CMPT 473
Software Quality Assurance

Logic Based Criteria

Nick Sumner
Material from Ammonn & Offutt
Logic Based Coverage Criteria

- Logical conditions are pervasive.
Logic Based Coverage Criteria

- Logical conditions are pervasive.
- if statements are the most frequently fixed statements in bug fixes. [Pan, ESE 2008]
Logic Based Coverage Criteria

- Logical conditions are pervasive.
- if statements are the most frequently fixed statements in bug fixes. [Pan, ESE 2008]
- Safety critical systems often involve many complex conditions. (avionics, medical, ...)

[40x511]Logic Based Coverage Criteria
[48x435]●
[74x432]Logical conditions are pervasive.
[48x390]●
[74x385]if statements are the most frequently fixed statements in bug fixes. [Pan, ESE 2008]
[48x311]●
[74x307]Safety critical systems often involve many complex conditions. (avionics, medical, ...)

[74x275](avionics, medical, ...)

[224x275]
Logic Based Coverage Criteria

- Logical conditions are pervasive.
- if statements are the most frequently fixed statements in bug fixes. [Pan, ESE 2008]
- Safety critical systems often involve many complex conditions. (avionics, medical, ...)
- Isn't branch coverage enough?
Logic Based Coverage Criteria

- Logical conditions are pervasive.
- if statements are the most frequently fixed statements in bug fixes. [Pan, ESE 2008]
- Safety critical systems often involve many complex conditions. (avionics, medical, ...)
- Isn't branch coverage enough?

```python
if (a || b) && (c || d):
    s
```
Logic Based Coverage Criteria

- Logical conditions are pervasive.
- *if* statements are the most frequently fixed statements in bug fixes. [Pan, ESE 2008]
- Safety critical systems often involve many complex conditions. (avionics, medical, ...)
- Isn't branch coverage enough?

```python
if (a || b) && (c || d):
    s
```
Logic Based Coverage Criteria

- Logical conditions are pervasive.
- If statements are the most frequently fixed statements in bug fixes. [Pan, ESE 2008]
- Safety critical systems often involve many complex conditions. (avionics, medical, ...)
- Isn't branch coverage enough?

```python
if (a || b) && (c || d):
    s
```

What doesn't branch coverage test?
Logic Based Coverage Criteria

- Logical conditions are pervasive.
- `if` statements are the most frequently fixed statements in bug fixes. [Pan, ESE 2008]
- Safety critical systems often involve many complex conditions. (avionics, medical, ...)
- Isn't branch coverage enough?
- Why not just use path coverage?
Logic Based Coverage Criteria

- Logical conditions are pervasive.
- if statements are the most frequently fixed statements in bug fixes. [Pan, ESE 2008]
- Safety critical systems often involve many complex conditions. (avionics, medical, ...)
- Isn't branch coverage enough?
- Why not just use path coverage?
  1) Scalability
Logic Based Coverage Criteria

- Logical conditions are pervasive.
- if statements are the most frequently fixed statements in bug fixes. [Pan, ESE 2008]
- Safety critical systems often involve many complex conditions. (avionics, medical, ...)
- Isn't branch coverage enough?
- Why not just use path coverage?
  1) Scalability
  2) `if (a \| b) \& (c \| d):`
    `s`
Logic Based Coverage Criteria

- Logical conditions are pervasive.
- if statements are the most frequently fixed statements in bug fixes. [Pan, ESE 2008]
- Safety critical systems often involve many complex conditions. (avionics, medical, ...)
- Isn't branch coverage enough?
- Why not just use path coverage?
  1) Scalability
  2) \( \text{if (} a \mid b \text{)} \& (c \mid d) : s \)
  3) Other languages (e.g., SQL)
Logic Based Coverage Criteria

- We want to reason about the *logical expressions* and how inputs affect their outcomes.

\[(a > 0) && \text{foo}(b) || c\]
Logic Based Coverage Criteria

- We want to reason about the logical expressions and how inputs affect their outcomes.

\[(a > 0) \land \land \text{ foo}(b) \lor c\]

- *Clauses* (in this context) are true or false and don't have logical operators.
Logic Based Coverage Criteria

- We want to reason about the *logical expressions* and how inputs affect their outcomes.

\[(a > 0) \&\& \text{foo}(b) \mid\mid c\]

- *Clauses* (in this context) are true or false and don't have logical operators.

