CS 5154: Software Testing

Testing with Determination

Instructor: Owolabi Legunsen

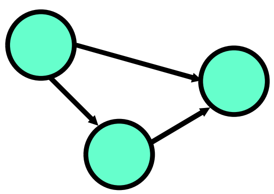
Fall 2021

Recall the four software models in this course

Input Domains

A: {0, 1, >1}
B: {600, 700, 800}
C: {cs, ece, is, sds}

Graphs



Logic Expressions

```
(!x | !y) & a & b
```

Syntax

```
if (x > y)
    z = x - y;
else
    z = 2 * x;
```

We need criteria that are not as costly as CoC

- The general idea is quite simple:

Test each clause independently from the other clauses

- But, getting the details right is hard
  - e.g., what exactly does "independently" mean ?

- The book presents this idea as "*making clauses active*" …

Active Clauses

- A weakness of Clause Coverage: values do not always make a difference
- *Values ((5 < 10) ∨ true) ∧ (1 >= 1*1)* for ((a < b) ∨ D) ∧ (m >= n*o)
    - Only the last clause counts!
- To really test the results of a clause, the clause should be the determining factor in what the predicate evaluates to

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

- Making c_i determine p is said to make the clause active

CS 5154: Software Testing

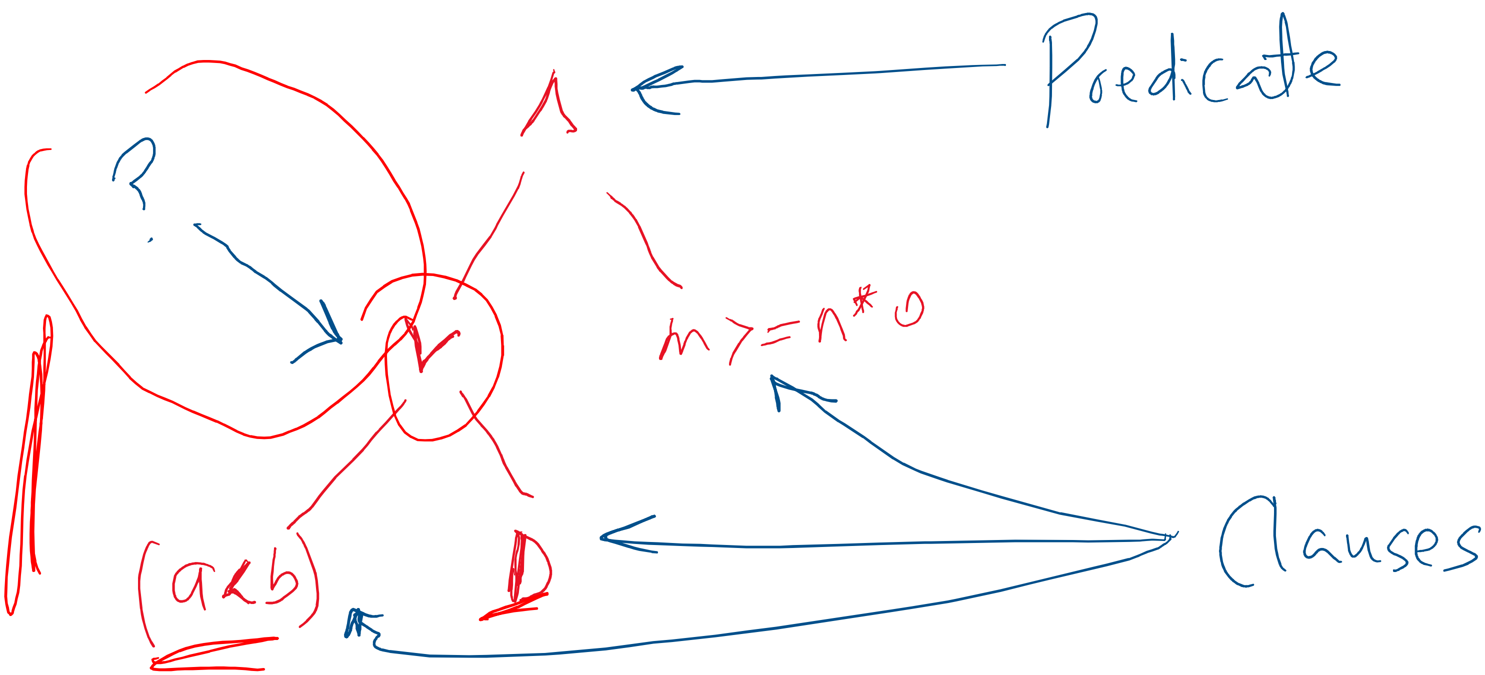
Testing with Determination (Active Clause Criteria)

