Unit 8. Combinational Testing

1. Combinational Test Models
2. Decision Tables
3. Logic Functions
4. Variable Negation Test Strategy
5. Non-binary Variable Domain Analysis

TB: Chapter 9 (9.6)

1. Combination Test Models

- Rely on knowledge acquired in hardware design and testing.
  - Test strategies are ***extensions of existing hardware testing*** strategies for software applications.
  - Can be used for ***any test scope***, from unit to system levels.

-Mostly used for test targets (e.g., function, system, etc.) where there is ***no state information***, or the responses depend only on current values and not on past input or output.

-Use **decision tables** to describe the inputs and outputs of the application under testing.

- Prior to generating test data, a decision table ***must be validated*** in order to remove inconsistencies and omissions.
- Then, a ***logic function is derived***, and used to generate test cases following suitable test strategies (e.g., **variable negation strategy**, domain analysis).

2. Decision Tables

-Used as test model in cases where there is no state information, or the responses depend only on current values and not on past input or output.

Example: Insurance Renewal

Decision table modeling the requirements for processing the annual renewal of a hypothetical auto insurance policy. The table specifies the business rules used to cancel or renew a policy, and determine the premium.

Variants	Conditions		Actions		
	Number of claims	**Insured Age**	**Premium increase ($)**	**Send warning**	**Cancel**
1	0	25 or younger	50	No	No
2	0	26 or older	25	No	No
3	1	25 or younger	100	Yes	No
4	1	26 or older	50	No	No
5	2 to 4	25 or younger	400	Yes	No
6	2 to 4	26 or older	200	Yes	No
7	5 or more	Any	0	No	Yes

Faults to Catch in Decision Table

- Incorrect value assigned to a decision table
- Incorrect or missing operator in a predicate
- Incorrect or missing variable in a predicate
- Incorrect structure in a predicate (dangling else, etc.)
- Incorrect or missing default case
- Incorrect or missing actions
- Extra actions
- Structural errors in table 's implementation
- Missing or incorrect class or method signature when variants are implemented by dynamic binding
- Generic errors: wrong versions, ambiguous reqs, incorrect or missing specification item

3. Logic Functions

-A logic function is ***derived from the decision table***, and used to ***generate test cases*** by following appropriate test strategies.

-A logic function is a Boolean function mapping n Boolean input variables to m Boolean output variables.

- Logic functions, are implemented traditionally, in hardware, as digital circuits using ***nand*** gates.

-A logic function can be represented in an enumerative form using a ***truth table***, or in a more compact form as a ***boolean expression***.

-A truth table is appropriate in testing situations, where the inputs/outputs variables are binary, so can be represented as boolean factors.

-In cases, where ***complex conditions and actions*** are involved, truth tables are not effective; ***instead decision tables should be used***.

Truth Tables

-A truth table gives an enumeration of all possible values of a logic function.

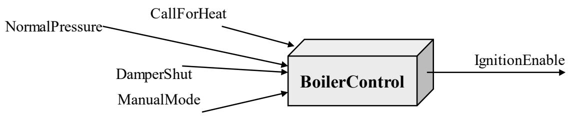
- It is a particular kind of decision table in which the inputs/outputs variables are binary, so can be represented as boolean factors.

-A table with n variables always contains 2^n rows.

- The inputs are boolean values
- The decimal representation of each input vector corresponds to the number of the row
- The outputs are derived from the logical combination of current inputs.

Example: Boiler Control System

Consider the control action to enable or disable ignition of a Boiler.



- The following parameters are involved:

-*CallForHeat* is on when the ambient temperature falls below the set point

-*NormalPressure* is true if the pressure in the boiler is within safe operating limits

-*DamperShut* is on when the exhaust duct is closed

-*ManualMode* is true when manual operation has been selected

-*IgnitionEnable* is the output of the system.

Truth Values for the Boiler Control

Truth value	NormalPressure (A)	CallForHeat (B)	DamperShut (C)	ManualMode (D)	IgnitionEnable (Z)
0	No	Off	No	No	Off
1	Yes	On	Yes	Yes	On

Truth Table for the Boiler Control

Input vector number	NormalPressure (A)	CallForHeat (B)	DamperShut (C)	ManualMode (D)	Ignition Enable (Z)
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

Logic Minimization

-From testing perspective, a compact representation of the logic function is preferred because it ensures test efficiency.

-Logic minimization is the process of deriving compact representation for the logic function.

