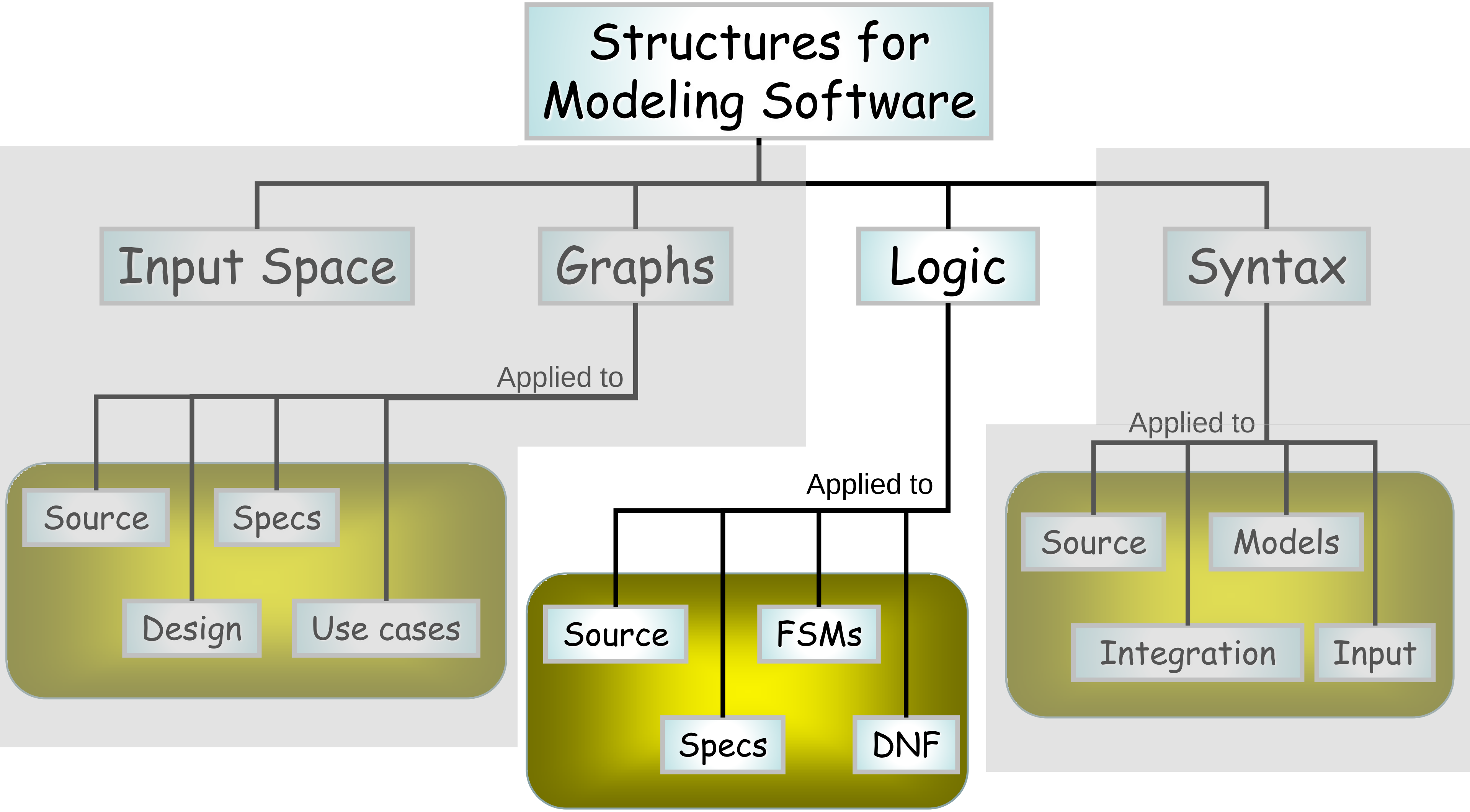
Software Quality Assurance and Testing

Amr Kamel
Associate Professor,
Department of Computer Science,
Faculty of Computers & Information,
Cairo University.

