

Introduction to Software Testing

Chapter 3.1, 3.2

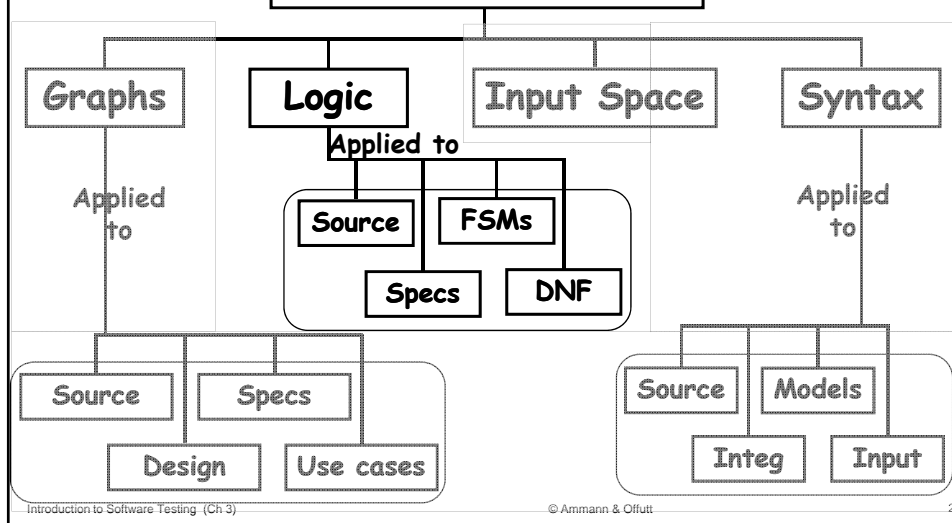
Logic Coverage

Paul Ammann & Jeff Offutt

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

Ch. 3 : Logic Coverage

Four Structures for Modeling Software



Covering Logic Expressions (3.1)

- 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 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

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**
 - It would be nice to quantify that claim!
- **Sources of predicates**
 - Decisions in programs
 - Guards in finite state machines
 - Decisions in UML activity graphs
 - Requirements, both formal and informal
 - SQL queries

Translating from English

- “I am interested in SWE 637 and CS 652”
- $course = swe637 \text{ OR } course = cs652$

Humans have trouble translating from English to Logic

- “If you leave before 6:30 AM, take Braddock to 495, if you leave after 7:00 AM, take Prosperity to 50, then 50 to 495”
- $time < 6:30 \rightarrow path = Braddock \vee time > 7:00 \rightarrow path = Prosperity$
- **Hmm ... this is incomplete !**
- $time < 6:30 \rightarrow path = Braddock \vee time > \text{6:30} \rightarrow path = Prosperity$

Testing and Covering Predicates (3.2)

- 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

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

predicate coverage

Predicate = true

a = 5, b = 10, D = true, m = 1, n = 1, o = 1
 = (5 < 10) \vee true \wedge (1 \geq 1*1)
 = true \vee true \wedge TRUE
 = true

