The following are modified versions of the publicly-available slides for Chapter 8 in the Ammann and Offutt Book, “Introduction to Software Testing” (http://www.cs.gmu.edu/~offutt/softwaretest)
Semantic Logic Criteria

• 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

- **Predicate**: an expression that evaluates to a boolean value
- Predicates can contain
  - boolean variables
  - non-boolean variables that are related by $>$, $<$, $==$, $\geq$, $\leq$, $!=$
  - function calls that return booleans
- Internal structure is created by logical operators
  - $\neg$ – the *negation* operator
  - $\wedge$ – the *and* operator
  - $\vee$ – the *or* operator
  - $\implies$ – the *implication* operator
  - $\oplus$ – the *exclusive or* operator
  - $\iff$ – the *equivalence* operator
- A **clause** is a predicate with no logical operators
Example and Facts

• \((a < b) \lor f(z) \land D \land (m \geq n*o)\) has 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
  – 88.5% have 1 clause
  – 9.5% have 2 clauses
  – 1.35% have 3 clauses
  – Only 0.65% have 4 or more!

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

From a study of non-FAA, 63 open-source programs with >400,000 predicates
Translating from English

- “I am interested in CS 5154 and CS 5150”
  - $\text{course} = \text{cs5154 OR course} = \text{cs5150}$

- “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”
  - $(\text{time} < 6:30 \rightarrow \text{path} = \text{Braddock}) \land (\text{time} > 7:00 \rightarrow \text{path} = \text{Prosperity})$
  - Hmm … this is incomplete!
  - $(\text{time} < 6:30 \rightarrow \text{path} = \text{Braddock}) \land (\text{time} \geq 6:30 \rightarrow \text{path} = \text{Prosperity})$
Logic Coverage Criteria

• We use predicates in testing as follows:
  – Developing a model of the software as a set of predicates
  – Requiring tests to satisfy some combination of clauses

• Abbreviations that we will use in later slides:
  – $P$ 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 evaluate 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.

• If predicates are conditions on edges, PC 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

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

**Predicate coverage**

**Predicate = true**

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

\[= (5 < 10) \lor true \land (1 \geq 1\times1)\]

\[= true \lor true \land true\]

\[= true\]

**Predicate = false**

\[a = 5, b = 10, D = true, m = 0, n = 1, o = 1\]

\[= (5 < 10) \lor true \land (0 \geq 1 \times 1)\]

\[= true \lor true \land false\]

\[= false\]
### Clause Coverage Example

\[((a < b) \lor D) \land (m \geq n \cdot o)\]

**Clause coverage**

<table>
<thead>
<tr>
<th>(a &lt; b)</th>
<th>(a &lt; b) = true</th>
<th>(a &lt; b) = false</th>
<th>D = true</th>
<th>D = false</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = 5, b = 10</td>
<td>true</td>
<td>false</td>
<td>a = 10, b = 5</td>
<td>D = true</td>
</tr>
<tr>
<td>D = true</td>
<td>D = false</td>
<td>D = false</td>
<td>D = false</td>
<td>D = false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>m &gt;= n\cdot o</th>
<th>m &gt;= n\cdot o = true</th>
<th>m &gt;= n\cdot o = false</th>
</tr>
</thead>
<tbody>
<tr>
<td>m = 1, n = 1, o = 1</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>m = 1, n = 2, o = 2</td>
<td>m &gt;= n\cdot o = false</td>
<td>false</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
  – Example: a \lor b
  – 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>$m &gt;= n*o$</th>
<th>$((a &lt; b) \lor D) \land (m &gt;= n*o)$</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

- CoC 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:
  – \((5 < 10) \lor \text{true}) \land (1 \geq 1 \times 1)\)
  – Only the last 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

\[ P = A \lor B \]

- if \( B = true \), \( p \) is always true.
- so if \( B = false \), \( A \) determines \( p \).
- if \( A = false \), \( B \) determines \( p \).

\[ P = A \land B \]

- 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 goal is formalized in a **family of 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 $c_i$ determines $p$. TR has two requirements for each $c_i$: $c_i$ evaluates to true and $c_i$ evaluates to false.

