Logic Coverage Overview part 2

Isaac Griffith

CS 4422 and CS 5599
Department of Computer Science
Idaho State University

Outcomes

At the end of Today's Lecture you will be able to:

- Understand the basic concepts and notation for logic coverage
- Understand the different types of logic coverage criteria

Inspiration

"The trouble with programmers is that you can never tell what a programmer is doing until it's too late." – Seymour Cray

General Active Clause Coverage

General Active Clause Coverage (GACC)

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. 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 $c_j(c_i = true) \neq c_j(c_i = false)$ 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!

Restricted ACC

Restricted Active Clause Coverage

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. 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.

- 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 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. 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 $p(c_i = true) \neq p(c_i = false)$.

- A **more recent** interpretation
- **Implicitly** allows minor clauses to have different values
- Explicitly satisfies (**subsumes**) predicate coverage

CACC and RACC

	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	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

- P_a : b = true or c = true, thus a is the major clause
- CACC can be satisfied by choosing any of rows 1, 2, 3 AND any of rows 5, 6, 7 – a total of 9 pairs
- RACC can only be satisfied by row pairs (1, 5), (2, 6), or (3, 7) – only three pairs

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 C_p, choose minor clauses $c_j, j \neq i$, 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, (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 <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_i OR $c_j(c_i = false) \neq c_j(c_i = false)$ for all c_j.

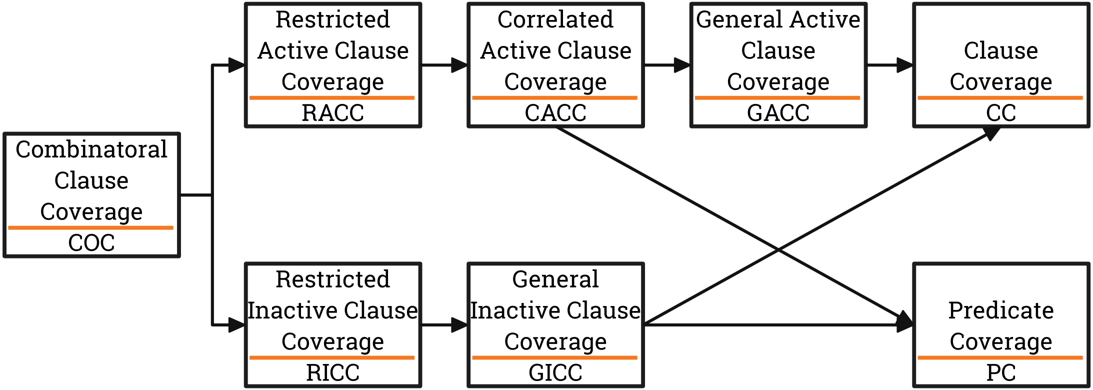
Restricted Inactive Clause Coverage (RICC)

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 <u>does not</u> determine p. The values chosen for the minor clauses c_j <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

- Consider the predicate: $(a > b \vee b > c) \vee c > a$
- $(a > b) = true, (b > c) = true, (c > a) = true$ 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

- 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

Examples

$p = a \vee b$

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

$p = a \wedge b$

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

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

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

- "$\neg b \vee \neg c$" means either b or c can be false
- RACC requires the same choice for both values of, CACC does not

XOR Identity Rules

Exclusive-OR (XOR, $\oplus$) means both cannot be true.

That is, $A \oplus B$ means **“A or B is true, but not both”**

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

Repeated Variables

- The definitions in this chapter yield the same tests no matter how the predicate is expressed
- $(a \vee b) \wedge (c \vee b) == (a \wedge c) \vee b$
- $(a \wedge b) \vee (b \wedge c) \vee (a \wedge c)$
  - Only has 8 possible tests, not 64
- Use the simplest form of the predicate, and ignore contradictory true table assignments

A More Subtle Example

Example

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

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

Example

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

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

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

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			

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	**1**		
2	T	T	F	T			
3	T	F	T	T			
4	T	F	F	F			
5	F	T	T	F	**1**		
6	F	T	F	F			
7	F	F	T	F			
8	F	F	F	F			

b & c are the same, a differs, and p differs ... thus TTT and FTT cause a to determine the value of p

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	1		
2	T	T	F	T	**2**		
3	T	F	T	T			
4	T	F	F	F			
5	F	T	T	F	1		
6	F	T	F	F	**2**		
7	F	F	T	F			
8	F	F	F	F			

Again, b & c are the same, so TTF and FTF cause a to determine the value of p

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	1		
2	T	T	F	T	2		
3	T	F	T	T	**3**		
4	T	F	F	F			
5	F	T	T	F	1		
6	F	T	F	F	2		
7	F	F	T	F	**3**		
8	F	F	F	F			

Finally, this third pair, TFT and FFT, also cause a to determine the value of p

Idaho State University Computer Science

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	1		
2	T	T	F	T	2	**4**	
3	T	F	T	T	3		
4	T	F	F	F		**4**	
5	F	T	T	F	1		
6	F	T	F	F	2		
7	F	F	T	F	3		
8	F	F	F	F			

For clause b, only one pair, TTF and TFF can cause b to determine the value of p

ROAR

Idaho State University Computer Science

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			5
4	T	F	F	F			5
5	F	T	T	F			
6	F	T	F	F			
7	F	F	T	F			
8	F	F	F	F			

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

ROAR

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	1		
2	T	T	F	T	2	4	
3	T	F	T	T	3		5
4	T	F	F	F		4	5
5	F	T	T	F	1		
6	F	T	F	F	2		
7	F	F	T	F	3		
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

Summary

- Predicates are often **very simple**—in practice, most have less than 3 clauses
  - In fact, most predicates only have one clause!
  - With only one clause, PC is enough
  - With 2 or 3 clauses, CoC is practical
  - Advantages of ACC and ICC criteria are significant for large predicates
- **Control software** often has many complicated predicates, with lots of clauses
- **Question** … why don’t complexity metrics count the number of clauses in predicates?

Are there any questions?