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 $>$, $<$, $\geq$, $\leq$, $!=$
  - 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
  - $\equiv$ - the equivalence operator
- A clause is a predicate with no logical operators
Examples

- \((a < b) \lor f(z) \land (m \geq n^2)\)

Four clauses:
- \((a < b)\) – relational expression
- \(f(z)\) – boolean-valued function
- \(D\) – boolean variable
- \((m \geq n^2)\) – 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 \lor course = 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”
  - \((time < 6:30 \rightarrow path = Braddock) \land (time > 7:00 \rightarrow path = Prosperity)\)
  - Hmm … this is incomplete!

- \((time < 6:30 \rightarrow path = Braddock) \land (time \geq 6:30 \rightarrow path = Prosperity)\)

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

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

**Predicate coverage**

- **Predicate = true**
  
  \[
  a = 5, b = 10, D = true, m = 1, n = 1, o = 1 \\
  = (5 < 10) \lor true \land (1 \geq 1) \\
  = true \land true \land true \\
  = true
  \]

- **Predicate = false**
  
  \[
  a = 10, b = 5, D = false, m = 1, n = 1, o = 1 \\
  = (10 < 5) \lor false \land (1 \geq 1) \\
  = false \lor false \land true \\
  = false
  \]

Clause Coverage Example

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

**Clause coverage**

- **Two tests**
  
  - \(a < b) = true\)
    
    | \(a = 5, b = 10\) | \(a = 10, b = 5\) |
    |---|---|
    | D = true | D = true |
    | \(m = n \times o = true\) | \(m = n \times o = false\) |
    | \(m = 1, n = 1, o = 1\) | \(m = 1, n = 2, o = 2\) |

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>(a \leq b)</th>
<th>D</th>
<th>(m \geq n \times o)</th>
<th>((a &lt; b) \lor D) \land (m \geq n \times o))</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>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>7</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>8</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>9</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

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:
  \(-(5 < 10) \lor \text{true} \land (1 \geq 1^2)\)
- 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

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

Active Clause Coverage (ACC): For each \(p\) in \(P\) and each major clause \(C_j\) in \(C_p\), choose minor clauses \(C_j \neq i\), so that \(C_j\) determines \(p\). TR has two requirements for each \(C_j: C_i\) evaluates to true and \(C_j\) 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

- 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: \( a \)
\[ a = \text{true}, b = \text{false}, c = \text{true} \]
\[ a = \text{false}, b = \text{false}, c = \text{true} \]

General Active Clause Coverage

- 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

- 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

- A more recent interpretation
- Implicitly allows minor clauses to have different values
- Explicitly satisfies (subsumes) predicate coverage
CACC and RACC

<table>
<thead>
<tr>
<th>b</th>
<th>c</th>
<th>a ( A(b \lor \neg c) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>


**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.
- Each \( c_i \) does not affect \( p \), so it 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, \( c_j(c_i = true) \neq c_j(c_i = false) \) for all \( c_j \) OR \( c_j(c_i = true) = c_j(c_i = false) \) 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)**
- **General Active Clause Coverage (GACC)**
- **Correlated Active Clause Coverage (CACC)**
- **Restricted Active Clause Coverage (RACC)**
- **General Inactive Clause Coverage (GICC)**
- **Restricted Inactive Clause Coverage (RICC)**
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**

\[ p = a \lor b \]

\[ P_a = P_{a=true} \oplus P_{a=false} \]

\[ = (true \lor b) \oplus (false \lor b) \]

\[ = true \lor b \]

\[ P_b = P_{b=true} \oplus P_{b=false} \]

\[ = (true \land c) \oplus (false \land c) \]

\[ = (b \land c) \]

\[ = (\neg b \lor \neg c) \]

\[ = \neg (b \land c) \]

**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**

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

\[ P_a = P_{a=true} \oplus P_{a=false} \]

\[ = ((true \land b) \lor (true \land \neg b)) \oplus ((false \land b) \lor (false \land \neg b)) \]

\[ = (b \lor \neg b) \oplus false \]

\[ = true \]

\[ P_b = P_{b=true} \oplus P_{b=false} \]

\[ = ((a \land true) \lor (a \land \neg true)) \oplus ((a \land false) \lor (a \land \neg false)) \]

\[ = (a \lor \neg a) \oplus false \]

\[ = (a \lor false) \oplus (false \lor a) \]

\[ = false \]

\[ \bullet \quad \text{a always determines the value of this predicate} \]

\[ \bullet \quad \text{b never determines the value - b is irrelevant!} \]
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?

