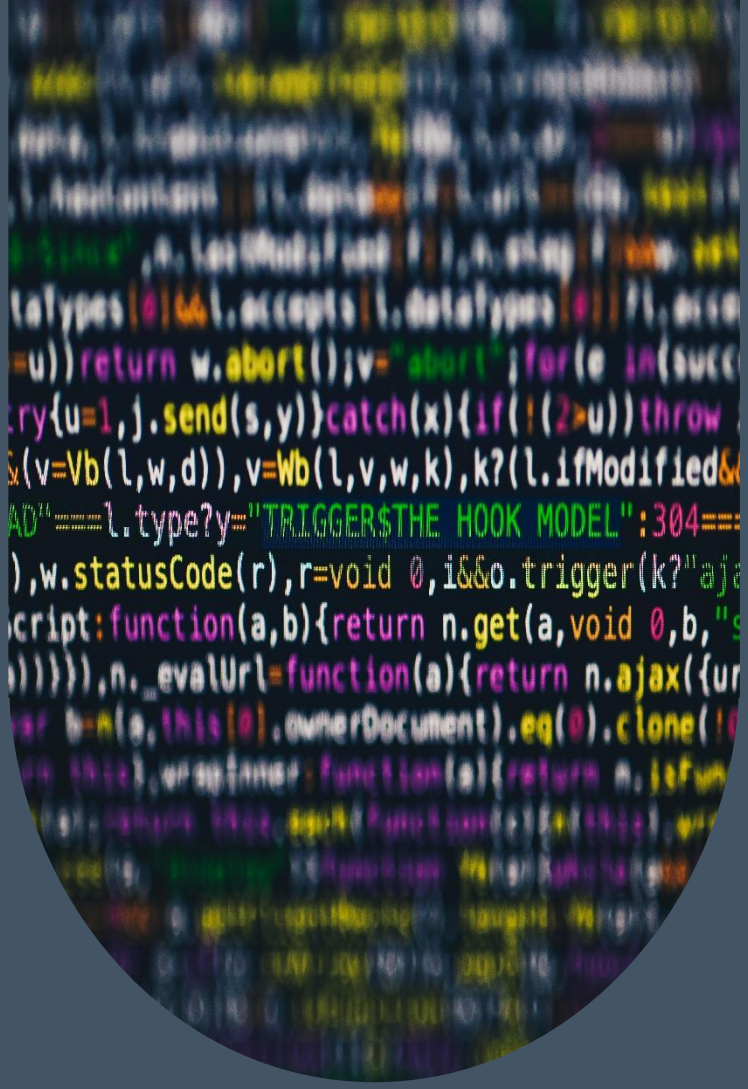
Games ENG II
Chapter 3,
Logic Coverage
Examples



Dr. Omer Ali

Adapted from
Introduction to Software Testing (Ammann & Offutt)

SETU
Ollscoil Teicneolaíochta an Oirdheiscirt
South East Technological University

setu.ie
INSPIRING FUTURES

Lecture 5 summary

Can Graph Coverage identify bugs?

```
#include <iostream>// Function to determine user access
bool hasAccess(bool isAuthenticated, bool isAdmin, bool isVip) {
        if (isAuthenticated && (isAdmin || isVip))
                {
                  return true;    }
          return false;
}
```

We need only two test cases to get 100% branch coverage: one that enters the if block (**true** branch) and one that skips it (**false** branch).

Test Case 1 : True branch

isAuthenticated = true, isAdmin = true, isVip = false

Result = hasAccess (true) → Test Passed

Test Case 2 : False branch

isAuthenticated = false, isAdmin = true, isVip = false

Result = hasAccess (false) → Test Passed

Based on that we have 100% branch coverage.

Testing with Logic Coverage?

