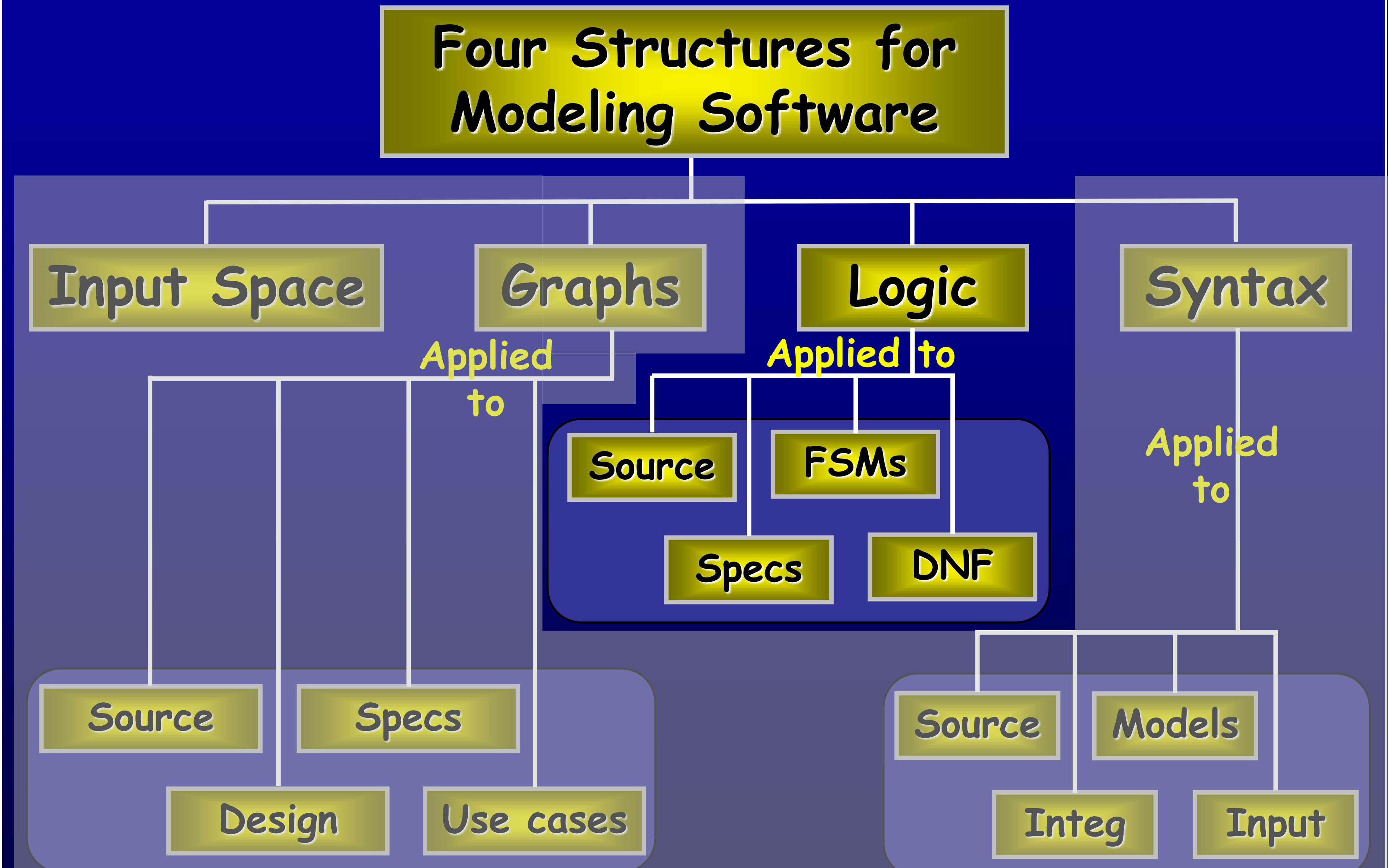
Introduction to Software Testing
Chapter 8.1
Logic Coverage