Instructor: Owolabi Legunsen

Fall 2021

Recall: predicates and clauses

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



Why do we care about clause coverage?

```
int stringFactor(String i, int n) {
  if (i != null || n !=0)
   return i.length()/n;
  else
   return -1;
}
// Tests: ("happy", 2), (null, 0)
```

"happy", 0

F F
F T
T F
T T

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

- Making c_i determine p is said to make the clause active
- Condition under which c_i determines p

$\forall c_j \in C_p \setminus c_i \;\; \exists$ assignment (c_j) s.t. $p(c_i = true) \neq p(c_i = false)$

where a ssignment (c_j) is $c_j = true$ or $c_j = false$

Active clause criteria

The essence of testing with determination

1. Pick one clause in predicate p to be the major clause c_i
2. Find conditions under which c_i determines p
3. Find a test that makes c_i true and a test that makes c_i false
4. Repeat steps 1 to 3 for all other clauses in p
5. Eliminate redundant tests

Examples: 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*.

	a	b	a ∨ b
1	T	T	T
2	T	F	T
3	F	T	T
4	F	F	F

	a	b	a ∧ b
1	T	T	T
2	T	F	F
3	F	T	F
4	F	F	F

More examples: determining predicates

P = A ⊕ B

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

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

so, *A* determines *p for any B*.

P = A ↔ B

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

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

so, *A* determines *p for any B*.

	a	b	a ⊕ b
1	T	T	F
2	T	F	T
3	F	T	T
4	F	F	F

	a	b	a ↔ b
1	T	T	T
2	T	F	F
3	F	T	F
4	F	F	T

Testing with determination ☺

- Goal : Find tests for each clause when that clause determines the value of the predicate

- This goal is formalized in a family of criteria that have subtle, but very important, differences

Active Clause Coverage

- Step 1: 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.

<u>Active Clause Coverage (ACC)</u> : TR has two requirements for each c_i : c_i evaluates to true and c_i evaluates to false.

- ACC is a form of Multiple Condition Decision Coverage (MCDC)
- MCDC is required by the FAA for safety-critical software

Example on Active Clause Coverage

$p = a \vee b$

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

A formulaic way of determining predicates

- Finding values for minor clauses c_j is easy for simple predicates
- How to find values for more complicated predicates?
- We need some “formula” that is easy to apply

A definitional way: when does c determine p?

- Let $\boldsymbol{p_{c=true}}$ be predicate p with every occurrence of c replaced by ***true***

- Let $\boldsymbol{p_{c=false}}$ be predicate p with every occurrence of c replaced by ***false***

- To find values for the minor clauses, connect $\boldsymbol{p_{c=true}}$ and $\boldsymbol{p_{c=false}}$ with *XOR*

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

- After solving, $\boldsymbol{p_c}$ describes exactly the values needed for $\boldsymbol{c}$ to determine $\boldsymbol{p}$

An example using the definitional way

- Let $p = a \vee (b \wedge c)$. What values of b and c will cause a to determine p?

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

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

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

- "*!b* $\vee$ *!c*" means a determines p when either *b* or *c* is false

Exercise 1: using the definitional way

- Let $p = a \vee b$. What values of b will cause a to determine p?

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

$$p_a = p_{a=true} \oplus p_{a=false}$$
$$= (true \vee b) \oplus (false \vee b)$$
$$= true \oplus b$$
$$= !b$$

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

- "*!b*" means a determines p when *b* is false
- We obtained the same result from reasoning about the truth table

Exercise 2: using the definitional way

- Let $p = a \leftrightarrow b$. What values of b will cause a to determine p?

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

$$p_a = p_{a=true} \oplus p_{a=false}$$
$$= (true \leftrightarrow b) \oplus (false \leftrightarrow b)$$
$$= b \oplus !b$$
$$= true$$

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

- "*true*" means that a always determines p
- We obtained the same result from reasoning about the truth table

Is there a problem with Active Clause Coverage?

- Step 1: 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.