- Logical coverage criteria identify a set values for clauses to test.

<table>
<thead>
<tr>
<th>(a&gt;0)</th>
<th>foo(b)</th>
<th>c</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
Logic Based Coverage Criteria

- We want to reason about the *logical expressions* and how inputs affect their outcomes.

\[(a > 0) \&\& \text{foo(b)} \mid\mid c\]

- *Clauses* (in this context) are true or false and don't have logical operators.

- Logical coverage criteria identify a set values for clauses to test.

<table>
<thead>
<tr>
<th>(a&gt;0)</th>
<th>foo(b)</th>
<th>c</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
Predicate & Clause Coverage

- A *predicate* is simply a boolean expression.
Predicate & Clause Coverage

- A *predicate* is simply a boolean expression.
- *Predicate Coverage* requires each predicate to be true in one test & be false in one test.
• A *predicate* is simply a boolean expression.

• *Predicate Coverage* requires each predicate to be true in one test & be false in one test.

```
if (a || b) && (c || d):
s
if (a | b) & (c | d):
s
```

How does it do in these cases?
Predicate & Clause Coverage

- A *predicate* is simply a boolean expression.
- *Predicate Coverage* requires each predicate to be true in one test & be false in one test.

\[
\begin{align*}
\text{if } (a \lor b) \land (c \lor d) : \\
&\text{s} \\
\end{align*}
\]
Predicate & Clause Coverage

- A *predicate* is simply a boolean expression.
- *Predicate Coverage* requires each predicate to be true in one test & be false in one test.
- *Clause Coverage* requires each clause to be true in one test & be false in one test.
Predicate & Clause Coverage

- A *predicate* is simply a boolean expression.
- *Predicate Coverage* requires each predicate to be true in one test & be false in one test.
- *Clause Coverage* requires each clause to be true in one test & be false in one test.

```java
if (a || b) && (c || d):
    s
```

```java
if (a | b) & (c | d):
    s
```

How does it do in these cases?
Predicate & Clause Coverage

- A *predicate* is simply a boolean expression
- *Predicate Coverage* requires each predicate to be true in one test & be false in one test.
- *Clause Coverage* requires each clause to be true in one test & be false in one test.

```
if (a || b) && (c || d):

s
T F T F
T F T F
T F T F
T F T F
T F T F
T F T F
T F T F
T F T F
```

```
if (a | b) & (c | d):

s
T F T F
T F T F
T F T F
T F T F
T F T F
T F T F
T F T F
T F T F
```

How many tests?
Predicate & Clause Coverage

- A *predicate* is simply a boolean expression
- *Predicate Coverage* requires each predicate to be true in one test & be false in one test.
- *Clause Coverage* requires each clause to be true in one test & be false in one test.

Minimum of 2 tests
Predicate & Clause Coverage

• A predicate is simply a boolean expression

• Predicate Coverage requires each predicate to be true in one test & be false in one test.

• Clause Coverage requires each clause to be true in one test & be false in one test.

if (a || b) && (c || d):

Minimum of 2 tests

a=true, b=true, c=false, d=false
a=false, b=false, c=true, d=true
• A **predicate** is simply a boolean expression

• **Predicate Coverage** requires each predicate to be true in one test & be false in one test.

• **Clause Coverage** requires each clause to be true in one test & be false in one test.

If \((a \lor b) \land (c \lor d)\):

- Minimum of **2** tests

  - \(a=\text{true}, \ b=\text{true}, \ c=\text{false}, \ d=\text{false}\)
  - \(a=\text{false}, \ b=\text{false}, \ c=\text{true}, \ d=\text{true}\)
Combinatorial Coverage

- *Combinatorial/Multiple Condition Coverage* requires each possible combination of clauses to be tested.
  (Each row of the *truth table*)
Combinatorial Coverage

- **Combinatorial/Multiple Condition Coverage** requires each possible combination of clauses to be tested. (Each row of the *truth table*)

(a > 0) && foo(b) || c
Combinatorial Coverage

- **Combinatorial/Multiple Condition Coverage** requires each possible combination of clauses to be tested. (Each row of the truth table)