Logic Coverage



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 state-charts
  - 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
  - $\wedge$ – the *and* operator
  - $\vee$ – 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

Example and Facts

- $(a < b) \vee f(z) \wedge D \wedge (m >= n*o)$ has four clauses:
  - $(a < b)$ – relational expression
  - $f(z)$ – Boolean-valued function
  - D – Boolean variable
  - $(m >= n*o)$ – 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 !

from a study of 63 open source programs, >400,000 predicates

- 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 CS496 and CS352"
- *course = CS495 OR course = CS352*

Humans have trouble translating from English to Logic

"If you leave before 6:30 AM, take ring-road to Giza, if you leave after 7:00 AM, take 26 of July Corridor to Lebanon Sq, then ring-road to Giza"

- *(time < 6:30 $\rightarrow$ path = Ring Road) $\wedge$ (time > 7:00 $\rightarrow$ path = 26of July Corridor)*
- Hmm … this is incomplete !
- *(time < 6:30 $\rightarrow$ path = Ring Road) $\wedge$ (time >= 6:30 $\rightarrow$ path = 26of July Corridor)*

Logic Coverage Criteria

- 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

<u>Predicate Coverage (PC)</u> : 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 …

<u>Clause Coverage (CC)</u> : For each *c* in *C*, *TR* contains two requirements: *c* evaluates to true, and *c* evaluates to false.

Predicate Coverage Example

$((a < b) \vee D) \wedge (m >= n*o)$

predicate coverage

<u>Predicate = true</u>

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

$= (5 < 10) \vee true \wedge (1 >= 1*1)$

$= true \vee true \wedge TRUE$

= true

<u>Predicate = false</u>

a = 10, b = 5, D = false, m = 1, n = 1, o = 1

$= (10 < 5) \vee false \wedge (1 >= 1*1)$

$= false \vee false \wedge TRUE$

= false

Clause Coverage Example

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

clause coverage

(a < b) = true	(a < b) = false	D = true	D = false
a = 5, b = 10	a = 10, b = 5	D = true	D = false

m >= n*o = true	m >= n*o = false
m = 1, n = 1, o = 1	m = 1, n = 2, o = 2

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

true cases

false cases

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 <u>not</u> 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 Cp to evaluate to each possible combination of truth values.

	a < b	D	m >= n*o	((a < b) ∨ D) ∧ (m >= n*o)
1	T	T	T	T
2	T	T	F	F
3	T	F	T	T
4	T	F	F	F
5	F	T	T	T
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

Combinatorial Coverage

- This is simple, but quite expensive!
- 2^N tests, where N is the number of clauses
  - Impractical for predicates with more than 3 or 4 clauses
- Literature has lots of alternative 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?
- In our course, we adopt the idea of "*making clauses active*" …

Active Clauses

- Clause coverage has a weakness: The values do not always make a difference
- 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

$P = A \vee B$	**$P = A \wedge B$**
if B = *true*, p is always true.	if B = *false*, p is always false.
so if B = *false*, A determines p.	so if B = *true*, A determines p.
if A = *false*, B determines p.	if A = *true*, B determines p.

- Goal: Find tests for each clause when the clause determines the value of the predicate
- This is formalized in a family of criteria that have subtle, but very important, differences

Active Clause Coverage

a is major clause

b is major clause

$p = a \vee b$

1) a = true, b = false	3) a = false, b = true
2) a = false, b = false	4) ~~a = false, b = false~~

Duplicate

Active Clause Coverage (ACC) : For each p in P and each major clause c_i in Cp, choose minor clauses c_j, j != 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.

- 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

$$p = a \vee (b \wedge c)$$

Major clause : a

a = true, b = false, c = true

a = false, b = false, c = false

Is this allowed?

- This question caused confusion among testers
- 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

General Active Clause Coverage

General Active Clause Coverage (GACC) … ch p in P and each major clause c_i in C_p, choose minor c… = i, so that ci determines p. TR has two requi… r each ci : ci evaluates to true and ci evaluates to fa… lues chosen for the minor clauses cj do not ne… e same when ci is true as when ci is false, that is, ci(… = cj(ci = false) for all cj OR cj(ci = true) != cj(ci = false) … 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

<u>Restricted Active Clause Coverage (RACC):</u> For each p in P and each major clause ci in Cp, choose minor clauses cj, j != 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

Correlated Active Clause Coverage (CACC): For each p in P and each major clause ci in Cp, choose minor clauses cj, j != 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 cause p to be true for one value of the major clause ci and false for the other, that is, it is required that p(ci = true) != p(ci = false).

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

CACC and RACC

	a	b	c	a ∧ (b ∨ c)
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	F
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

	a	b	c	a ∧ (b ∨ c)
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	F
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

major clause Pa: 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

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 ci in Cp, choose minor clauses cj, j != i, so that ci does not determine p. TR has four requirements for each ci : (1) ci evaluates to true with p true, (2) ci evaluates to false with p true, (3) ci evaluates to true with p false, and (4) ci evaluates to false with p false.