Logic Expressions from Source

- Predicates are derived from decision statements in programs
- In programs, most predicates have less than four clauses
  - Wise programmers actively strive to keep predicates simple
- When a predicate only has one clause, COC, ACC, ICC, and CC all collapse to predicate coverage (PC)
- Applying logic criteria to program source is hard because of reachability and controllability:
  - Reachability: Before applying the criteria on a predicate at a particular statement, we have to get to that statement
  - Controllability: We have to find input values that indirectly assign values to the variables in the predicates
    - Variables in the predicates that are not inputs to the program are called internal variables
- These issues are illustrated through an example in the following slides …

Thermostat (pg 1 of 2)

```
1  // Jeff Offutt--October 2010
2  // Programmable Thermostat
3  import java.io.*;
4  class thermostat
5  {
6      private Heater myHeater;
7      // Decide whether to turn the heater on, and for how long.
8      public boolean turnHeaterOn (int curTemp, /* Current temperature reading */
9          int thresholdDiff, /* Temp difference until we turn heater on */
10         Minutes timeSinceLastRun, /* Time since heater stopped */
11         Minutes minLag, /* How long I need to wait */
12         Time timeOfDay, /* current time (Hours and minutes) */
13         Day dayOfWeek, /* Monday, Tuesday, ... */
14         Settings programmedSettings [], /* User's program, by day */
15         boolean Override, /* Has user overridden the program */
16         int overTemp /* OverridingTemp */
17             )
```

```
Thermostat (pg 2 of 2)

```java
19 {
20     int desiredTemp;
21     // getPeriod() translates time into Morning, Day, Evening, Night
22     desiredTemp = programmedSettings [dayOfWeek].getDesiredTemp
23            (getPeriod [TimeOfDay]);
24     if (((curTemp < desiredTemp - thresholdDiff) ||
25          (Override && curTemp < overTemp - thresholdDiff)) &&
26          timeSinceLastRun.greaterThan (minLag))
27     {  // Turn on the heater
28        // How long? Assume 1 minute per degree (Fahrenheit)
29        int timeNeeded = curTemp - desiredTemp;
30        if (Override)
31           timeNeeded = curTemp - overTemp;
32        myHeater.setRunTime (timeNeeded);
33        return (true);
34     }
35     else
36        return (false);
37 }  // End turnHeaterOn
38 }  // End class
```

### Two Thermostat Predicates

**Predicate Coverage (true)**

- `a : true`
- `b : true`
- `c : true`
- `d : true`