(a > 0) && foo(b) || c

<table>
<thead>
<tr>
<th>(a&gt;0)</th>
<th>foo(b)</th>
<th>c</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
Defining a Better Goal

- Clause coverage takes each clause into account.
Defining a Better Goal

- Clause coverage takes each clause into account.
- Combinatorial coverage tests the impact of a combination.
Defining a Better Goal

- Clause coverage takes each clause into account.
- Combinatorial coverage tests the impact of a combination.
- Can we test for the impact of each clause?
Defining a Better Goal

- Clause coverage takes each clause into account.
- Combinatorial coverage tests the impact of a combination.
- Can we test for the impact of each clause?

```python
if (a | b) & (c | d):
    s
```

How can we test the impact of a clause?
Defining a Better Goal

- Clause coverage takes each clause into account.
- Combinatorial coverage tests the impact of a combination.
- Can we test for the impact of each clause?

```python
if (a | b) & (c | d):
    s
```

How can we test the impact of a clause?

The *relative* behavior when changing one clause matters.
Defining a Better Goal

- Clause coverage takes each clause into account.
- Combinatorial coverage tests the impact of a combination.
- Can we test for the impact of each clause?
  - This is the intuition behind MC/DC testing...
Modified Condition/Decision Coverage

- Modified Condition/Decision Coverage
Modified Condition/Decision Coverage

- *Modified Condition/Decision Coverage*
  1) Each entry & exit is used
Modified Condition/Decision Coverage

- Modified Condition/Decision Coverage
  1) Each entry & exit is used
  2) Each decision/branch takes every possible outcome
• Modified Condition/Decision Coverage

1) Each entry & exit is used
2) Each decision/branch takes every possible outcome
3) Each clause takes every possible outcome
Modified Condition/Decision Coverage

- **Modified Condition/Decision Coverage**
  1) Each entry & exit is used
  2) Each decision/branch takes every possible outcome
  3) Each clause takes every possible outcome
  4) Each clause independently impacts the outcome
Modified Condition/Decision Coverage

- Modified Condition/Decision Coverage
  1) Each entry & exit is used
  2) Each decision/branch takes every possible outcome
  3) Each clause takes every possible outcome
  4) Each clause independently impacts the outcome

- Use in safety critical systems: avionics, spacecraft, ...
Modified Condition/Decision Coverage

- **Modified Condition/Decision Coverage**
  1) Each entry & exit is used
  2) Each decision/branch takes every possible outcome
  3) Each clause takes every possible outcome
  4) Each clause independently impacts the the outcome

- Use in safety critical systems: avionics, spacecraft, ...

- Not only ensures that clauses are tested, but that each has an impact
How To Show a Clause Has Impact

- A clause *determines* the outcome of a predicate when changing only the value of that clause changes the outcome of the predicate.
How To Show a Clause Has Impact

- A clause *determines* the outcome of a predicate when changing only the value of that clause changes the outcome of the predicate

\[
\varphi(a,b,c) \neq \varphi(a,b,\neg c)
\]
How To Show a Clause Has Impact

• A clause *determines* the outcome of a predicate when changing only the value of that clause changes the outcome of the predicate

\[ \varphi(a,b,c) \neq \varphi(a,b,\neg c) \]

\[ (a \mid \mid b \&\& c) \]
How To Show a Clause Has Impact

- A clause *determines* the outcome of a predicate when changing only the value of that clause changes the outcome of the predicate

\[ \varphi(a,b,c) \neq \varphi(a,b,\neg c) \]

\[
\begin{array}{ccc}
(a \mid \mid b \ \&\& c) \\
\hline
a=F & a=F \\
b=T & b=T \\
c=T & c=F \\
T & F
\end{array}
\]
How To Show a Clause Has Impact

- A clause *determines* the outcome of a predicate when changing only the value of that clause changes the outcome of the predicate.

\[ \varphi(a,b,c) \neq \varphi(a,b,\neg c) \]

\[ (a \mid\mid b \&\& c) \]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
</table>
| F | T | T | T
| F | T | F | F

This pair of tests shows the impact of \( c \).
How To Show a Clause Has Impact

- A clause *determines* the outcome of a predicate when changing only the value of that clause changes the outcome of the predicate.