Paul Ammann & Jeff Offutt

http://www.cs.gmu.edu/~offutt/softwaretest/

Ch. 8 : Logic Coverage



Semantic Logic Criteria (8.1)

- Logic expressions show up in many situations
- Covering logic expressions is required by the US Federal Aviation Administration (FAA) 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 number of truth assignments to the expressions

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

Example and Facts

- $(a < b) \lor f(z) \land D \land (m >= n*o)$ has four clauses:
  - $(a < b)$ – relational expression
  - $f(z)$ – boolean-valued function
  - D – boolean variable
  - $(m >= n*o)$ – relational expression
- Most predicates have few clauses
  - 88.5% have 1 clause
  - 9.5% have 2 clauses
  - 1.35% have 3 clauses
  - Only 0.65% have 4 or more !

from a study of 63 open source programs, >400,000 predicates

- Sources of predicates
  - Decisions in programs
  - Guards in finite state machines
  - Decisions in UML activity graphs
  - Requirements, both formal and informal
  - SQL queries

Logic Coverage Criteria (8.1.1)

- 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 the set of predicates
  - p is a single predicate in P
  - C is the set of clauses in P
  - C_p is the 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.

- 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.

Predicate Coverage Example

$((a < b) \lor D) \land (m >= n*o)$

predicate coverage

Predicate = true

a = 5, b = 10, D = true, m = 1, n = 1, o = 1

$= (5 < 10) \lor true \land (1 >= 1*1)$

$= true \lor true \land TRUE$

$= true$

Predicate = false

a = 10, b = 5, D = false, m = 1, n = 1, o = 1

$= (10 < 5) \lor false \land (1 >= 1*1)$

$= false \lor false \land TRUE$

$= false$

Clause Coverage Example

$((a < b) \vee D) \wedge (m >= n*o)$

Clause coverage

(a < b) = true	(a < b) = false	D = true	D = false
a = 5, b = 10	a = 10, b = 5	D = true	D = false

m >= n*o = true	m >= n*o = false
m = 1, n = 1, o = 1	m = 1, n = 2, o = 2

true cases

false cases

Two tests

1) a = 5, b = 10, D = true, m = 1, n = 1, o = 1
2) a = 10, b = 5, D = false, m = 1, n = 2, o = 2

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
  - This is definitely not what we want !

	a	b	$a \vee b$
1	T	T	T
2	T	F	T
3	F	T	T
4	F	F	F

- The simplest solution is to test all combinations …

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 Cp to evaluate to each possible combination of truth values.

	a < b	D	m >= n*o	((a < b) ∨ D) ∧ (m >= 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

- What exactly does "independently" mean ?
- The book presents this idea as "*making clauses active*" …

Active Clauses (8.1.2)

- Clause coverage has a weakness : The values do not always make a difference
- 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 active*

Determining Predicates

$P = A \vee B$

if B = *true*, p is always true.

so if B = *false*, A determines p.

if A = *false*, B determines p.

$P = A \wedge B$

if B = *false*, p is always false.

so if B = *true*, A determines p.

if A = *true*, B determines p.

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

Active Clause Coverage

<u>Active Clause Coverage (ACC)</u> : For each *p* in *P* and each major clause c_i in *Cp*, choose minor clauses c_j, j != 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.

<u>p = a ∨ b</u>

1) a = true, b = false
2) a = false, b = false
3) a = false, b = true
4) ~~a = false, b = false~~

a is major clause

b is major clause

Duplicate

- This is a form of MCDC, which is required by the 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

Is this allowed ?

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

General Active Clause Coverage

General Active Clause Coverage (GACC) : For each *p* in *P* and each major clause c_i in *Cp*, choose minor clauses c_j, j != i, so that *ci* determines *p*. TR has two requirements for each *ci* : c_i evaluates to true and c_i evaluates to false. 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, $c_j(c_i = true) = c_j(c_i = false)$ for all c_j OR $c_j(c_i = true) \,!\!= c_j(c_i = false)$ for all c_j.

