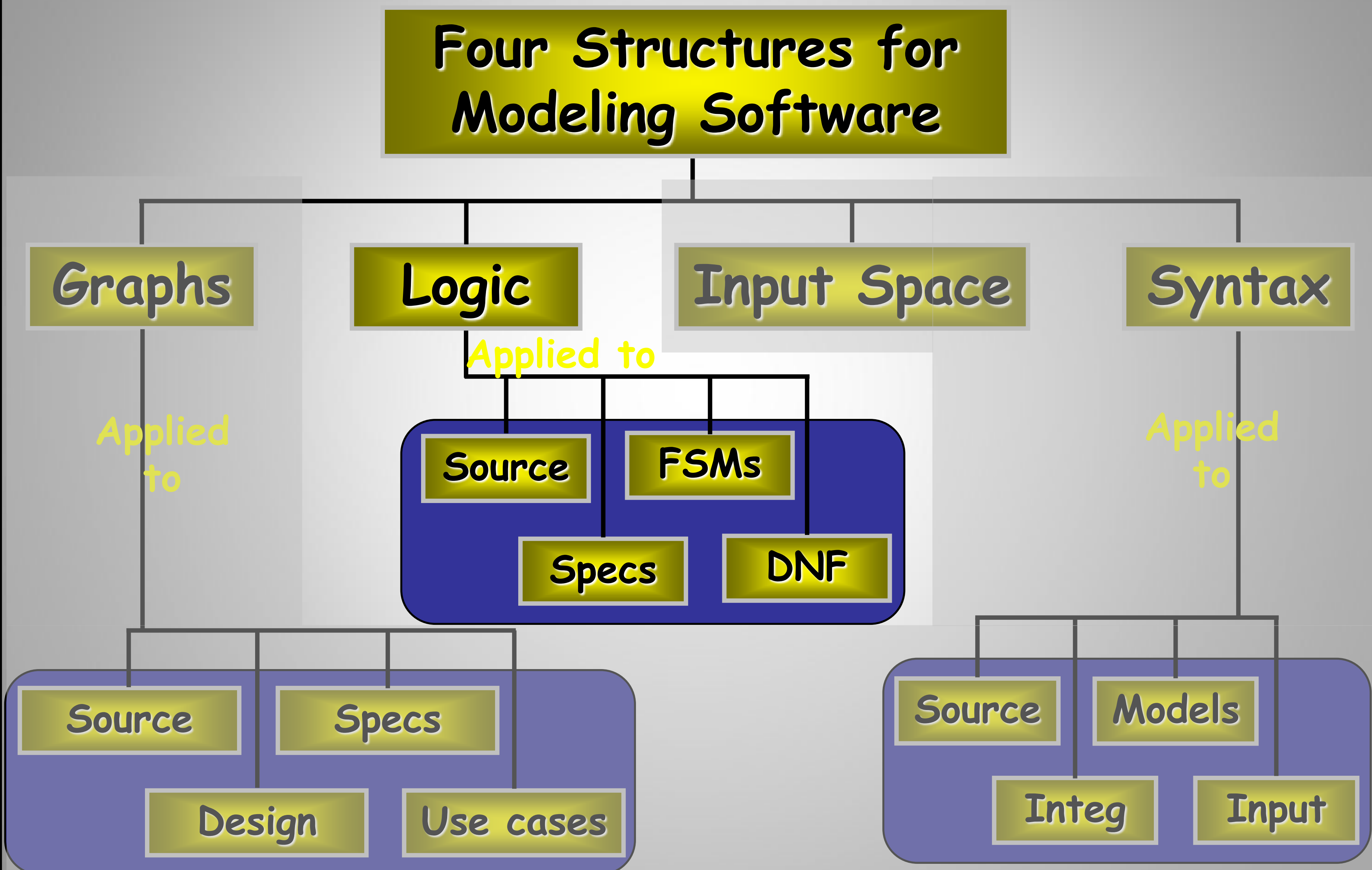
Introduction to Software Testing

Chapter 3.1, 3.2
Logic Coverage

Paul Ammann & Jeff Offutt

Updated by Sunae Shin

Ch. 3 : Logic Coverage



Covering Logic Expressions

- 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 >, <, ==, >=, <=, !=
    - boolean function calls
  - Internal structure is created by logical operators
    1. $\neg$ – the *negation* operator
    2. $\wedge$ – the *and* operator
    3. $\vee$ – the *or* operator
    4. $\rightarrow$ – the *implication* operator
    5. $\oplus$ – the *exclusive or* operator
    6. $\leftrightarrow$ – the *equivalence* operator
- A ***clause*** is a predicate with **no logical operators**

Examples

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

- Four clauses:
  - (a < b) – relational expression
  - f (z) – boolean-valued function call
  - D – boolean variable
  - (m >= n*o) – relational expression
- Sources of predicates
  - Decisions in programs
  - Guards in finite state machines
  - Decisions in UML activity graphs
  - Requirements, both formal and informal

Translation

- **Translating from source code**

```
if ((a > b) || C) && (x < y)
   o.m();
else
   o.n();
```

$$((a > b) \vee C) \wedge (x < y)$$

- **Translating from precondition in a specification**

  " pre: stack Not full AND object reference parameter not null"

  $$\neg\, stackFull(\,) \wedge newObj \neq null$$

Logic Expression 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.