- The basic steps come from & and |
How To Show a Clause Has Impact

- A clause *determines* the outcome of a predicate when changing only the value of that clause changes the outcome of the predicate.

- The basic steps come from & and |

  \[
  a \ & \ b \\
  \text{If } a=\text{True, } b \text{ determines the outcome.}
  \]
How To Show a Clause Has Impact

• A clause *determines* the outcome of a predicate when changing only the value of that clause changes the outcome of the predicate.

• The basic steps come from & and |

\[
\begin{align*}
a & \land b \\
\text{If } a=\text{True}, \ b & \text{ determines the outcome.}
\end{align*}
\]

\[
\begin{align*}
a & \lor b \\
\text{If } a=\text{False}, \ b & \text{ determines the outcome.}
\end{align*}
\]
How To Show a Clause Has Impact

- A clause *determines* the outcome of a predicate when changing only the value of that clause changes the outcome of the predicate.

- The basic steps come from & and |:

  \[
  \begin{align*}
  a \& b & \quad \text{If } a=\text{True}, b \text{ determines the outcome.}
  \\
  a \mid b & \quad \text{If } a=\text{False}, b \text{ determines the outcome.}
  \end{align*}
  \]

- By definition, solve $\varphi = \varphi_c=\text{true} \oplus \varphi_c=\text{false}$
How To Show a Clause Has Impact

- A clause *determines* the outcome of a predicate when changing only the value of that clause changes the outcome of the predicate.

- The basic steps come from & and |:

  \[
  \begin{align*}
  a & \land b \\
  & \text{If } a=\text{True}, b \text{ determines the outcome.}
  \\
  a & \lor b \\
  & \text{If } a=\text{False}, b \text{ determines the outcome.}
  \end{align*}
  \]

- By definition, solve $\varphi = \varphi_{c=\text{true}} \oplus \varphi_{c=\text{false}}$

  Let's try:
  \[
  \begin{align*}
  a & \land b \\
  a & \lor b \\
  a & \lor (b \land c)
  \end{align*}
  \]
How To Show a Clause Has Impact

- Given $a \mid (b \& c)$, generate tests for $a$
How To Show a Clause Has Impact

- Given \( a \mid (b \& c) \), generate tests for \( a \)

\[
a \text{ has impact } \iff \text{ #T } \mid (b \& c) \neq \text{ #F } \mid (b \& c)
\]

by definition
How To Show a Clause Has Impact

- Given \( a \mid (b \& c) \), generate tests for \( a \)

  \[
  a \text{ has impact} \iff \#T \mid (b \& c) \neq \#F \mid (b \& c) \\
  \iff \#T \neq
  \]

How To Show a Clause Has Impact

• Given $a \mid (b \& c)$, generate tests for $a$

  $a$ has impact $\iff$ $\#T \mid (b \& c) \neq \#F \mid (b \& c)$

  $\iff$ $\#T \neq b \& c$
How To Show a Clause Has Impact

- Given \( a \mid (b \& c) \), generate tests for \( a \) has impact

\[
\begin{align*}
a \text{ has impact} & \iff \#T \mid (b \& c) \neq \#F \mid (b \& c) \\ & \iff \#T \neq b \& c \\ & \iff \#T = \neg b \mid \neg c
\end{align*}
\]
How To Show a Clause Has Impact

- Given \( a \mid (b \& c) \), generate tests for \( a \)

  \[
  a \text{ has impact} \iff \#T \mid (b \& c) \neq \#F \mid (b \& c) \\
  \iff \#T \neq b \& c \\
  \iff \#T = \neg b \mid \neg c \\
  \iff b \text{ is false or } c \text{ is false}
  \]
How To Show a Clause Has Impact