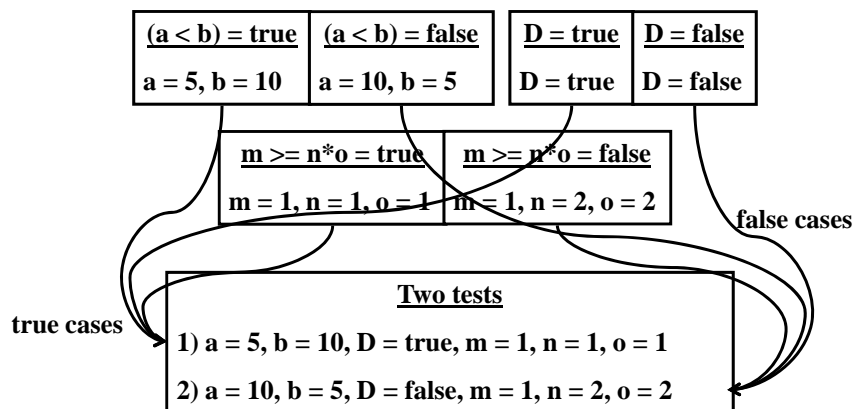
Predicate = false

a = 10, b = 5, D = false, m = 1, n = 1, o = 1
 = (10 < 5) \vee false \wedge (1 \geq 1*1)
 = false \vee false \wedge TRUE
 = 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
 - This is definitely not what we want !
- **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 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
- Consider the first test for clause coverage, which caused each clause to be true:
 - $(5 < 10) \vee true \wedge (1 \geq 1*1)$
- Only the first clause counts !
- To really test the results of a clause, the clause should be the determining factor in the value of the predicate

<u>Determination</u> :	A clause C_i in predicate p , called the <u>major clause</u> , <u>determines</u> p if and only if the values of the remaining <u>minor clauses</u> 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 $B = \text{true}$, p is always true.
 so if $B = \text{false}$, A determines p .
 if $A = \text{false}$, B determines p .

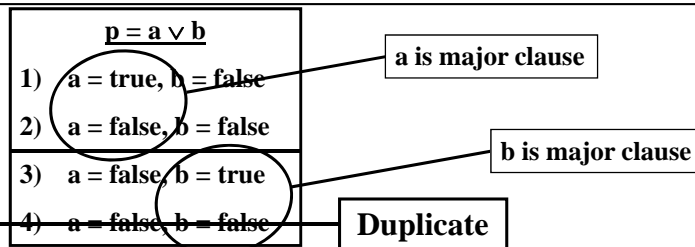
$$P = A \wedge B$$

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_j 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 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 $c_j, j \neq i$, 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 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 $c_j(c_i = true) = c_j(c_i = false)$ for all c_j .

- This has been a common interpretation by aviation developers
- 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 $c_j, j \neq i$, 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

major clause

$P_a : b = \text{true} \vee c = \text{true}$

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

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 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.

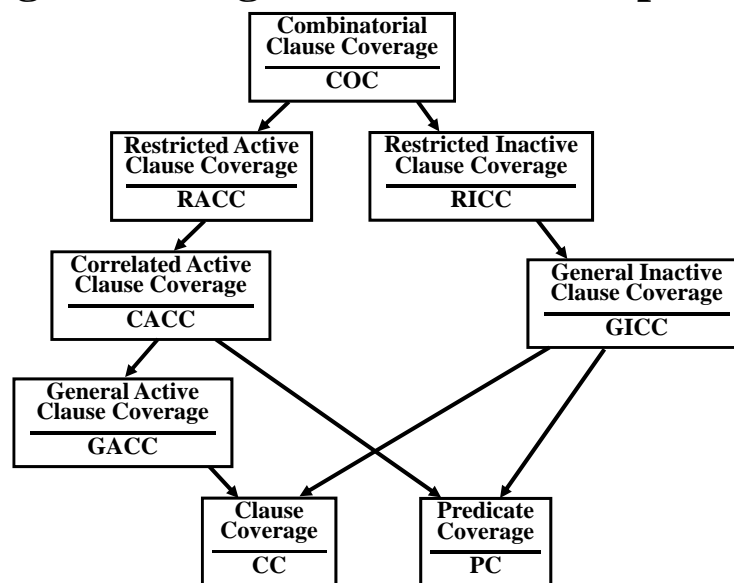
General and Restricted ICC

- 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, $c_j(c_i = true) = c_j(c_i = false)$ for all c_j OR $c_j(c_i = true) \neq c_j(c_i = false)$ for all c_j .

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 $c_j(c_i = true) = c_j(c_i = false)$ 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

Examples

$$p = a \vee b$$

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

$$p = a \wedge b$$

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

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

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

- “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

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 truth table assignments

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 !

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
- Software testing is inexact – engineering, not science

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
- Question ... why don't complexity metrics count the number of clauses in predicates?