General and Restricted ICC

<u>General Inactive Clause Coverage (GICC)</u>: For each p in P and each major clause ci in Cp, choose minor clauses cj , j != i, so that ci does not determine p. 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) != cj(ci = false) for all c_j.

<u>Restricted Inactive Clause Coverage (RICC)</u>: For each p in P and each major clause ci in Cp, choose minor clauses cj, j != i, so that ci does not determine p. 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.

- c_i does not determine p, so cannot correlate with p
- Predicate coverage is always guaranteed

Infeasibility & Subsumption

- Consider the predicate:

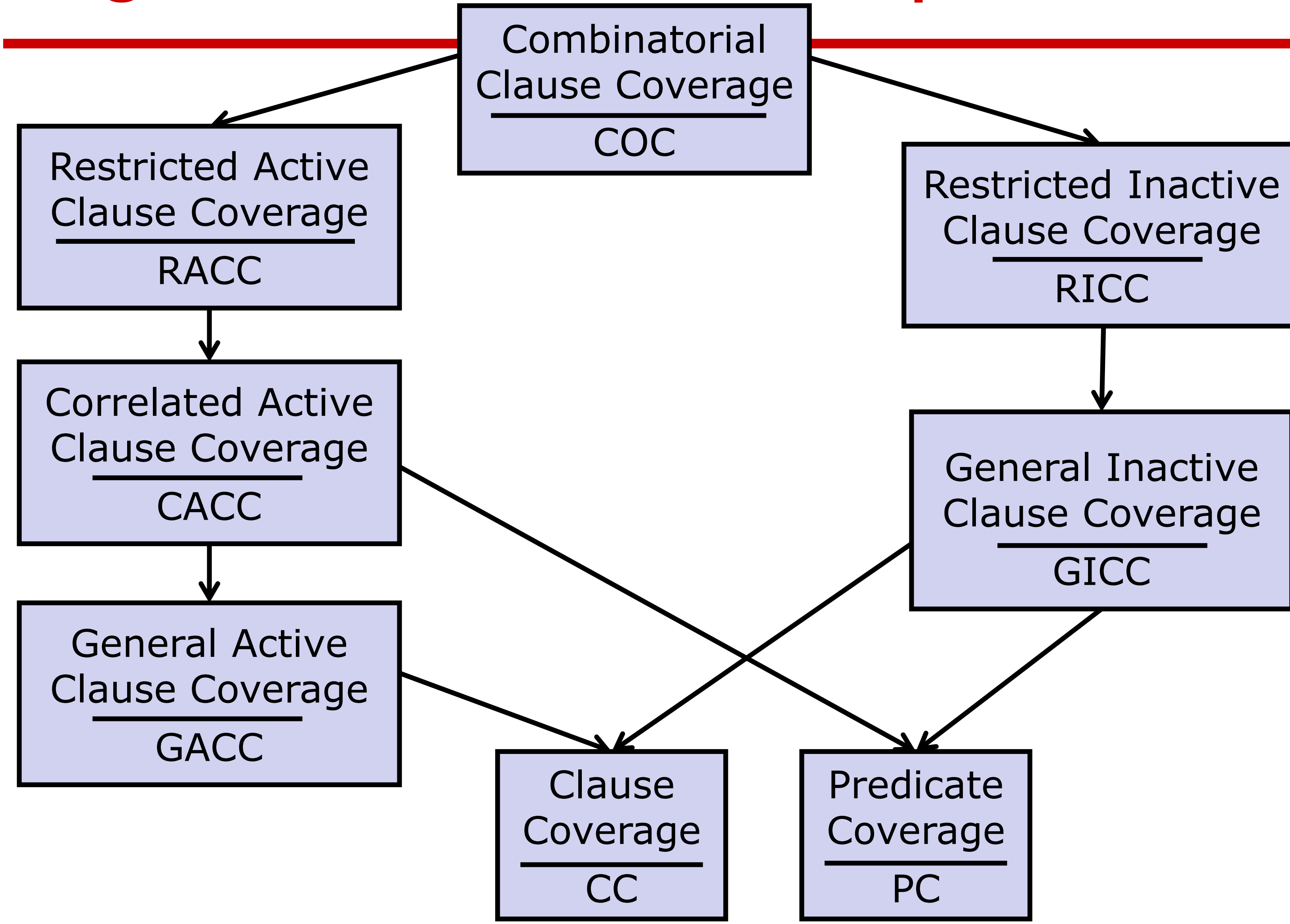
$$(a > b \wedge b > c) \vee c > a$$

- $(a > b) = true, (b > c) = true, (c > a) = true$

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 Criteria Subsumption



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

$$p = a \vee b$$

$$\begin{aligned} p_a &= p_{a=true} \oplus p_{a=false} \\ &= (true \vee b) \oplus (false \vee b) \\ &= true \oplus b \\ &= \neg b \end{aligned}$$

