Covering Logical Expressions

- Logic expression show up in many situations
- Covering logical expressions have a long history, many are the covering criteria are mandated by the US FAA in safety critical software.
- Logical expressions can come from many sources:
  - Decisions in programs
  - Finite State Machines and statechartes
  - Requirements
Covering Logical Expressions

- Typical balance in software testing:
  - In theory too many options to test everything.
  - Find good coverage criteria that have a chance of covering useful cases.
Motivation

```c
if ((x > 0 && y > 0) || z > 0)
    {do_something;}
else
    {something_else;}
```

Branch or edge coverage would only require one test case that executes the two branches. Instead you might want to test all the possibilities of the logical expression.
So a test corresponding to the line \textit{TFT} would require some value \(x > 0\), \(y \leq 0\) and \(z > 0\). You would have to find a test path with these values.
Logical Connectives: Revisions?

<table>
<thead>
<tr>
<th>( P )</th>
<th>( Q )</th>
<th>( P \land Q, P &amp;&amp; Q )</th>
<th>( P \lor Q, P || Q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( P )</th>
<th>( Q )</th>
<th>( P \rightarrow Q )</th>
<th>( P \leftrightarrow Q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( P )</th>
<th>( Q )</th>
<th>( P \oplus Q ) Exclusive Or</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
A **predicate** is an expression that evaluates to true or false.

A predicate can contain:
- Boolean variables.
- Expressions evaluating to Boolean variables that contain comparison operations: `>`, `<`, `==`, `>=`, `!=`, `<=`
- Boolean function calls.

Internal structure created by logical operators (see previous slide)

A clause is a predicate with no logical operators.

- \( P \land Q \) is a predicate
- \( x>0 \) is a clause.
Testing and Covering Predicates

- Reduce the artefact to a set of predicates
- State covering criteria in terms of predicates and clauses.

- $\mathcal{P}$ - the set of predicates.
- $p$ is a single predicate in $\mathcal{P}$.
- $\mathcal{C}$ - set of clauses in $\mathcal{P}$.
- $\mathcal{C}_p$ - set of clauses in predicate $p$.
- $c$ is a single clause in $\mathcal{C}$.
Predicate Coverage

Very simple and very blunt.

- *Predicate Coverage (PC)*: For each $p$ in $\mathcal{P}$, the test requirements include that $p$ evaluates to true and $\overline{p}$ evaluates to false.
Predicate Coverage

```c
if (x>0 || y>0)
    {do_something;}
else
    {something_else;}
if (z>0)
    {foo;}
else
    {bar;}
```

The predicates are \( x>0 \) \( \text{or} \) \( y>0 \) and \( z>0 \). So we have 4 test requirements. So test case inputs could be \( x = 3 \) \( \land \) \( y = -10 \), \( x = -3 \) \( \land \) \( y = -10 \), \( z = 20 \) and \( z = -120 \).
Clause Coverage

- Not every clause is exercised.

So we get a new coverage criterion.

- *Clause Coverage (CC)* - For each clause $c$ in $\mathcal{C}$, we have a test requirement that $c$ evaluates to true and $c$ evaluates to false.
The clause are $x>0$, $y>0$ and $z>0$. So we have 6 test requirements.
Example

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

- Predicate true: \( a = 5, \ b = 10, \ D \) is true, \( m = 1 \) and \( n = 1 \).
  \[ ((5 < 10) \lor T) \land (1 \geq 1) = (T \lor T) \land T \]

- Predicate false: \( a = 10, \ b = 5, \ D \) is false, \( m = 1 \) and \( n = 1 \).
  \[ ((5 < 10) \lor F) \land (1 \geq 1) = (T \lor T) \land T \]
  \[ (F \lor F) \land T = F. \]
Clause Coverage

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

Test requirements

- $a < b$ is true: $a = 5$, $b = 10$.
- $a < b$ is false: $a = 10$, $b = 5$.
- $D$ is true and $D$ is false.
- $m \geq n$ is true: $m = 1$, $n = 1$.
- $m \geq n$ is false: $m = 1$, $n = 2$.

We can do this with two test cases:

- $a = 5, b = 10, D$ is true, $m = 1$ and $n = 1$.
- $a = 10, b = 5, D$ is false, $m = 1$ and $n = 2$. 
Problems with Predicate and Clause Coverage

- Predicate coverage does not fully exercise all the clauses, especially in the presence of short circuit evaluation:
  \[ P \&\& Q \]

- If \( P \) is false then \( Q \) is never evaluated\(^1\) in C++.

\(^1\)It is a bit more complicated than this.
Problems with Predicate and Clause Coverage

- Clause coverage does not imply predicate coverage.
- We can satisfy clause coverage without causing the predicate to be both true and false.

\[(P \lor Q) \land R\]

- Clauses \(P\), \(Q\) and \(R\). 6 test requirements \(P = T, P = F, Q = T, Q = F, R = T\) and \(R = F\).
- \((P = F \lor Q = F) \land R = T\) evaluates to false.
- \((P = T \lor Q = T) \land R = F\) evaluates to false.
- All test requirements covered. So we need something better, all possibilities?
Combinatorial Coverage

- Combinatorial Coverage For each \( p \) in \( \mathcal{P} \), the test requirement includes that each clause in \( C_p \) evaluate to each possible combination of truth values.
- That really means test all rows in the truth table.
Combinatorial Coverage

- Simple, clean, neat, and comprehensive.
- But it can be quite expensive $2^N$ tests for $N$ clauses.
- Lots of suggestion in the literature, but the general idea is simple:
  - Test each clause *independently* from the other clauses.
- You have to work out what exactly does independent mean. The book uses “making clauses active”.
Active Clauses

- The major weakness of clause coverage is that the values do not always make a difference.
- We really want the test results of a clause to be a determining factor in the truth or falsehood of the predicate.
Determination

- **Major clause** is the clause under consideration.
- **Minor clause** are the rest of the clauses.
- Let $c_i$ be the major clause in a predicate $p$. Then $c_i$ determines $p$ if and only if the value of the remaining minor clauses are such that changing the value of $C_i$ changes the value of $p$. 
Determining Predicates

- \( P = A \lor B \).
  - Major clause A, and minor clause B. If \( B = F \) then A determines \( P \).
  - Major clause B, and minor clause A. If \( A = F \) then B determines \( P \).