```java
curTemp < desiredTemp - thresholdDiff : true
Override : true
curTemp < overTemp - thresholdDiff : true
timeSinceLastRun.greaterThan (minLag) : true
```
**Predicate Coverage (false)**

\[(a \| (b \& c)) \& d\]

- \(a\): false
- \(b\): false
- \(c\): false
- \(d\): false

**Cur Correlated Active Clause Coverage (2 of 5)**

- \(a\): T T F T
- \(b\): T T F T
- \(c\): F T T F
- \(d\): T T T F

Duplicates

Six tests needed for CACC on Thermostat

**Correlated Active Clause Coverage (3 of 5)**

- \(\text{curTemp < desiredTemp - thresholdDiff}\)
- \(\text{desiredTemp = programmedSettings[Monday].setDesiredTemp(Morning, 69)}\)
- \(\text{dayOfWeek = Monday}\)
- \(\text{timeOfDay = 8:00}\)
- \(\text{Override}\)
- \(\text{Override = 0}\)
- \(\text{timeSinceLastRun.greaterThan(minLag)}\)
- \(\text{These values then need to be placed into calls to turnHeaterOn( ) to satisfy the 6 tests for CACC} \)

**Correlated Active Clause Coverage (1 of 5)**

\[P_a = ((a \| (b \& c)) \& d) \oplus ((a \| (b \& c)) \& d)\]

\[((T \| (b \& c)) \& d) \oplus ((F \| (b \& c)) \& d)\]

\[(T \& d) \oplus ((b \& c) \& d)\]

\[5(b \& c) \& d\]

\[(!b \| c) \& d\]

Check with the logic coverage web app

http://cs.gmu.edu:8080/offutt/coverage/logicCoverage

**Cur Correlated Active Clause Coverage (1 of 5)**

\[P_a = ((a \| (b \& c)) \& d) \oplus ((a \| (b \& c)) \& d)\]

\[((T \| (b \& c)) \& d) \oplus ((F \| (b \& c)) \& d)\]

\[(T \& d) \oplus ((b \& c) \& d)\]

\[5(b \& c) \& d\]

\[(!b \| c) \& d\]
Correlated Active Clause Coverage (4 of 5)

desiredTemp = programmedSettings [Monday].setDesiredTemp (Morning, 69)
1. T t f t
   a = T : curTemp = 63;   c = f : curTemp = 66
   turnHeaterOn ( 66/66, 5, 12, 10, 8:00, Monday, programmedSettings, true, 70 )
2. F t t
   turnHeaterOn ( 66, 5, 12, 10, 8:00, Monday, programmedSettings, true, 70 )
3. f T t t
   a = f : curTemp = 66;   c = t : curTemp = 63
   turnHeaterOn ( 63/66, 5, 12, 10, 8:00, Monday, programmedSettings, true, 70 )
4. f t F f
   turnHeaterOn ( 66, 5, 12, 10, 8:00, Monday, programmedSettings, true, 70 )
5. t t t T
   turnHeaterOn ( 63, 5, 12, 10, 8:00, Monday, programmedSettings, true, 70 )
6. t t t F
   turnHeaterOn ( 63, 5, 12, 10, 8:00, Monday, programmedSettings, true, 70 )

Correlated Active Clause Coverage (5 of 5)

• Tests 1 and 3 are infeasible with the values we chose
• But we can choose different values for clause c
• curTemp is fixed by the solution to clause a
• thresholdDiff is also fixed by the solution to clause a
• So we choose different values for overtemp ...

Program Transformation Issues

if ((a && b) || c) {
    S1;
} else {
    S2;
}  
Transform (1)?

if (a) {
    if (b) {
        S1;
    } else {
        if (c) {
            S1;
        } else {
            S2;
        }
    }
} else {
    if (c) {
        S1;
    } else {
        S2;
    }
}  
Transform (2)?

d = a && b;

if (d) {
    if (e) {
        S1;
    } else {
        S2;
    }
} else {
    S2;
}

Problems with Transformed Programs

• Maintenance is certainly harder with Transform (1)
  – Not recommended!
• Coverage on Transform (1)
  – PC on transform does not imply CACC on original
  – CACC on original does not imply PC on transform
• Coverage on Transform (2)
  – Structure used by logic criteria is “lost”
  – Hence CACC on transform 2 only requires 3 tests
  – Note: Mutation analysis (Chapter 5) addresses this problem
• Bottom Line: Logic coverage criteria are there to help you!
Summary: Logic Coverage for Source Code
- Predicates appear in decision statements
  - if, while, for, etc.
- Most predicates have less than four clauses
  - But some applications have predicates with many clauses
- The hard part of applying logic criteria to source is resolving the internal variables
- Sometimes setting variables requires calling other methods
- Non-local variables (class, global, etc.) are also input variables if they are used
- If an input variable is changed within a method, it is treated as an internal variable thereafter
- To maximize effect of logic coverage criteria:
  - Avoid transformations that hide predicate structure

Specifications in Software
- Specifications can be formal or informal
  - Formal specs are usually expressed mathematically
  - Informal specs are usually expressed in natural language
- Lots of formal languages and informal styles are available
- Most specification languages include explicit logical expressions, so it is very easy to apply logic coverage criteria
