F
4	T	F	F	T	T	T	F
5	F	T	T	F	T	F	F
6	F	T	F	F	F	F	F
7	F	F	T	F	T	F	F
8	F	F	F	F	T	F	F

- Conditions under which each of the clauses determines p

- p_a : $(\neg b \vee c)$
- p_b : $a \wedge \neg c$
- p_c : $a \wedge b$

- All pairs of rows satisfying CACC
 - a: $\{1,3,4\} \times \{5,7,8\}$, b: $\{(2,4)\}$, c: $\{(1,2)\}$
- All pairs of rows satisfying RACC
 - a: $\{(1,5),(3,7),(4,8)\}$
 - Same as CACC pairs for b, c
- GICC
 - a: $\{(2,6)\}$ for $p=F$, no feasible pair for $p=T$
 - b: $\{5,6\} \times \{7,8\}$ for $p=F$, $\{(1,3)\}$ for $p=T$
 - c: $\{5,7\} \times \{6,8\}$ for $p=F$, $\{(3,4)\}$ for $p=T$
- RICC
 - a: same as GICC
 - b: $\{(5,7),(6,8)\}$ for $p=F$, $\{(1,3)\}$ for $p=T$
 - c: $\{(5,6),(7,8)\}$ for $p=F$, $\{(3,4)\}$ for $p=T$

Logic Coverage Summary

- Predicates are often **very simple**—in practice, most have less than 3 clauses
 - In fact, most predicates only have one clause !
 - With only clause, PC is enough
 - With 2 or 3 clauses, CoC is practical
 - Advantages of ACC and ICC criteria significant for large predicates
 - CoC is impractical for predicates with many clauses
- Control software often has many complicated predicates, with lots of clauses

Bug Observability/Detection Model: Reachability, Infection, Propagation, and Revelation (RIPR)

- Terminology

- **Fault**: static defect in a program text (a.k.a a bug)
- **Error**: dynamic (intermediate) behavior that deviates from its (internal) intended goal
 - A fault causes an error (i.e. error is a symptom of fault)
- **Failure**: dynamic behavior which violates a ultimate goal of a target program
 - Not every error leads to failure due to error masking or fault tolerance

- Graph coverage

- Test requirement satisfaction == **Reachability**

- the fault in the code has to be reached

- Logic coverage

- Test requirement satisfaction

- == **Reachability +Infection**

- the fault has to put the program into an error state.

- Note that a program is in an error state does not mean that it will always produce the failure

- Mutation coverage

- Test requirement satisfaction

- == **Reachability +Infection + Propagation**

- the program needs to exhibit incorrect outputs

- Furthermore, test oracle plays critical role to reveal failure of a target program (**Revelation**)