- This is complicated !
- It is possible to satisfy GACC without satisfying predicate coverage
- We really want to cause predicates to be both true and false

GACC Problem

- Unfortunately, it turns out that General Active Clause Coverage does not subsume Predicate Coverage
- Consider the following example:
- $p = a \leftrightarrow b$
- Clause a determines p for any assignment of truth values to b
  - So, when a is true, we choose b to be true (TT)
  - When a is false, we choose b to be false as well (FF)

a	b	$a \leftrightarrow b$
T	T	T
T	F	F
F	T	F
F	F	T

- We make the same selections for clause b
- We end up with only two test inputs: {TT, FF}
  - p evaluates to true for both of these cases
  - So Predicate Coverage is not achieved
- GACC also does not subsume PC when an exclusive or operator is used

Restricted Active Clause Coverage

<u>Restricted Active Clause Coverage (RACC)</u> : For each p in P and each major clause c_i in Cp, choose minor clauses c_j, j != 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 <u>must be the same</u> when c_i is true as when c_i is false, that is, it is required that $c_j(c_i = true) = c_j(c_i = false)$ for all c_j.

- This has been a common interpretation by aviation developers
- RACC subsumes Predicate Coverage
- RACC often leads to infeasible test requirements
- There is no logical reason for such a restriction

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 != 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 $p(c_i = true)$!= $p(c_i = false)$.

- A more recent interpretation
- Implicitly allows minor clauses to have different values
- Explicitly satisfies (subsumes) predicate coverage
- RACC subsumes CACC (if no infeasible requirements exist)
  - Choosing the same value for minor clauses forces the predicate to be both true and false

CACC and RACC

	a	b	c	a ∧ (b ∨ c)
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	F
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

major clause

P_a : b=true or c = true

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 ∧ (b ∨ c)
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	F
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

RACC can only be satisfied by row pairs (1, 5), (2, 6), or (3, 7)

Only three pairs

CACC versus RACC

It turns out that some logical expressions can be completely satisfied under CACC, but have infeasible test requirements under RACC. These expressions are a little subtle and only exist if dependency relationships exist among the clauses, that is, some combinations of values for the clauses are prohibited. Since this often happens in real programs, because program variables frequently depend upon one another, we introduce the following example.

Consider a system with a valve that might be either open or closed, and several modes, two of which are "Operational" and "Standby." Assume the following two constraints:

1. The valve must be open in "*Operational*" and closed in all other modes.
2. The mode cannot be both "*Operational*" and "*Standby*" at the same time.

CACC versus RACC

This leads to the following clause definitions:

$a =$ *"The valve is closed"*

$b =$ *"The system status is Operational"*

$c =$ *"The system status is Standby"*

Suppose that a certain action can be taken only if the valve is closed and the system status is either in *Operational* or *Standby*. That is:

$$
\begin{aligned}
p &= \textit{valve is closed AND (system status is Operational OR} \\
&\quad \textit{system status is Standby)} \\
&= a \wedge (b \vee c)
\end{aligned}
$$

This is exactly the predicate that was analyzed above. The constraints above can be formalized as:

$$1 \neg a \leftrightarrow b$$

$$2 \neg (b \wedge c)$$

CACC versus RACC

These constraints limit the feasible values in the truth table. As a reminder, the complete truth table for this predicate is:

	a	b	c	$(a \wedge (b \vee c))$	
1	T	T	T	T	violates constraints 1 & 2
2	T	T	F	T	violates constraint 1
3	T	F	T	T	
4	T	F	F	F	
5	F	T	T	F	violates constraint 2
6	F	T	F	F	
7	F	F	T	F	violates constraint 1
8	F	F	F	F	violates constraint 1

a	b	~a ↔ b
T	T	F
T	F	T
F	T	T
F	F	F

1 $\neg a \leftrightarrow b$

2 $\neg(b \wedge c)$

major clause: a

P_a : b=true or c = true

CACC can be satisfied by choosing any of rows {1, 2, 3} AND {5, 6, 7 }

RACC can only be satisfied by row pairs (1, 5) or (2, 6), or (3, 7)