- Forcing all logical combinations to identify flaws/bugs
- This is sometimes referred to as Multiple Condition Coverage (MCC)
- MCC requires testing all Boolean conditions within a scope (predicate, more to that later)

Test Case 3 : MCC Coverage

isAuthenticated = false, isAdmin = false, isVip = true

Result = hasAccess (false) → Test Passed

- **The Problem:** The test for a VIP user who is not authenticated ***failed***.
- **Expected:** true (if a user is a VIP, they should have access regardless of authentication status).
- **Actual:** false.
- **Conclusion:** The code has a bug! Logic coverage forced us to create a test case that exposed the defect.

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

  ¬ : the *negation* operator

  ∧ : the *and* operator

  ∨ : the *or* operator

  → : the *implication* operator

  ⊕ : the *exclusive or* operator

  ↔ : the *equivalence* operator
- A *clause* is a predicate with no logical operators

  ((a **<** b) ∧ C) ∨ ((a **>=** b) ∧ p(x))

Review: Truth Table for Logical Operators

p	q	$p \wedge q$	$p \vee q$	$p \oplus q$	$p \rightarrow q$	$p \leftrightarrow q$
T	T	T	T	F	T	T
T	F	F	T	T	F	F
F	T	F	T	T	T	F
F	F	F	F	F	T	T

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

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

predicate coverage

Predicate = true

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

= (5 < 10) ∨ true ∧ (1 >= 1*1)

= true ∨ true ∧ TRUE

= true

Predicate = false

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

=(10< 5) ∨ false ∧ (1>=1*1)