<u>Active Clause Coverage (ACC)</u> : TR has two requirements for each c_i : c_i evaluates to true and c_i evaluates to false.

- Ambiguity : Must minor clauses have the same values when the major clause is true and when the major clause is false?

Illustrating the ambiguity in ACC

- Recall: a determines p when "$!b \vee !c$", i.e., when either *b* or *c* is false

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

Major clause : a

a = true, b = false, c = true

a = false, b = false, c = false

Is this allowed ?

Three options for resolving ACC ambiguity

- Minor clauses do not need to be the same

- Minor clauses must be the same

- Minor clauses allow the predicate to become both true and false

Option 1: minor clauses don’t need to be the same

- Step 1: For each p in P and each major clause c_i in C_p, choose minor clauses c_j, j != i, so that c_i determines p.

- Step 2 (ACC): TR has two requirements for each c_i : c_i evaluates to true and c_i evaluates to false.

General Active Clause Coverage (GACC) : 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.

Problem: GACC doesn't subsume Predicate Coverage

Major clause : a

	a	b	a ↔ b
1	T	T	T
2	T	F	F
3	F	T	F
4	F	F	T

a b

Option 2: minor clauses do need to be the same

- Step 1: For each p in P and each major clause c_i in C_p, choose minor clauses c_j, j != i, so that c_i determines p.

- Step 2 (ACC): TR has two requirements for each c_i : c_i evaluates to true and c_i evaluates to false.

Restricted Active Clause Coverage (RACC) : 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 $c_j(c_i = true) = c_j(c_i = false)$ for all c_j.

Exercise 3: using the definitional way

- Let $p = a \wedge (b \vee c)$. What values of b and c will cause a to determine p?

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

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

- "$b \vee c$" means a determines p when either b or c is true

Example on Restricted Active Clause Coverage

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

Major clause : a, P_a= $\mathbf{b} \vee \mathbf{c}$

	a	b	c	$a \wedge (b \vee c)$
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 (c_i = a) can only be satisfied by row pairs (1, 5), (2, 6), or (3, 7)

Only three pairs can be used

Notes on RACC

- Does RACC subsume predicate and clause coverage?

- RACC was a common interpretation by developers for FAA

- Problem: RACC often leads to infeasible test requirements

Option 3: minor clauses allow predicate to be true and false

- Step 1: 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.

- Step 2 (ACC): TR has two requirements for each c_i : c_i evaluates to true and c_i evaluates to false.

Correlated Active Clause Coverage (CACC) : 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 $p(c_i = true)$!= $p(c_i = false)$.

Example on CACC

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

a determines P when (b=true or c = true)

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

Notes on CACC

- CACC implicitly allows minor clauses to have different values
- CACC explicitly subsumes predicate coverage
- Does CACC subsume clause coverage?

Does CACC subsume clause coverage?