- Implicit logical expressions in natural-language specifications should be re-written as explicit logical expressions as part of test design
  - You will often find mistakes
- One of the most common is preconditions...

Preconditions
- Programmers often include preconditions for their methods
- The preconditions are often expressed in comments in method headers
- Preconditions can be in javadoc, “requires”, “pre”, ...

Example - Saving addresses
// name must not be empty
// state must be valid
// zip must be 5 numeric digits
// street must not be empty
// city must not be empty

Rewriting to logical expression
name != "" \land state \in stateList \land zip >= 00000 \land zip <= 99999 \land street \neq "" \land city \neq ""

Shortcut for Predicates in Conjunctive Normal Form
- A predicate is in conjunctive normal form (CNF) if it consists of clauses or disjuncts connected by the and operator
  - \( A \land B \land C \land \ldots \)
  - \( (A \lor B) \land (C \lor D) \)
- A major clause is made active by making all other clauses true
- ACC tests are “all true” and then a “diagonal” of false values:
Shortcut for Predicates in Disjunctive Normal Form

- A predicate is in disjunctive normal form (DNF) if it consists of clauses or conjuncts connected by the or operator
  \[ A \lor B \lor C \lor \ldots \]
  \[ (A \land B) \lor (C \land D) \]
- A major clause is made active by making all other clauses false
- ACC tests are “all false” and then a “diagonal” of true values:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>

Summary: Logic Coverage for Specs

- Logical specifications can come from lots of places:
  - Preconditions
  - Java asserts
  - Contracts (in design-by-contract development)
  - OCL conditions
  - Formal languages
- Logical specifications can describe behavior at many levels:
  - Methods and classes (unit and module testing)
  - Connections among classes and components
  - System-level behavior
- Many predicates in specifications are in disjunctive normal or conjunctive normal form—simplifying the computations

Covering Finite State Machines

- FSMs are graphs
  - nodes represent state
  - edges represent transitions among states
- Transitions often have logical expressions as guards or triggers
- As we said:
  Find a logical expression and cover it

Example—Subway Train

- \( \text{trainSpeed} = 0 \land \text{platform}=\text{left} \land \neg \text{emergencyStop} \land \neg \text{overrideOpen} \land \text{doorsClear} \)
- \( \text{trainSpeed} = 0 \land \text{platform}=\text{right} \land \neg \text{emergencyStop} \land \neg \text{overrideOpen} \land \text{doorsClear} \)
- \( \text{secondPlatform} = \text{right} \land \neg \text{emergencyStop} \land \neg \text{overrideOpen} \land \text{doorsClear} \)
- \( \text{secondPlatform} = \text{left} \land \neg \text{emergencyStop} \land \neg \text{overrideOpen} \land \text{doorsClear} \)
- \( \text{All Doors Open} \)
- \( \text{All Doors Closed} \)
Determination of the Predicate

\[
\text{trainSpeed} = 0 \land \text{platform} = \text{left} \land (\text{inStation} \lor (\text{emergencyStop} \land \text{overrideOpen}))
\]

\[
\text{trainSpeed} = 0 \land \text{platform} = \text{left} \land (\text{inStation} \lor (\text{emergencyStop} \land \text{overrideOpen}))
\]

\[
\text{trainSpeed} = 0 \land \text{platform} = \text{left} \\
\text{inStation} : \text{trainSpeed} = 0 \land \text{platform} = \text{left} \\
\text{emergencyStop} : \text{trainSpeed} = 0 \land \text{platform} = \text{left} \\
\text{overrideOpen} : \text{trainSpeed} = 0 \land \text{platform} = \text{left}
\]

Test Truth Assignments (CACC)

<table>
<thead>
<tr>
<th>trainSpeed</th>
<th>platform</th>
<th>inStation</th>
<th>emergencyStop</th>
<th>overrideOpen</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Test Scripts

- Test scripts are executable sequences of value assignments
- Mapping problem: The names used in the FSMs may not match the names in the program
  - Sometimes a direct name-to-name mapping can be found
  - Sometimes more complicated actions must be taken to assign the appropriate values
  - Simulation: Directly inserting value assignments into the middle of the program
- The solution to this is implementation-specific
Summary FSM Logic Testing

- FSMs are widely used at all levels of abstraction
- Many ways to express FSMs
  - Statecharts, tables, Z, decision tables, Petri nets, ...
- Predicates are usually explicitly included on the transitions
  - Guards
  - Actions
  - Often represent safety constraints
- FSMs are often used in embedded software