RACC is infeasible for a

Due to constraints 1 & 2, the only feasible rows are 3, 4, and 6

Inactive Clause Coverage (8.1.3)

- 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

<u>Inactive Clause Coverage (ICC)</u> : For each *p* in *P* and each major clause c_i in *Cp*, choose minor clauses c_j, *j != i*, so that c_i <u>does not</u> determine *p*. TR has <u>four</u> 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.

ICC Example

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

Major clause : a

$P_a : b \wedge c = true$

Minor clauses:

b = true, c = true

Four requirements:

1) a = true, p = true

2) a= true, p = false

3) a = false, p = true

4) a = false, p = false

Test cases:

abc = {TTT, FTT}

Requirements 2 & 4 are infeasible in this predicate

General and Restricted ICC

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

<u>General Inactive Clause Coverage (GICC)</u> : For each *p* in *P* and each major clause c_i in *Cp*, choose minor clauses c_j , *j != i*, so that c_i <u>does not</u> determine *p*. The values chosen for the minor clauses c_j <u>do not</u> need to be the same when c_i is true as when c_i is false, that is, $c_j(c_i = true) = c_j(c_i = false)$ for all c_j OR $c_j(c_i = true) != c_j(c_i = false)$ for all c_j.

<u>Restricted Inactive Clause Coverage (RICC)</u> : For each *p* in *P* and each major clause c_i in *Cp*, choose minor clauses c_j, *j != i*, so that c_i <u>does not</u> determine *p*. The values chosen for the minor clauses *cj* <u>must be</u> the same when c_i is true as when c_i is false, that is, it is required that $c_j(c_i = true) = c_j(c_i = false)$ for all c_j.

Infeasibility & Subsumption (8.1.4)

- Consider the predicate:

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

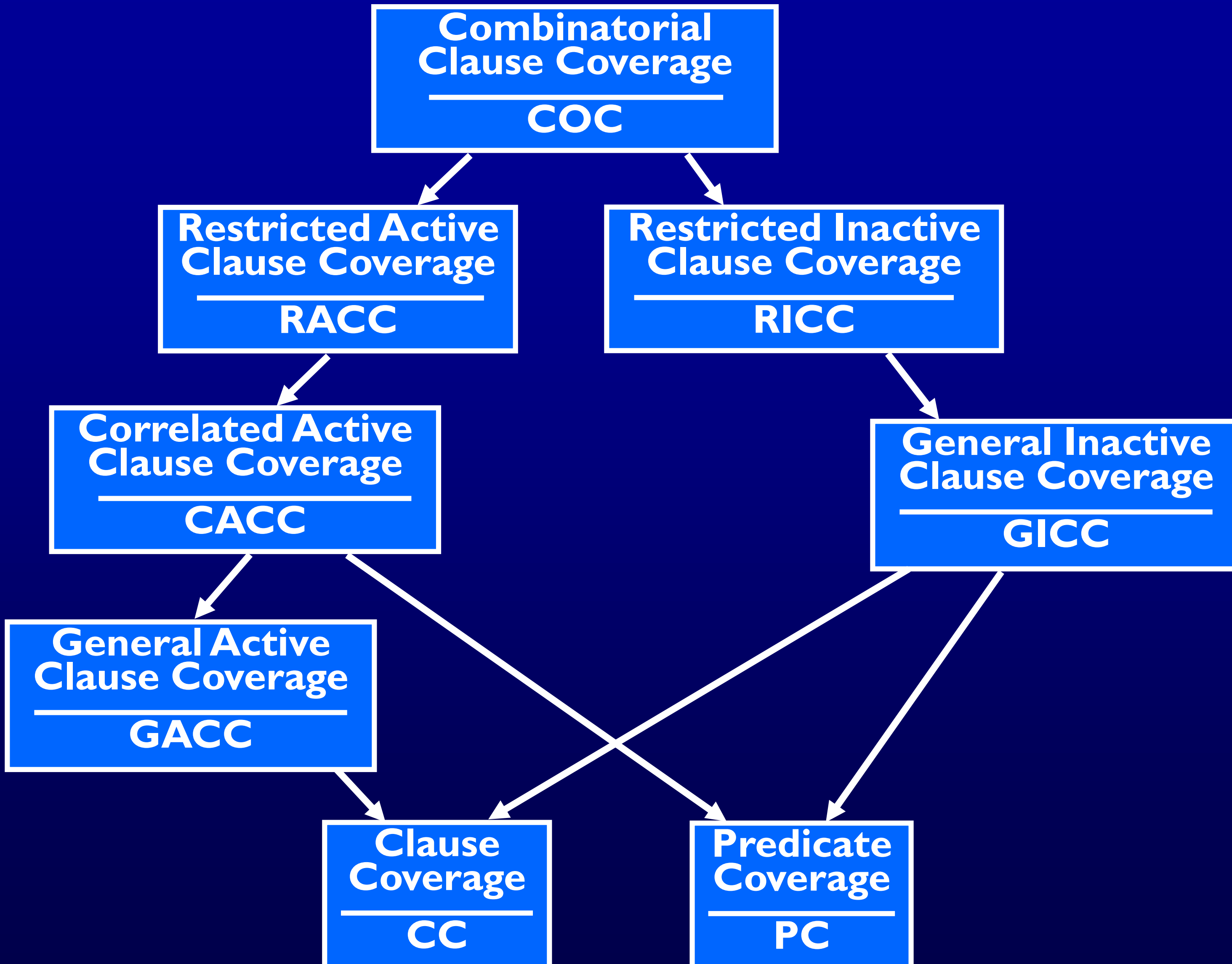
- $(a > b) = true$, $(b > c) = true$, $(c > a) = true$ is infeasible

- As with graph-based criteria, infeasible test requirements have to be recognized and ignored

- Recognizing infeasible test requirements is hard, and in general, undecidable

- Software testing is inexact – engineering, not science

Logic Criteria Subsumption



Making Clauses Determine a Predicate

(8.1.5)

- 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

XOR Identity Rules

Exclusive-OR (*xor*, $\oplus$) means both cannot be true

That is, A *xor* B means

"*A or B is true, but not both*"

$$A \oplus B = (A \wedge \neg B) \vee (\neg A \wedge B)$$

$p = A \oplus A \wedge b$
$= A \wedge \neg b$

$p = A \oplus A \vee b$
$= \neg A \wedge b$

Inputs		Output X = A ⊕ B
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

with fewer symbols …

p = A xor (A and b)
= A and !b

p = A xor (A or b)
= !A and b

Determinantion Examples

$p = a \vee b$

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

$p = a \wedge b$

$$
\begin{aligned}
P_a &= P_{a=true} \oplus P_{a=false} \\
&= (true \wedge b) \oplus (false \wedge b) \\
&= b \oplus false \\
&= b
\end{aligned}
$$

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

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

Inputs		Output
A	B	$X = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

- **"*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

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

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

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

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

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

Repeated Variables

- The definitions in this chapter yield the same tests no matter how the predicate is expressed
- $(a \vee b) \wedge (c \vee b) == (a \wedge c) \vee b$
- Repeated variables are counted once
- $(a \wedge b) \vee (b \wedge c) \vee (a \wedge c)$
  - Only 3 variables, not 6
  - Only has 8 possible tests, not 64 (2^6)
- Use the simplest form of the predicate