- \( Q = A \land B \).
  - if \( B = F \) then \( Q \) is always false.
  - If \( B = T \) then A determines \( p \).
Determining Predicates

- For a determining clause you have to take into account the assignments of the minor clauses.
- Basic idea is for each clause you want assignments to the minor clauses so that your clause becomes a determining clause.
- It becomes more complicated, when you consider how the different test requirements overlap.
Active Clause Coverage

- Active Clause Coverage (ACC): For each predicate $p$ in $\mathcal{P}$ and each major clause, choose values for the minor clauses so that the major clause determines $p$. The test requirements contain two a requirement that the major clause evaluates to true and the major clause evaluates to false.
Active Clause Coverage

\[ P = A \lor B \]

- \( A \) as the major clause.
  - \( A = T, B = F \)
  - \( A = F, B = F \).

- \( B \) as the major clause.
  - \( B = T, A = F \)
  - \( B = F, A = F \).

There is a source of *ambiguity*: Do the minor clauses have to have the same values when the major clause is true and false?
Resolving the Ambiguity

\[ P = A \lor (B \land C) \]

Major clause A:
- \( A = T, B = F, C = T \)
- \( A = F, B = F, C = F \)

Do we allow different assignments to the minor clauses?

Three possible criteria:
- Minor clauses do not need to be the same.
- Minor clauses do need to be the same.
- Minor clauses force predicate to become both true and false.
General Active Clause Coverage

*General Active Clause Coverage.* For each predicate \( p \) in \( \mathcal{P} \) and each major clause in \( C_p \), choose minor clauses so that it determines \( p \). Test requirements include that the major clause evaluates to true and false. The values of the minor clauses do *not* need to be the same.

This is the most general definition.
General Active Clause Coverage

General active clause coverage does not imply predicate coverage.

\[ P = A \leftrightarrow B \]

- **Major clause A**
  - \( A = T, B = F, P = A \leftrightarrow B = F \).
  - \( A = F, B = T, P = A \leftrightarrow B = F \).

- **Major Clause B**
  - \( B = T, A = F, P = A \leftrightarrow B = F \).
  - \( B = F, A = T, P = A \leftrightarrow B = F \).

We have general active clause coverage, but the predicate always evaluates to false. We really want clauses to cause predicates to evaluate to both true and false.
Restricted Active Clause Coverage

- *Restricted Active Clause Coverage*: For each predicate and each major clause in each predicate, choose values of minor clauses so that the major determines the predicate. Test requirements include each major clause evaluates to true and false. The values chosen for the minor clauses must be the same.

- Been used in aviation software.

- No logical reason for the values of the minor clauses to be the same.

- We still haven’t solved the problem of the predicate evaluating to both true and false.
Correlated Active Clause Coverage:

For each predicate and each major clause in each predicate, choose values of minor clauses so that the major determines the predicate. Test requirements include each major clause evaluates to true and false. The values of the minor clauses must be chosen such that the predicate evaluates to true and false.
Correlated Active Clause Coverage

\[ A \land (B \lor C) \]

- Major Clause A.
  - \(A = T, B = T, C = T, A \land (B \lor C) = T\)
  - \(A = F, B = T, C = F, A \land (B \lor C) = F\)
Making clauses determine a predicate

- Finding values for minor clauses it easy for simple predicates.
- Trick with more complicated predicates.
- \( P_{c=T} \) replace every occurrence of \( c \) by \( T \).
- \( P_{c=F} \) replace every occurrence of \( c \) by \( F \).
- Solve \( P_c = P_{c=T} \oplus P_{c=F} = T \) this gives you the values of the minor clauses for the major clause \( c \) to determine the predicate.
Making clauses determine a predicate

- \( P = A \lor B \), Major clause \( A \).
  - \( P_A = P_{A=T} \oplus P_{A=F} = (T \lor B) \oplus (F \lor B) \) which gives \( T \oplus B = \neg B \).
  - So \( B \) must be false.

- \( P = A \land B \), Major clause \( A \).
  - \( P_A = P_{A=T} \oplus P_{A=F} = (T \land B) \oplus (F \land B) \) which gives \( B \oplus F = B \).
  - So \( B \) must be true.
Making clauses determine a predicate

- \( P = A \lor (B \land C) \), Major clause A.
  - \( P_A = P_{A=T} \oplus P_{A=F} = (T \lor (B \land C)) \oplus (false \lor (B \land C)) \)
    which gives \( T \oplus (B \land C) = \neg(B \land C) = \neg B \lor \neg C \).
  - So \( B = F \) and \( C = T \) is a solution.
  - \( A \lor (F \land T) = A \lor F \).
Logic Coverage Summary

- Combinatorial coverage gives too many test cases.
- Predicate coverage is too blunt.
- Clause coverage has to be refined to force the predicate to evaluate to both true and false.
- We want the test case to do some work, so we use correlated active clause coverage.

Think about infeasible test requirements: $X > 0 \land X < 0$. 