$$p = a \wedge b$$

$$\begin{aligned} p_a &= p_{a=true} \oplus p_{a=false} \\ &= (true \wedge b) \oplus (false \wedge b) \\ &= b \oplus false \\ &= b \end{aligned}$$

$$p = a \vee (b \wedge c)$$

$$\begin{aligned} p_a &= p_{a=true} \oplus p_{a=false} \\ &= (true \vee (b \wedge c)) \oplus (false \vee (b \wedge c)) \\ &= true \oplus (b \wedge c) \\ &= \neg (b \wedge c) \\ &= \neg b \vee \neg c \end{aligned}$$

- RACC requires the same choice for *b* & *c* values, CACC does not

A More Subtle Example

$$
\begin{aligned}
p_a &= p_{a=true} \oplus p_{a=false} \\
&= ((true \wedge b) \vee (true \wedge \neg b)) \oplus ((false \wedge b) \vee (false \wedge \neg b)) \\
&= (b \vee \neg b) \oplus false \\
&= true \oplus false \\
&= true
\end{aligned}
$$

$$p = (a \wedge b) \vee (a \wedge \neg b)$$

$$
\begin{aligned}
p_b &= p_{b=true} \oplus p_{b=false} \\
&= ((a \wedge true) \vee (a \wedge \neg true)) \oplus ((a \wedge false) \vee (a \wedge \neg false)) \\
&= (a \vee false) \oplus (false \vee a) \\
&= a \oplus a \\
&= false
\end{aligned}
$$

- *a* always determines the value of this predicate
- *b* never determines the value – *b* is irrelevant!

Tabular Method for Determination

- A truth ta

b & c are the same, a differs, and p differs …
thus
valu

Aga
cau

Fina
a to

For
cau

Likewise, for clause c, only one pair, TFT and TFF, cause c to determine the value of p

Example

	a	b	c	$a \wedge (b \vee c)$	p_a	p_b	p_c
1	T	T	T	T	★		
2	T	T	F	T	★	★	
3	T	F	T	T	★		★
4	T	F	F	F		★	★
5	F	T	T	F	★		
6	F	T	F	F	★		
7	F	F	T	F	★		
8	F	F	F	F			

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 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

Logic Expressions from Source

- 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*

Thermostat (*Pg 1 of 2*)

```
// Jeff Offutt & Paul Ammann—September 2014
// Programmable Thermostat
import java.io.*;
public class Thermostat
{
   private int curTemp;              // Current temperature reading
   private int thresholdDiff;        // Temp difference until heater on
   private int timeSinceLastRun; // Time since heater stopped
   private int minLag;               // How long I need to wait
   private boolean Override;         // Has user overridden the program
   private int overTemp;             // Overriding Temp
   private int runTime;              // output of turnHeaterOn–how long to run
   private boolean heaterOn;         // output of turnHeaterOn – whether to run
  private Period period;             // morning, day, evening, or night
  private DayType day;               // week day or weekend day
```

Thermostat (*Pg 2 of 2*)

```
// Decide whether to turn the heater on, and for how long.
 public boolean turnHeaterOn (ProgrammedSettings pSet)
 {
 int dTemp = pSet.getSetting (period, day);
 if (((curTemp < dTemp - thresholdDiff) ||
      (Override && curTemp < overTemp - thresholdDiff)) &&
      (timeSinceLastRun > minLag))
 {  // Turn on the heater
    // How long? Assume 1 minute per degree (Fahrenheit)
    int timeNeeded = curTemp - dTemp;
    if (Override)
       timeNeeded = curTemp - overTemp;
    setRunTime (timeNeeded);
    setHeaterOn (true);
  return (true);
 }
```

```
else
  {
     setHeaterOn (false);
     return (false);
  }
} // End turnHeaterOn
```

