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

- **Graphs**
- **Logic**
- **Input Space**
- **Syntax**

Applied to:
- **Source**
- **Specs**
- **DNF**
- **FSMs**

Applied to:
- **Source**
- **Specs**

Applied to:
- **Source**
- **Models**
- **Integ**
- **Input**

Applied to:
- **Design**
- **Use cases**
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
  - $\land$ – the *and* operator
  - $\lor$ – 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

• \((a < b) \lor f(z) \land D \land (m \geq n \times o)\)

• Four clauses:
  – \((a < b)\) – relational expression
  – \(f(z)\) – boolean-valued function
  – \(D\) – boolean variable
  – \((m \geq n \times o)\) – 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”
  • \( course = \text{swe637} \text{ OR } course = \text{cs652} \)

• “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”
  • \( time < 6:30 \rightarrow path = \text{Braddock} \lor time > 7:00 \rightarrow path = \text{Prosperity} \)
  • Hmm ... this is incomplete!
  • \( time < 6:30 \rightarrow path = \text{Braddock} \lor time \geq 6:30 \rightarrow path = \text{Prosperity} \)
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

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

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

\(((a < b) \lor D) \land (m \geq n \times o)\)

Clause coverage

<table>
<thead>
<tr>
<th>Condition 1</th>
<th>Condition 2</th>
<th>Condition 3</th>
<th>Condition 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>((a &lt; b) = true)</td>
<td>((a &lt; b) = false)</td>
<td>(D = true)</td>
<td>(D = false)</td>
</tr>
<tr>
<td>(a = 5, b = 10)</td>
<td>(a = 10, b = 5)</td>
<td>(D = true)</td>
<td>(D = false)</td>
</tr>
<tr>
<td>(m \geq n \times o = true)</td>
<td>(m \geq n \times o = false)</td>
<td>(m = 1, n = 1, o = 1)</td>
<td>(m = 1, n = 2, o = 2)</td>
</tr>
</tbody>
</table>

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!

• 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.

<table>
<thead>
<tr>
<th></th>
<th>$a &lt; b$</th>
<th>$D$</th>
<th>$m \geq n^*o$</th>
<th>$(a &lt; b) \lor D \land (m \geq n^*o)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>4</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>7</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>8</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
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:
  
  - \((5 < 10) \lor \text{true} \land (1 \geq 1*1)\)

• 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

**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**

- **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.

\[
P = A \lor B
\]

- If \( B = \text{true} \), \( p \) is always true.
- So if \( B = \text{false} \), \( A \) determines \( p \).
- If \( A = \text{false} \), \( B \) determines \( p \).

\[
P = A \land B
\]

- If \( B = \text{false} \), \( p \) is always false.
- So if \( B = \text{true} \), \( A \) determines \( p \).
- If \( A = \text{true} \), \( B \) determines \( p \).
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.

\[
p = a \lor b
\]

1) \( a = \text{true}, b = \text{false} \)
2) \( a = \text{false}, b = \text{false} \)
3) \( a = \text{false}, b = \text{true} \)
4) \( a = \text{false}, b = \text{false} \)  

- 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

- 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

\[ p = a \lor (b \land c) \]

**Major clause:**
- \( a = \text{true}, b = \text{false}, c = \text{true} \)
- \( a = \text{false}, b = \text{false}, c = \text{false} \)

Is this allowed?
General Active Clause Coverage (GACC) : For each $p$ in $P$ and each major clause $ci$ in $Cp$, choose minor clauses $cj, j \neq i$, so that $ci$ determines $p$. TR has two requirements for each $ci$: $ci$ evaluates to true and $ci$ evaluates to false. The values chosen for the minor clauses $cj$ do not need to be the same when $ci$ is true as when $ci$ is false, that is, $cj(ci = true) = cj(ci = false)$ for all $cj$ OR $cj(ci = true) \neq cj(ci = false)$ for all $cj$.

- 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 (RACC) : For each $p$ in $P$ and each major clause $ci$ in $Cp$, choose minor clauses $cj, j \neq i$, so that $ci$ determines $p$. TR has two requirements for each $ci$: $ci$ evaluates to true and $ci$ evaluates to false. The values chosen for the minor clauses $cj$ must be the same when $ci$ is true as when $ci$ is false, that is, it is required that $cj(ci = true) = cj(ci = false)$ for all $cj$.

- 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 (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

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>a \land (b \lor c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>7</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>a \land (b \lor c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>7</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

**major clause**

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

- Combinatorial Clause Coverage (COC)
- Restricted Active Clause Coverage (RACC)
- Restricted Inactive Clause Coverage (RICC)
- Correlated Active Clause Coverage (CACC)
- General Active Clause Coverage (GACC)
- Clause Coverage (CC)
- Predicate Coverage (PC)

- General Inactive Clause Coverage (GICC)
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:
  - $pc=true$ is predicate $p$ with every occurrence of $c$ replaced by $true$
  - $pc=false$ is predicate $p$ with every occurrence of $c$ replaced by $false$
- To find values for the minor clauses, connect $pc=true$ and $pc=false$ with exclusive OR.

\[ pc = pc=true \oplus pc=false \]

- After solving, $pc$ describes exactly the values needed for $c$ to determine $p$.
### Examples

**p = a ∨ b**

\[
p_a = p_{a=\text{true}} \oplus p_{a=\text{false}} \\
= (\text{true} \lor b) \text{ XOR } (\text{false} \lor b) \\
= \text{true XOR } b \\
= \neg b
\]

**p = a ∧ b**

\[
p_a = p_{a=\text{true}} \oplus p_{a=\text{false}} \\
= (\text{true} \land b) \text{ XOR } (\text{false} \land b) \\
= b \oplus \text{false} \\
= b
\]

**p = a ∨ (b ∧ c)**

\[
p_a = p_{a=\text{true}} \oplus p_{a=\text{false}} \\
= (\text{true} \lor (b \land c)) \text{ XOR } (\text{false} \lor (b \land c)) \\
= \text{true XOR } (b \land c) \\
= \neg (b \land c) \\
= \neg b \lor \neg c
\]

- "**NOT b ∨ 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

- \((a \lor b) \land (c \lor b) == (a \land c) \lor b\)

- \((a \land b) \lor (b \land c) \lor (a \land c)\)
  - Only has 8 possible tests, not 64

- Use the simplest form of the predicate, and ignore contradictory truth table assignments
A More Subtle Example

\[ p = (a \land b) \lor (a \land \lnot b) \]

- \( p_a = p_{a=true} \oplus p_{a=false} \)
  - \( = ((true \land b) \lor (true \land \lnot b)) \oplus ((false \land b) \lor (false \land \lnot b)) \)
  - \( = (b \lor \lnot b) \oplus false \)
  - \( = true \oplus false \)
  - \( = true \)

- \( p_b = p_{b=true} \oplus p_{b=false} \)
  - \( = ((a \land true) \lor (a \land \lnot true)) \oplus ((a \land false) \lor (a \land \lnot false)) \)
  - \( = (a \lor false) \oplus (false \lor a) \)
  - \( = a \oplus a \)
  - \( = false \)

- \( a \) always determines the value of this predicate
- \( b \) never determines the value – \( b \) is irrelevant!
Infeasible Test Requirements

- Consider the predicate:
  \[(a > b \land b > c) \lor c > a\]

- \((a > b) = true, (b > c) = true, (c > a) = true\) 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 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?