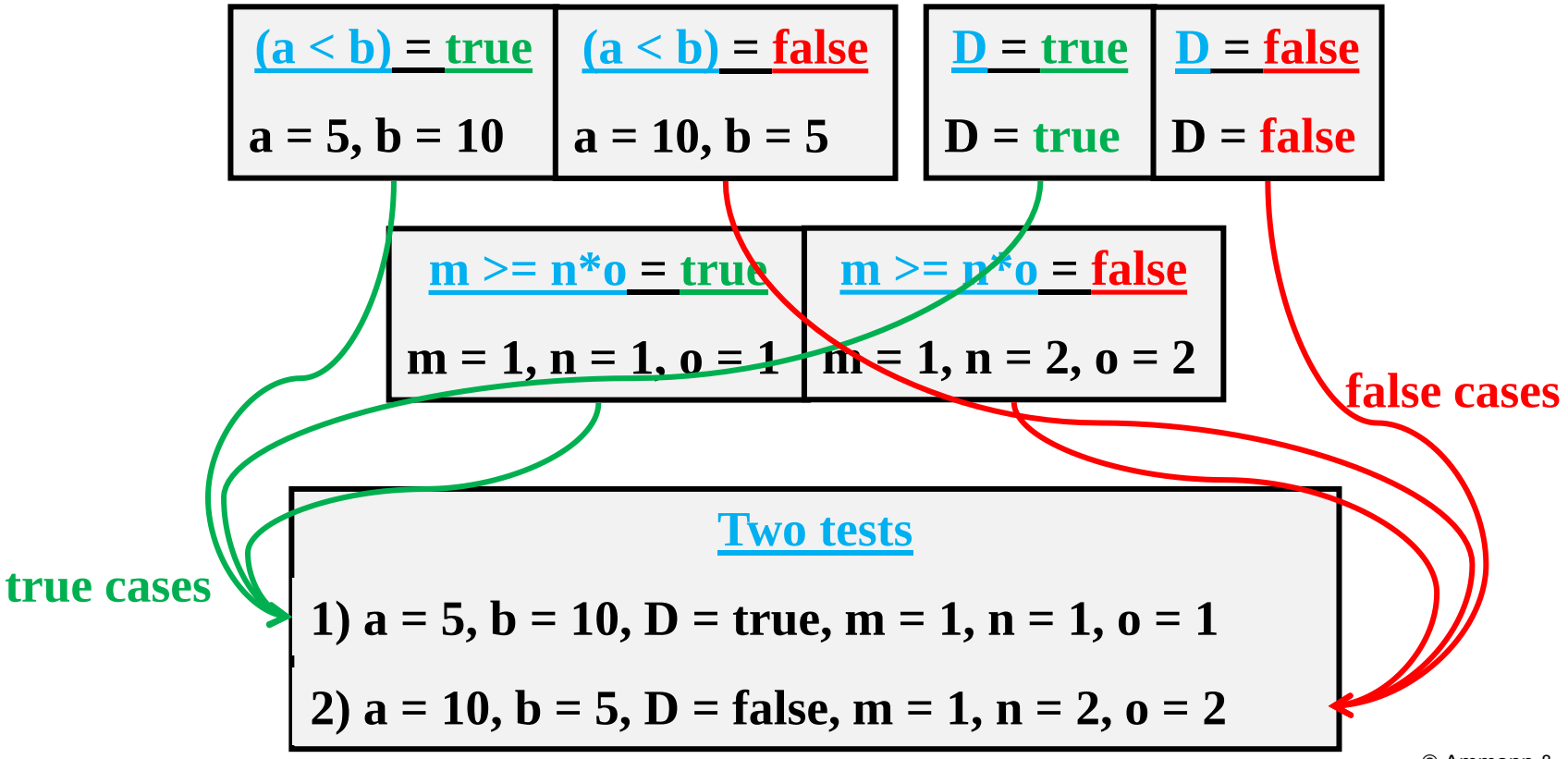
= false ∨ false ∧ TRUE

= false

Clause Coverage Example

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

Clause coverage



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.

	a < b	D	m >= n*o	P = ((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

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: $(5 < 10) \vee true \wedge (1 >= 1*1)$
- 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 that determines p *iff* 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*

Active Clauses

```
int HonorRollStatus (double CGPA, double termGPA, int creditsCompleted, Boolean fullTimeStatus)
{
   if ((creditsCompleted > 30) && (CGPA >3.4) && (fullTimeStatus == true) && (termGPA >2.5)){
   cout<<"Made it to the Dean's list"<<endl;}
   else if ((creditsCompleted > 30) && (CGPA >3.7) && (fullTimeStatus == true) && (termGPA > 2.5)){
   cout<<"The student is on High honor list"<<endl;}
   else if ((creditsCompleted > 30) && (CGPA > 2.5) && (fullTimeStatus == true) && (termGPA > 2.5)){
    cout<<"You're on honors list");}
    else{
     cout<<"You're on a 'hit list' somewhere"<<endl;
  }
}
```

Where,
CC = creditsCompleted, FT = fullTimeStatus
TGPA = termGPA

CC	CGPA	FT	TGPA	Honor's List?
29	3.3	T	2.4	F
[illegible]	3.3	T	2.4	T
31	3.0	T	2.4	F
[illegible]	3.3	T	2.4	T
31	3.3	F	2.4	F
[illegible]	3.3	T	2.4	T
31	3.3	T	1.9	F
[illegible]	3.3	T	2.4	T