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

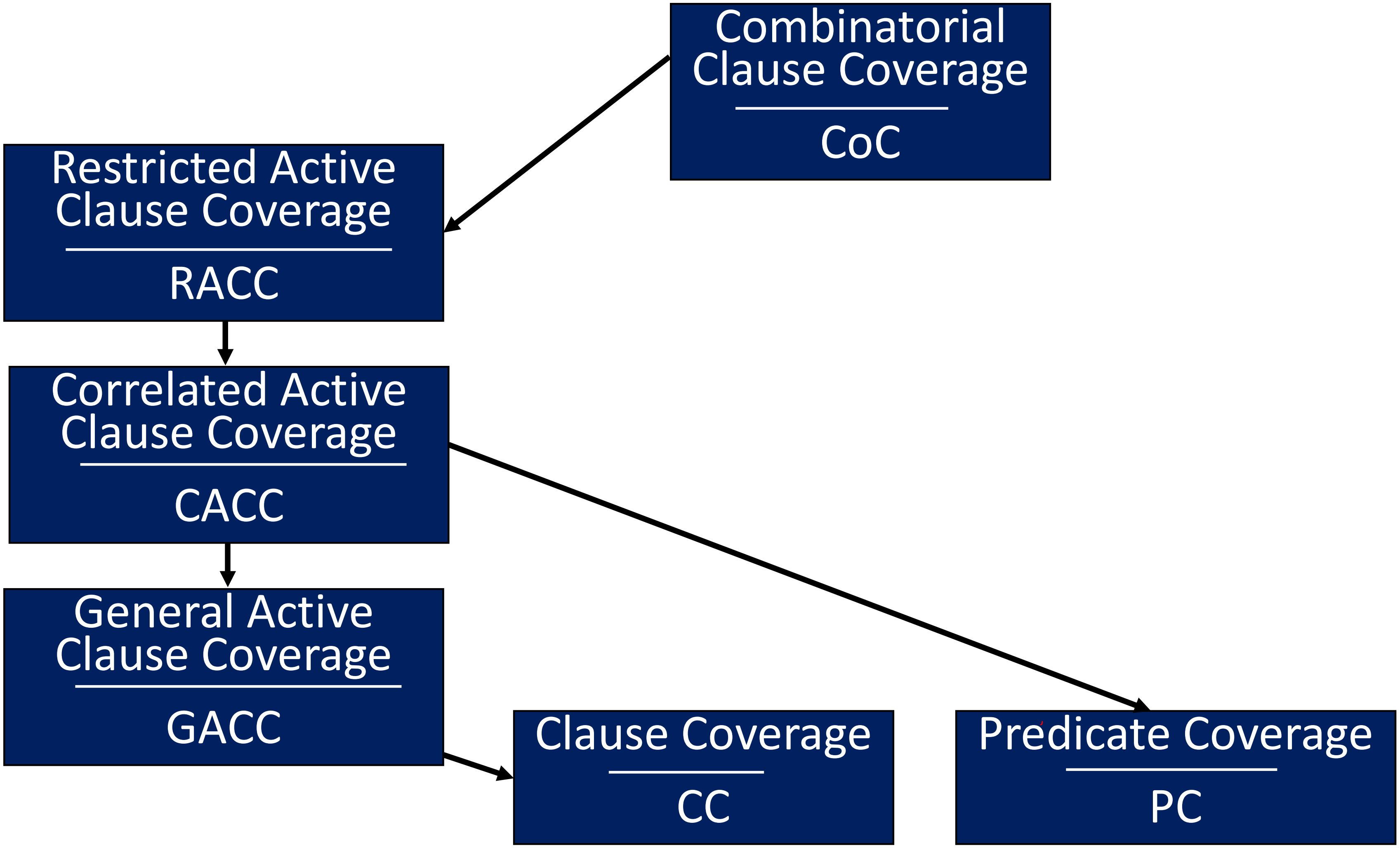
a determines P when (b=true or c = true)

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

Infeasibility

- Consider the predicate: $(a > b \wedge b > c) \vee c > a$
- *Infeasible:* $(a > b) = true, (b > c) = true, (c > a) = true$ is infeasible
- As with other criteria, infeasible test requirements must be recognized and dealt with
- Recognizing infeasible test requirements is hard, and in general, undecidable

Subsumption among Logic coverage criteria



2, 3, 4, 6

An end-to-end example with RACC

b & (diffe determine the v

Ag ca

Fin cau

For (caus

Likewise, for clause *c*, only one pair, TFT and TFF, cause *c* to determine the value of *p*

	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*

How many tests does RACC yield, compared to Combinatorial Clause Coverage?

A more subtle exercise on determination

$a \vee (y \wedge !b)$

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

$p_a = p_{a=true} \oplus p_{a=false}$

$= ((true \wedge b) \vee (true \wedge !b)) \oplus ((false \wedge b) \vee (false \wedge !b))$

$= (b \vee !b) \oplus false$

$= true \oplus false$

$= true$

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

$p_b = p_{b=true} \oplus p_{b=false}$

$= ((a \wedge true) \vee (a \wedge \neg true)) \oplus ((a \wedge false) \vee (a \wedge \neg false))$

$= (a \vee false) \oplus (false \vee a)$

$= a \oplus a$

$= false$

A more subtle exercise on determination (2)

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

- *a* always determines the value of this predicate
- *b* never determines the value – *b* is irrelevant !
- So, why would anyone write a predicate like this?

Logic Coverage Summary

- Predicates are often very simple—in practice, most have <3 clauses
  - In fact, most predicates only have one clause !
- Only clause? PC is enough
- 2 or 3 clauses? CoC is practical
- Advantages of ACC criteria can be significant for large (no. of) predicates
  - CoC is impractical for predicates with many clauses

Next

- Applying Logic Coverage to source code