Two Thermostat Predicates

28-30 : (((curTemp < dTemp - thresholdDiff) ||
(Override && curTemp < overTemp - thresholdDiff)) &&
timeSinceLastRun > minLag))

34 : (Override)

<u>Simplify</u>

a: curTemp < dTemp - thresholdDiff

b: Override

c: curTemp < overTemp - thresholdDiff

d: timeSinceLastRun > minLag)

28-30: (a || (b && c)) && d

34: b

Reachability for Thermostat Predicates

28-30 : True

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

curTemp < dTemp - thresholdDiff

Need to solve for the internal variable *dTemp*

pSet.getSetting (period, day);

```
setSetting (Period.MORNING, DayType.WEEKDAY, 69);
setPeriod (Period.MORNING);
setDay (DayType.WEEKDAY);
```

Predicate Coverage (*true*)

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

a: true   b: true
c: true   d: true

a: curTemp < dTemp – thresholdDiff: true
b: Override: true
c: curTemp < overTemp – thresholdDiff: true
d: timeSinceLastRun > (minLag): true

```
thermo = new Thermostat();  // Needed object
settings = new ProgrammedSettings();  // Needed object
settings.setSetting (Period.MORNING, DayType.WEEKDAY, 69);  // dTemp
thermo.setPeriod (Period.MORNING);  // dTemp
thermo.setDay (DayType.WEEKDAY);  // dTemp
thermo.setCurrentTemp (63);  // clause a
thermo.setThresholdDiff (5);   // clause a
thermo.setOverride (true);  // clause b
thermo.setOverTemp (70);  // clause c
thermo.setMinLag (10);  // clause d
thermo.setTimeSinceLastRun (12);  // clause d
assertTrue (thermo.turnHeaterOn (settings));   // Run test
```

Correlated Active Clause Coverage

$Pa = ((a \mid\mid (b \&\& c)) \&\& d) \oplus ((a \mid\mid (b \&\& c)) \&\& d)$

$((T \mid\mid (b \&\& c)) \&\& d) \oplus ((F \mid\mid (b \&\& c)) \&\& d)$

$(T \&\& d) \oplus ((b \&\& c) \&\& d)$

$d \oplus ((b \&\& c) \&\& d)$

$T \oplus ((b \&\& c) \&\& T)$

$!(b \&\& c) \&\& d$

$(!b \mid\mid !c) \&\& d$

Correlated Active Clause Coverage
(2 of 6)

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

	a	b	c	d
P_a :	**T**	t	f	t
	F	t	f	t
Pb:	f	**T**	t	t
	f	**F**	t	t
Pc:	~~f~~	~~t~~	~~**T**~~	~~t~~
	~~f~~	~~t~~	~~**F**~~	~~t~~
Pd:	t	t	t	**T**
	t	t	t	**F**

duplicates

Six tests needed for CACC on Thermostat

Correlated Active Clause Coverage

	curTemp	dTemp	thresholdDiff
a=t : curTemp < dTemp - thresholdDiff	63	69	5
a=f : !(curTemp < dTemp - thresholdDiff)	66	69	5

dTemp:

```
settings.setSettings (Period.MORNING, DayType.WEEKDAY, 69)
thermo.setPeriod (Period.MORNING);
thermo.setDay (Daytype.WEEKDAY);
```

	Override
b=t : Override	T
b=f : !Override	F

These values then need to be placed into calls to turnHeaterOn() to satisfy the 6 tests for CACC

	curTemp	overTemp	thresholdDiff
c=t : curTemp < overTemp - thresholdDiff	63	72	5
c=f : !(curTemp < overTemp - thresholdDiff)	66	67	5

	timeSinceLastRun	minLag
d=t : timeSinceLastRun > minLag	12	10
d=f : !(timeSinceLastRun > minLag)	8	10

Correlated Active Clause Coverage

dTemp = 69 (period = MORNING, daytype = WEEKDAY)

1. **T t f t**

```
thermo.setCurrentTemp (63);
thermo.setThresholdDiff (5);
thermo.setOverride (true);
thermo.setOverTemp (67); // c is false
thermo.setMinLag (10);
thermo.setTimeSinceLastRun (12);
```

2. **F t f t**

```
thermo.setCurrentTemp (66); // a is false
thermo.setThresholdDiff (5);
thermo.setOverride (true);
thermo.setOverTemp (67); // c is false
thermo.setMinLag (10);
thermo.setTimeSinceLastRun (12);
```