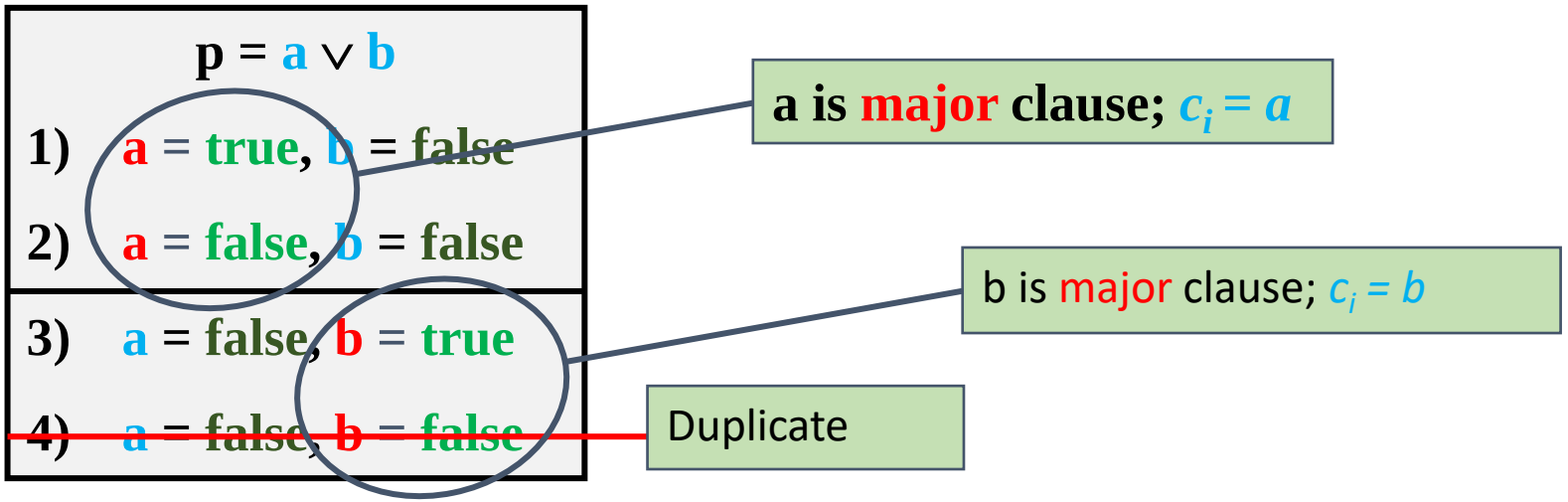
Determining Predicates

Determining Predicates

- Determination, the conditions under which a clause influences the outcome of a predicate
- If you flip the clause, and the predicate changes value, then the clause determines the predicate.
- The **major** clause, c_i , is the clause on which we are focusing.
- All of the other clauses c_j , j != i, are **minor** clauses.
- **Typically**, to satisfy a given criterion, each clause is treated in turn (*One at a time, One by one*) as a **major** clause.
- **From the testing perspective**, we would like to test each clause under circumstances where the clause determines the predicate

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 != 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 (*Modified Condition/Decision Coverage*), 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 (or $c_i = a$)

a = true, b = false, c = true

a = false, b = false, c = false

Is this allowed ?

- Is the minor clauses c_j need to have the same values when the major clause c_i is true as when c_i is false ???
- This question caused confusion among testers for years
- Considering this carefully leads to three separate criteria :
  1. Minor clauses do not need to be the same (GACC)
  2. Minor clauses do need to be (must be) the same (RACC)
  3. Minor clauses force the predicate to become both true and false (CACC)

General Active Clause Coverage

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 != 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.

Condition: 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
$c_j(c_i = true)$!= $c_j(c_i = false)$ for all c_j

- This is complicated !
- It is possible to satisfy GACC without satisfying predicate coverage (PC) . i.e. p = a ↔ b
- We really want to cause predicates to be both true and false !

Restricted Active Clause Coverage

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 != 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.

Condition: 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 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 c_i in C_p, 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.

Condition: 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 $p(c_i = true)$!= $p(c_i = false)$.

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

Simple Example #1: GACC, RACC, CACC

	a	b	p = a ↔ b
1	T	T	T
2	T	F	F
3	F	T	F
4	F	F	T

NOTE: For simple predicates, RACC and CACC might be the same. But, they sure differ for complex (bigger) predicates

GACC:

Remember: 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

GACC for $c_i = a$: $T_{14} = \{TT, FF\}$;

GACC for $c_i = b$: $T_{14} = \{TT, FF\}$,

GACC = $T_{14} \cup T_{14} = \{TT, FF\}$; This satisfies CC but not PC

RACC:

Remember: The values chosen for the minor clauses c_j must be the same when c_i is true as when c_i is false

RACC for $c_i = a$: $T_{13} = \{TT, FT\}$

RACC for $c_i = b$: $T_{12} = \{TT, TF\}$

RACC = $T_{13} \cup T_{12} = \{TT, FT, TF\}$; This satisfies CC & PC

CACC:

Remember: 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

CACC for $c_i = a$: $T_{13} = \{TT, FT\}$;