• Given \( a \mid (b \& c) \), generate tests for \( a \)

  \[
  \begin{align*}
  \text{a has impact} & \iff \#T \mid (b \& c) \neq \#F \mid (b \& c) \\
  & \iff \#T \neq b \& c \\
  & \iff \#T = \neg b \mid \neg c \\
  & \iff b \text{ is false or } c \text{ is false}
  \end{align*}
  \]

  defines two different ways to test \( a \)
How To Show a Clause Has Impact

- Given $a \mid (b \& c)$, generate tests for $a$
  
  $a$ has impact $\iff \neg \neg (b \& c)$ $\neq \neg \neg \neg (b \& c)$
  
  $\iff \neg \neg (b \& c)$ $\neq b \& c$
  
  $\iff \neg \neg (b \& c)$ $= \neg b \| \neg c$
  
  $\iff b$ is false or $c$ is false

defines two different ways to test $a$

Have $b$ be $\neg F$

- $a=\neg T$, $b=\neg F$, $c=\neg T$
- $a=\neg F$, $b=\neg F$, $c=\neg T$
How To Show a Clause Has Impact

- Given \( a \mid (b \& c) \), generate tests for \( a \)

\[
\begin{align*}
\text{a has impact} & \iff #T \mid (b \& c) \neq #F \mid (b \& c) \\
& \iff #T \neq b \& c \\
& \iff #T = \neg b \mid \neg c \\
& \iff b \text{ is false or } c \text{ is false}
\end{align*}
\]

defines two different ways to test \( a \)

<table>
<thead>
<tr>
<th>Have ( b ) be #F</th>
<th>Have ( c ) be #F</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a=#T, b=#F, c=#T )</td>
<td>( a=#T, b=#T, c=#F )</td>
</tr>
<tr>
<td>( a=#F, b=#F, c=#T )</td>
<td>( a=#F, b=#T, c=#F )</td>
</tr>
</tbody>
</table>
Showing Impact

- What about \((a \& b) \mid (a \& \neg b)\)?
  - Can you show the impact of \(a\)?
  - Can you show the impact of \(b\)?
McDC coverage also identifies redundancies that are likely bugs.
Generating a Test Suite with MC/DC

\[(a > 0) \land \land \text{foo}(b) \lor c\]
Generating a Test Suite with MC/DC

\[(a > 0) \land \land \text{foo}(b) \lor \lor c\]

Select a first clause.
Generating a Test Suite with MC/DC

1  
2  
3

\((a > 0) \land \land \text{foo}(b) \lor c\)

\[ 2=\#T, \ 3=\#F \]

Solve the constraints for other clauses.
Generating a Test Suite with MC/DC

\[(a > 0) \land \text{foo(b)} \lor c\]

Include tests for the clause in the test suite.

Tests: 1=\#T, 2=\#T, 3=\#F

1=\#F, 2=\#T, 3=\#F
Generating a Test Suite with MC/DC

\[ (a > 0) \land \land \text{foo}(b) \lor \lor c \]

Consider the next clause.

Tests:

1=\#T, 2=\#T, 3=\#F

Tests: 1=\#F, 2=\#T, 3=\#F
Generating a Test Suite with MC/DC

(a > 0) && foo(b) || c

Try to add a test for it based on existing tests.

Tests:

1=#T, 2=#T, 3=#F
1=#F, 2=#T, 3=#F
1=#T, 2=#F, 3=#F
Generating a Test Suite with MC/DC

\[(a > 0) \&\& \text{foo}(b) || c\]

Repeat for the last clause.

Tests:

1=#T, 2=#T, 3=#F
1=#F, 2=#T, 3=#F
1=#T, 2=#F, 3=#F
1=#T, 2=#F, 3=#T
Generating a Test Suite with MC/DC?

• *BUT* NASA recommended *not* generating MC/DC coverage.
Generating a Test Suite with MC/DC?

- **BUT** NASA recommended *not* generating MC/DC coverage.
  - Use MC/DC as a means of evaluating test suites generated by other means
• Some historical ambiguities
  – Originally only required impact when changing clause
  – Changing other clauses at the same time was allowed!
MC/DC Over Time

- Some historical ambiguities
  - Originally only required impact when changing clause
  - Changing other clauses at the same time was allowed!
  - Why is this problematic?
MC/DC Over Time

- Some historical ambiguities
  - Originally only required impact when changing clause
  - Changing other clauses at the same time was allowed!
  - Why is this problematic?

- The form presented here is also known as *Restricted Active Clause Coverage*
Logic and MC/DC Testing

- Tests complex interactions in conditions.
- Required for avionics software.

Is it good? Bad?