- Example in next page…

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

More Example – Predicate Coverage

$$((a > b) \vee C) \wedge p(x)$$

- Two tests that satisfy predicate coverage

  (a = 5, b = 4, C = true, p(x) = true) and

  (a = 5, b = 6, C = false, p(x) = false)

- **Problem!!**

  (a = 5, b = 4, C = true, p(x) = true) and

  (a = 5, b = 4, C = true, p(x) = false)

→ The individual clauses are not always exercised.

  - first two clauses never have the value false!

Predicate and Clause 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.

- Example in next page..

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

false cases

true cases

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

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 C_p to evaluate to each possible combination of truth values.

	a < b	D	m >= n*o	$((a < b) \vee D) \wedge (m >= n*o)$
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, 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 general idea is simple:

Test each clause independently from the other clauses

- What exactly does "independently" mean ?
  - The book presents this idea as "*making clauses active*" …

Active Clauses

- Clause coverage has a <u>weakness</u> : The values do not always make a difference
- To really test the results of a clause, the clause should be the <u>determining factor</u> in the value of the predicate

<u>Determination</u> :

A clause C_i in predicate *p*, called the <u>major clause</u>, <u>determines</u> *p* if and only if the values of the remaining <u>minor clauses</u> C_j are such that changing C_i changes the value of *p*

- Simply, <u>if you flip the clause, and the predicate changes value, then the clause ***determines*** the predicate</u>
- This is considered to ***make the clause active***

Determining Predicates

$P = A \vee B$

if $B = true$, p is always true.

so if $B = false$, A determines p.

if $A = false$, B determines p.

$P = A \wedge B$

if $B = false$, p is always false.

so if $B = true$, A determines p.

if $A = true$, B determines p.

- **If we do not vary *b* under circumstances where *b* determines *p*, then we have no evidence that *b* is used correctly**
- Goal : **Find tests for each clause when the clause determines the value of the predicate**

Active Clause Coverage

<u>Active Clause Coverage (ACC)</u> : 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.

<u>p = a ∨ b</u>

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

a is major clause

b is major clause

Duplicate

- <u>Ambiguity</u> : 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 for years
- Considering this carefully leads to three separate criteria :
  - Minor clauses <u>do not</u> need to be the same
  - Minor clauses <u>do</u> need to be the same
  - Minor clauses <u>force the predicate</u> to become both true and false

General Active Clause Coverage

<u>General Active Clause Coverage (GACC)</u> : For each p in P and each major clause c_i in Cp, choose minor clauses c_j, j != i, so that ci determines p. TR has two requirements for each c_i : c_i evaluates to true and c_i evaluates to false. The values chosen for the minor clauses c_j do <u>not</u> 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.

Restricted Active Clause Coverage

<u>Restricted Active Clause Coverage (RACC)</u> : 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. The values chosen for the minor clauses c_j <u>must be the same</u> 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.

Correlated Active Clause Coverage

<u>Correlated Active Clause Coverage (CACC)</u> : 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. The values chosen for the minor clauses c_j must <u>cause *p* to be</u> 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
  - Minor clauses <u>force the predicate</u> to become both true and false

CACC and RACC

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

P_a : 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

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

$p = a \vee b$

$p_a = p_{a=true} \oplus p_{a=false}$

$= (true \vee b)$ XOR $(false \vee b)$

$= true$ XOR b

$= \neg b$

- b must be false (a determine the predicate)

$p = a \wedge b$

$p_a = p_{a=true} \oplus p_{a=false}$

$= (true \wedge b) \oplus (false \wedge b)$

$= b \oplus false$

$= b$

- b must be true to make a determine p

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

$p_a = p_{a=true} \oplus p_{a=false}$

$= (true \vee (b \wedge c)) \oplus (false \vee (b \wedge c))$

$= true \oplus (b \wedge c)$

$= \neg (b \wedge c)$

$= \neg b \vee \neg c$

- **"*NOT b* ∨ *NOT c*" means either *b* or *c* can be false**
- **RACC requires the same choice for both values of *a*, CACC does not**

Logic Coverage Summary

- **Predicates are often very simple—in practice, most have less than 3 clauses**
  - **With only clause, PC is enough**
  - **With 2 or 3 clauses, CoC is practical**
- **Control software often has many complicated predicates, with lots of clauses**

Logical Conjunction

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

Logical Disjunction

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

Logical Implication

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Exclusive Disjunction

p	q	$p \oplus q$
T	T	F
T	F	T
F	T	T
F	F	F