CACC for $c_i = b$: $T_{12} = \{TT, TF\}$

CACC = $T_{13} \cup T_{12} = \{TT, FT, TF\}$; This satisfies CC & PC

Example #2 : CACC *vs.* RACC

	a	b	c	p = a ∧ (b ∨ c)
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F

major clause

CACC can be satisfied by choosing any of rows 1, 2, 3 AND any of rows 5, 6, 7 – a total of nine pairs

TR = any pair of {1,2,3} X {5,6,7}

	a	b	c	p = a ∧ (b ∨ c)
1	T	T	T	T
5	F	T	T	F
2	T	T	F	T
6	F	T	F	F
3	T	F	T	T
7	F	F	T	F

major clause

RACC can only be satisfied by one of the three pairs above

TR = any pair of { (1,5), (2,6), (3,7) }

Example #3 : CACC *vs.* RACC

Predicate: (**A** ∧ **B**) ∨ **C**

•For **A** to matter: *B must be TRUE, C must be FALSE*
Because then the decision depends on whether A is true.
•For **B** to matter: *A must be TRUE, C must be FALSE*
•For **C** to matter: *both A and B must be FALSE*
Because then the A && B part is false, and only C can change the outcome.

Active Clause	Minor Clause	Test 1 (TRUE test case)	Test 2 (False test case)
A	B = TRUE, C = FALSE	(A = TRUE, B= TRUE, C= FALSE) → p (TRUE)	(A = FALSE, B= TRUE, C= FALSE) → p (FALSE)
B	A = TRUE, C = FALSE	(A = TRUE, B= TRUE, C= FALSE) → p (TRUE)	(A = TRUE, B= FALSE, C= FALSE) → p (FALSE)
C	A = FALSE, B = FALSE	(A = FALSE, B= FALSE, C= TRUE) → p (TRUE)	(A = FALSE, B= FALSE, C= FALSE) → p (FALSE)

These satisfy both CACC and RACC, but how?

Example #4 : Real-world Payment authorization

An online payment gateway authorizes payment iff:

1. Card is valid (number & expiry & not blocked)
2. Strong authentication passed (not blocked) ,
3. And either **risk is low** or **manual override** flag is set by operations.

Let the clauses be:

C1 : cardValid

C2: authPassed

C3: riskLow (e.g. riskScore < 75)

C4: override (manual flag) – usually overridden for small purchases or frequent purchases with the same vendor etc.

Predicate: (**c1** $\wedge$ **c2**) $\wedge$ (**c3** $\vee$ **c4**) $\xrightarrow[\text{TRUE}]{}$ Payment approved

Example #4 : Real-world Payment authorization

C1 : cardValid
C2: authPassed
C3: riskLow (e.g. riskScore < 75)
C4: override (manual flag)

Predicate: (**c1** ∧ **c2**) ∧ (**c3** ∨ **c4**) —TRUE→ Payment approved

Predicate Coverage (required predicate to be TRUE and FALSE at least once)
Test 1 (**True** case): (c1,c2,c3,c4) = (T,T,T,F) → approve (True)
Test 2 (**False** case): (c1,c2,c3,c4) = (F,F,F,F) → approve (False)

Clause Coverage (requires the clause to flip across set)
Test 1 (**True** case): (c1,c2,c3,c4) = (T,T,T,F) → approve (True)
Test 2 (**False** case): (c1,c2,c3,c4) = (F,T,T,F) → approve (False)

Test 1 (**True** case): (c1,c2,c3,c4) = (T,T,T,F) → approve (True)
Test 2 (**False** case): (c1,c2,c3,c4) = (T,F,T,F) → approve (False)

Test 1 (**True** case): (c1,c2,c3,c4) = (T,T,T,F) → approve (True)
Test 2 (**False** case): (c1,c2,c3,c4) = (T,T,F,F) → approve (False)

Test 1 (**True** case): (c1,c2,c3,c4) = (T,T,F,T) → approve (True)
Test 2 (**False** case): (c1,c2,c3,c4) = (T,T,F,F) → approve (False)