Tabular Method for Determination

- The math sometimes gets complicated
- A truth table can sometimes be simpler
- Example

	a	b	c	$a \wedge (b \vee c)$	p_a	p_b	p_c
1	T	T	T	T			
2	T	T	F	T			
3	T	F	T	T			
4	T	F	F	F			
5	F	T	T	F			
6	F	T	F	F			
7	F	F	T	F			
8	F	F	F	F			

In sum, three separate pairs of rows can cause *a* to determine the value of *p*, and only one pair each for *b* and *c*

Finding Satisfying Values

- The final step in applying the logic coverage criteria is to choose values that satisfy the criteria
- Example:

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

- Predicate Coverage:

$$TR_{PC} = \{p = true, p = false\}$$

and they can be satisfied with the following values for the clauses:

	a	b	c
$p = true$	t	t	t
$p = false$	t	t	f

Finding Satisfying Values

- Example:

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

- Predicate Coverage:

	a	b	c
$p = true$	t	t	t
$p = false$	t	t	f

To run the test cases, we need to refine these truth assignments to create values for clauses *a*, *b*, and *c*. Suppose that clauses *a*, *b*, and *c* were defined in terms of Java program variables as follows:

a	`x < y`, a relational expression for program variables `x` and `y`
b	`done`, a primitive boolean value
c	`list.contains(str)`, for `List` and `String` objects

Thus, the complete expanded predicate is actually:

$$p = (x < y \vee done) \wedge list.contains(str)$$

Finding Satisfying Values

- Example:

$$p = (a \lor b) \land c$$

- Predicate Coverage:

	a	b	c
$p = true$	t	t	t
$p = false$	t	t	f

$$p = (x < y \lor done) \land list.contains(str)$$

Then the following values for the program variables satisfy the test requirements for Predicate Coverage.

	a		b	c	
$p = true$	x=3	y=5	done = true	list=["Rat", "Cat", "Dog"]	str = "Cat"
$p = false$	x=0	y=7	done = true	list=["Red", "White"]	str = "Blue"

Further Examples

- Example:

$$p = (a \lor b) \land c$$

- Clause Coverage:

$$TR_{CC} = \{a = true, a = false, b = true, b = false, c = true, c = false\}$$

and they can be satisfied with the following values for the clauses (blank cells represent "don't-care" values):

	a	b	c
$a = true$	t		
$a = false$	f		
$b = true$		t	
$b = false$		f	
$c = true$			t
$c = false$			f

Refining the truth assignments to create values for program variables *x*, *y*, *done*, *list*, and *str* is left as an exercise for the reader.

Further Examples

- Example:

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

- Combinatorial Coverage:
  - Requiring all combinations of values for the clauses

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

Further Examples

- Example:

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

p_a	$\neg b \wedge c$
p_b	$\neg a \wedge c$
p_c	$a \vee b$

- General Active Clause Coverage:

$TR_{GACC} = \{(a = true \wedge p_a,\ a = false \wedge p_a), (b = true \wedge p_b,\ b = false \wedge p_b), (c = true \wedge p_c,\ c = false \wedge p_c)\}$

	a	b	c	p
$a = true \wedge p_a$	**T**	f	t	t
$a = false \wedge p_a$	**F**	f	t	f
$b = true \wedge p_b$	f	**T**	t	t
$b = false \wedge p_b$	f	**F**	t	f
$c = true \wedge p_c$	t	f	**T**	t
$c = false \wedge p_c$	f	t	**F**	f

Note the duplication; the first and fifth rows are identical, and the second and fourth are identical.

Thus, only four tests are needed to satisfy GACC.