<table>
<thead>
<tr>
<th>$p = a \lor b$</th>
<th>$a$ is major clause</th>
<th>$b$ is major clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) $a = true$, $b = false$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2) $a = false$, $b = false$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3) $a = false$, $b = true$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4) $a = false$, $b = false$</td>
<td><strong>Duplicate</strong></td>
<td></td>
</tr>
</tbody>
</table>

- 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 when it is 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

**Example:**

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

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

**Is this allowed?**
General Active Clause Coverage (GACC): For each \( p \) in \( P \) and each major clause \( c_i \) 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!

- We can satisfy GACC without satisfying predicate coverage

- We want to cause predicates to be both true and false
Restricted Active Clause Coverage (RACC) : 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 $c_j$.

- This was 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 $c_j$, $j 
eq i$, 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>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>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>
<tr>
<td>8</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

**major clause**

P<sub>a</sub> : 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

<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>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>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>
<tr>
<td>8</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

**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 $c_j$. 
Infeasibility & Subsumption

• 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
Logic Criteria Subsumption

- **Combinatorial Clause Coverage** (COC)
  - **Restricted Active Clause Coverage** (RACC)
  - **Correlated Active Clause Coverage** (CACC)
  - **General Active Clause Coverage** (GACC)
- **Restricted Inactive Clause Coverage** (RICC)
  - **General Inactive Clause Coverage** (GICC)

- **Clause Coverage** (CC)
- **Predicate Coverage** (PC)
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

\[ p = a \lor b \]

\[
P_a = \begin{cases} 
P_{a=\text{true}} \oplus P_{a=\text{false}} \\
= (\text{true} \lor b) \text{ XOR } (\text{false} \lor b) \\
= \text{true} \text{ XOR } b \\
= \neg b
\end{cases}
\]

\[ p = a \land b \]

\[
P_a = \begin{cases} 
P_{a=\text{true}} \oplus P_{a=\text{false}} \\
= (\text{true} \land b) \oplus (\text{false} \land b) \\
= b \oplus \text{false} \\
= b
\end{cases}
\]

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

\[
P_a = \begin{cases} 
P_{a=\text{true}} \oplus P_{a=\text{false}} \\
= (\text{true} \lor (b \land c)) \oplus (\text{false} \lor (b \land c)) \\
= \text{true} \oplus (b \land c) \\
= \neg (b \land c) \\
= \neg b \lor \neg c
\end{cases}
\]

- “**NOT b \lor NOT c**” means either b or c can be false
- **RACC** requires the same choice for both values of a, **CACC** does not
XOR Identity Rules

Exclusive-OR (xor, ⊕) means both cannot be true

That is, A xor B means

“A or B is true, but not both”

\[
p = A \oplus A \land b = A \land \neg b
\]

\[
p = A \oplus A \lor b = \neg A \land b
\]

with fewer symbols …

\[
p = A \text{ xor} (A \text{ and } b) = A \text{ and } \neg b
\]

\[
p = A \text{ xor} (A \text{ or } b) = \neg A \text{ and } b
\]
A More Subtle Example

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

\[ P_a = P_a=\text{true} \oplus P_a=\text{false} \]
\[ = ((\text{true} \land b) \lor (\text{true} \land \neg b)) \oplus ((\text{false} \land b) \lor (\text{false} \land \neg b)) \]
\[ = (b \lor \neg b) \oplus \text{false} \]
\[ = \text{true} \oplus \text{false} \]
\[ = \text{true} \]

\[ P_b = P_b=\text{true} \oplus P_b=\text{false} \]
\[ = ((a \land \text{true}) \lor (a \land \neg \text{true})) \oplus ((a \land \text{false}) \lor (a \land \neg \text{false})) \]
\[ = (a \lor \text{false}) \oplus (\text{false} \lor a) \]
\[ = a \oplus \text{false} \]
\[ = \text{false} \]

- \( a \) 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

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>( a \land (b \lor c) )</th>
<th>( p_a )</th>
<th>( p_b )</th>
<th>( p_c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In sum, three separate pairs of rows can cause \( a \) to determine the value of \( p \), and only one pair each for \( b \) and \( c \).
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 one clause, PC is enough
  – With 2 or 3 clauses, CoC is practical
  – Advantages of ACC & 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
Next

• Applying Logic Coverage to source code

• Group assignments, start working on your projects

• Reminder: HW2 is due on Monday 3/29 at 9:30am EST