Correlated Active Clause Coverage

dTemp = 69 (period = MORNING, daytype = WEEKDAY)

3. **f T t t**

```
thermo.setCurrentTemp (66); // a is false
thermo.setThresholdDiff (5);
thermo.setOverride (true);
thermo.setOverTemp (72); // to make c true
thermo.setMinLag (10);
thermo.setTimeSinceLastRun (12);
```

4. **F f T t**

```
thermo.setCurrentTemp (66); // a is false
thermo.setThresholdDiff (5);
thermo.setOverride (false); // b is false
thermo.setOverTemp (72);
thermo.setMinLag (10);
thermo.setTimeSinceLastRun (12);
```

Correlated Active Clause Coverage

dTemp = 69 (period = MORNING, daytype = WEEKDAY)

5. **t t t T**

```
thermo.setCurrentTemp (63);
thermo.setThresholdDiff (5);
thermo.setOverride (true);
thermo.setOverTemp (72);
thermo.setMinLag (10);
thermo.setTimeSinceLastRun (12);
```

6. **t t t F**

```
thermo.setCurrentTemp (63);
thermo.setThresholdDiff (5);
thermo.setOverride (true);
thermo.setOverTemp (72);
thermo.setMinLag (10);
thermo.setTimeSinceLastRun (8); // d is false
```

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;
}
```

Problems With Transformation 1

- We trade one problem for two problems :
  - Maintenance becomes harder
  - Reachability becomes harder
- Consider coverage :
  - CACC on the original requires four rows marked in table
  - PC on the transformed version requires five different rows

a	b	c	$(a \wedge b) \vee c$	CACC	PC_T
T	T	T	T		X
T	T	F	T	X	
T	F	T	T	X	X
T	F	F	F	X	X
F	T	T	T		X
F	T	F	F	X	
F	F	T	T		
F	F	F	F		X

- PC on the transformed version has two problems :
  - It does not satisfy CACC on the original
  - It is more expensive (more tests)

Program Transformation Issue 2

```
if ((a && b) || c)
{
    S1;
}
else
{
    S2;
}
```

Transform (1) ?

```
d = a && b;
e = d || c;
if (e)
{
    S1;
}
else
{
    S2;
}
```

Problems With Transformation 2

- We move complexity into computations
  - Logic criteria are not effective testing computations
- Consider coverage :
  - CACC on the original four requires rows marked in the table
  - PC on the transformed version requires only two

a	b	c	$(a \wedge b) \vee c$	CACC	PC_T
T	T	T	T		X
T	T	F	T	X	
T	F	T	T	X	
T	F	F	F	X	
F	T	T	T		
F	T	F	F	X	
F	F	T	T		
F	F	F	F		X

- PC on the transformed version becomes equivalent to clause coverage on the original
  - Not an effective testing technique

Transforming Does Not Work

Logic coverage criteria exist to help us make better software

Circumventing the criteria is unsafe

Side Effects in Predicates

- Side effects occur when a value is changed while evaluating a predicate
  - A clause appears twice in the same predicate
  - A clause in between changes the value of the clause that appears twice
- Example :

A && (B || A)   B is : changeVar (A)

  - Evaluation : Runtime system checks *A*, then *B*, if *B* is false, check *A* again
  - But now *A* has a different value!
  - How do we write a test that has two different values for the same predicate?

We suggest a social solution : Go ask the programmer

Summary : Logic Coverage for Source Code

- Predicates appear in decision statements (if, while, for, etc.)
- Most predicates have less than four clauses
  - But some programs have a few predicates with many clauses
- The hard part of applying logic criteria to source is usually 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
- Avoid transformations that hide predicate structure

Correlated Active Clause Coverage

$P_a = ((a \mid\mid (b \;\&\&\; c)) \;\&\&\; d) \oplus ((a \mid\mid (b \;\&\&\; c)) \;\&\&\; d)$

$((T \mid\mid (b \;\&\&\; c)) \;\&\&\; d) \oplus ((F \mid\mid (b \;\&\&\; c)) \;\&\&\; d)$

$(T \;\&\&\; d) \oplus ((b \;\&\&\; c) \;\&\&\; d)$

$d \oplus ((b \;\&\&\; c) \;\&\&\; d)$

$T \oplus ((b \;\&\&\; c) \;\&\&\; T)$

$!(b \;\&\&\; c) \;\&\&\; d$

$(!b \mid\mid !c) \;\&\&\; d$

Check with the logic coverage web app

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