Example #4 : Real-world Payment authorization

Predicate: (**c1** $\wedge$ **c2**) $\wedge$ (**c3** $\vee$ **c4**) $\xrightarrow{\text{TRUE}}$ Payment approved

Now determining Active Clauses

Active Clause	Minor Clause	Minor Clause	Minor Clause	Predicate
C1	C2 = TRUE	C3 = FALSE	C4 = TRUE	TRUE
C2	C1 = TRUE	C3 = FALSE	C4 = TRUE	TRUE
C3	C1 = TRUE	C2 = TRUE	C4 = FALSE	TRUE
C4	C1 = TRUE	C2 = TRUE	C3 = FALSE	TRUE

Bonus Points (+2) – All correct explanations received by 5 PM today

Example #4 : Real-world Payment authorization

Predicate: (**c1** $\wedge$ **c2**) $\wedge$ (**c3** $\vee$ **c4**) $\xrightarrow{\text{TRUE}}$ Payment approved

GACC Pairs (where minor clauses ***may*** differ)

C1 major (fix c2 = TRUE), make right TRUE via **override flag c4**
(T, T, -, T) **Vs** (F, T, -, T) → p flips

C2 major (fix c1 = TRUE), make right TRUE via **override flag c4**
(T, T, -, T) **Vs** (T, F, -, T) → p flips

C3 major (fix c1, c2 = TRUE), now right side is handled by c3, so set c4 = FALSE
(T, T, T, F) **Vs** (T, T, F, F) → p flips

C4 major (fix c1, c2 = TRUE), now right side is handled by c4, so set c3 = FALSE
(T, T, F, T) **Vs** (T, T, F, F) → p flips

Bonus Points (+3) – Can we also call this CACC, if so, why?

Example #4 : Real-world Payment authorization

Predicate: (**c1** $\wedge$ **c2**) $\wedge$ (**c3** $\vee$ **c4**) —TRUE→ Payment approved

RACC Pairs (where minor clauses ***are identical***)

C1 major (fix c2 = TRUE), make right TRUE via **override flag c4**
(T, T, *, T) **Vs** (F, T, *, T) → p flips

C2 major (fix c1 = TRUE), make right TRUE via **override flag c4**
(T, T, *, T) **Vs** (T, F, *, T) → p flips

C3 major (fix c1, c2 = TRUE), now right side is handled by c3, so set c4 = FALSE
(T, T, T, F) **Vs** (T, T, F, F) → p flips

C4 major (fix c1, c2 = TRUE), now right side is handled by c4, so set c3 = FALSE
(T, T, F, T) **Vs** (T, T, F, F) → p flips

Bonus Points (+3) – Can we also call this CACC, if so, why?

Example #4 : Real-world Payment authorization

Predicate: (**c1** $\wedge$ **c2**) $\wedge$ (**c3** $\vee$ **c4**) —TRUE→ Payment approved

```
#include <cassert>

struct Card { bool expiryFuture, luhnOk, blocked; };

bool approvePayment(const Card& card, bool otpOk, bool tdsOk,
                    int riskScore, bool overrideFlag) {
    bool c1 = card.expiryFuture && card.luhnOk && !card.blocked;
    bool c2 = (otpOk || tdsOk);
    bool c3 = (riskScore < 70);
    bool c4 = overrideFlag;
    return (c1 && c2) && (c3 || c4);
}

Int main ( ) {
Assert (approvePayment({true,true,false}, true,false,10,false) == true);
Assert (approvePayment({false,true,false}, false,false,94,false));

}
```

Logic Coverage Summary

Coverage Type	Ensures	Goals	Remarks
PC	Predicate is TRUE & FALSE	Very coarse	May miss logic errors
CC	Each clause TRUE & FALSE	Better	Doesn't ensure influence
GACC	Each clause determines predicate (loosely)	Easy to achieve	Others can vary
CACC	Clause determines predicate, result changes	Best practical coverage	Used in MC/DC
RACC	Only major clause changes	Very strict	Sometimes infeasible