Further Examples

- Example:

$$p = (a \lor b) \land c$$

p_a	$\neg b \land c$
p_b	$\neg a \land c$
p_c	$a \lor b$

- General Active Clause Coverage:

$TR_{GACC} = \{(a = true \land p_a, a = false \land p_a), (b = true \land p_b, b = false \land p_b), (c = true \land p_c, c = false \land p_c)\}$

- A different way of looking at GACC considers all of the possible pairs of test inputs for each pair of test requirements
  - Only one pair (3, 7) for a
  - Only one pair (5, 7) for b
  - 9 pairs satisfy the GACC test requirements for c

$\{(1,2), (1,4), (1,6), (3,2), (3,4), (3,6), (5,2), (5,4), (5,6)\}$

	a	b	c	$(a \lor b) \land c$
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

Further Examples

- Example:

$$p = (a \lor b) \land c$$

p_a	$\neg b \land c$
p_b	$\neg a \land c$
p_c	$a \lor b$

- Correlated Active Clause Coverage:
- A careful examination of the pairs of test cases for GACC reveals that p takes on both truth values in each pair
- Hence, GACC and CACC are the same for predicate p
- However in some cases, a test pair that satisfies GACC with respect to a clause c turns out not to satisfy CACC with respect to c (refer to the exercises)

	a	b	c	$(a \lor b) \land c$
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

Further Examples

- Example:

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

p_a	$\neg b \wedge c$
p_b	$\neg a \wedge c$
p_c	$a \vee b$

- Restricted Active Clause Coverage:

- Only one pair (3, 7) for a
- Only one pair (5, 7) for b
- 3 pairs satisfy the RACC for c:
  - {(1, 2), (3, 4), (5, 6)}

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

GACC or RACC or CACC?

- GACC does not require that Predicate Coverage (PC) be satisfied
  - When the predicates are very small (one or two terms), it is easy to find examples where GACC is satisfied but PC is not
  - For predicates with three or more terms, it is likely that GACC tests will be the same as CACC (and so satisfies PC)
- The restrictive nature of RACC can sometimes make it hard to satisfy the criterion
- Additionally, we have no evidence that RACC gives more or better tests
- Wise readers will by now realize that CACC is often the most practical flavor of ACC

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

Exercise

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

- Answer the following questions for the above predicate:
- (a) List the clauses that go with predicate p.
- (b) Compute (and simplify) the conditions under which each clause determines p.
- (c) Write the complete truth table for the predicate. Label your rows starting from 1.
- (d) List all pairs of rows from your table that satisfy (GACC) with respect to each clause.
- (e) List all pairs of rows from your table that satisfy (CACC) with respect to each clause.
- (f) List all pairs of rows from your table that satisfy (RACC) with respect to each clause.
- (g) List all 4-tuples of rows from your table that satisfy (GICC) with respect to each clause. List any infeasible GICC test requirements.
- (h) List all 4-tuples of rows from your table that satisfy (RICC) with respect to each clause. List any infeasible RICC test requirements.

Exercise

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

(a) *Clauses are* a, b, c.

(b)
$$p_a = T$$
$$p_b = c$$
$$p_c = b$$

(c) *Note: Blank cells represent values of 'F'.*

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

Exercise

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

(d) *GACC pairs for clause* a *are:* $\{1,2,3,4\} \times \{5,6,7,8\}$.
GACC pairs for clause b *are:* $\{1,5\} \times \{3,7\}$.
GACC pairs for clause c *are:* $\{1,5\} \times \{2,6\}$.

(e) *CACC pairs for clause* a *are:* $(1,5) \cup \{2,3,4\} \times \{6,7,8\}$.
CACC pairs for clause b *are:* $(1,3), (5,7)$ *for clause* b.
CACC pairs for clause c *are:* $(1,2), (5,6)$ *for clause* c.

(f) *RACC pairs for clause* a *are:* $(1,5), (2,6), (3,7), (4,8)$.
RACC pairs for clauses b *and* c *are the same as CACC pairs.*

(g) *There are no GICC tuples for clause* a.
GICC tuples for b *are:* $(2,4)$ *for* $p = F$; $(6,8)$ *for* $p = T$.
GICC tuples for c *are:* $(3,4)$ *for* $p = F$; $(7,8)$ *for* $p = T$.

(h) *RICC tuples for clauses* a, b, *and* c *are the same as GICC tuples.*