-Several minimization techniques are available, each appropriate for specific size of the truth table:

Logic Minimization Techniques	Approximate Upper Limit on Number of Boolean Variables
Inspection	3
KV matrix	5
Cause-effect graph	8
Quine-McKluskey method	9
Starner-Dietmeyer	14
Exact minimization	20-25
Heuristic minimization	20-25

Basic Notions of Boolean Logic

-***Literal***: occurrence of an individual variable in a boolean expression.

-***Product term:*** string of literals related by a logical ***and*** operator.

-***Sum-of-product:*** sequence of product terms related by logical ***or***.

-***Implicant***: term in a sum-of-products expression.

-***Minterm***: row in a truth table that evaluates to true (output):

- logical expression containing a literal for each variable in a logic function, which evaluates to true for that function.
- can be derived by analyzing the truth table or by expanding sum-of-products terms.

Karnaugh-Veitch (KV) Matrix

-KV matrix is a graphical logic minimization technique.

- KV matrices ***vary according to the number of inputs***.
- They are suitable for functions involving not more than ***five*** variables.

-In a KV matrix:

- Rows and columns correspond to one or two variables;
- Each ***cell contains the output value*** of corresponding input vector.
- In some representations, a cell may contain the output value corresponding to the input vector, while in other representations the cell may contain the number (or decimal representation) of the input vector.

Two-variable KV matrix: Z= F(AB)

B \ A	0	1
0	Z(00)	Z(10)
1	Z(01)	Z(11)

B \ A	0	1
0	0	2
1	1	3

Three-Variable KV Matrix: Z=F(ABC)

C \ AB	0 0	01	11	10
0	Z(000)	Z(010)	Z(110)	Z(100)
1	Z(001)	Z(011)	Z(111)	Z(101)

C \ AB	0 0	01	11	10
0	0	2	6	4
1	1	3	7	5

Four-Variable KV Matrix: Z=F(ABCD)

CD \ AB	0 0	01	11	10
00	Z(0000)	Z(0100)	Z(1100)	Z(1000)
01	Z(0001)	Z(0101)	Z(1101)	Z(1001)
11	Z(0011)	Z(0111)	Z(1111)	Z(1011)
10	Z(0010)	Z(011)	Z(111)	Z(1010)

CD \ AB	0 0	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

Steps to build a KV matrix

1. Identify input/output variables from the truth table; the number of input variables corresponds to the size of the KV matrix.
2. Set up the initial matrix. Write 1 in the cell for each corresponding implicant.
3. Find the largest group of adjacent 1 cells.
4. Transcribe the product terms for this group.
5. Repeat steps 3 and 4 until all valid groups have been identified and transcribed. The minimal logic function is the sum of the product terms.

-***Adjacent cell groups*** are formed by the following rules:

- An adjacent group is 2, 4, or 8 cells. Groups are formed from numbers of cells that are powers of two (e.g., 2, 4, 8, 16 etc.).
- An adjacent group is formed horizontally or vertically. Groups are not formed on any diagonal.
- In a matrix of three or more variables, an adjacent group may wrap around the right and left edges.
- In a four-variable matrix, an adjacent group may wrap around the right and left edges or around the top and bottom edges.
- Groups may overlap

-The product term is transcribed one variable at a time by considering the truth values for each cell's row and column:

- If the group covers only ***0 values for a variable, a negated literal*** is transcribed.
- If the group covers only ***1 values for this variable, the literal*** is transcribed.
- If the group covers both ***0 and 1 values for this variable, no literal*** is transcribed.

Example: Boiler control model

1. Set up initial table: write 1 in the cell for each corresponding implicant

CD \ AB	00	01	11	10
00			1	
01			1	1
11			1	1
10				

2. Find largest adjacent group; transcribe product term for this group

CD \ AB	00	01	11	10
00			1	
01			1	1
11			1	1
10				

A=1, B={0,1}, C={0,1}, D=1, AD=AxxD

3. Find next largest adjacent group; transcribe product term for this group

CD \ AB	00	01	11	10
00			1	
01			1	1
11			1	1
10				

A=1, B=1, C=0, D={0,1}

4. No more groups. Minimal logic Function is the sum of the product terms.

Z= AB~C + AD

Z= AB~C + AD

Logically we can then say:
Ignition is enabled when either
(1) the boiler is at NormalPressure, CallForHeat is on, and DamperShut is not true (AB~C is true), or
(2) the boiler is at NormalPressure and ManualMode is on (AD is true).

What do you think the logic functions is? Is there an easier way?

Possible implementation:

```
if ( (normalPressure && callForHeat && !damperShut) ||
     (normalPressure && manualMode) ) {
          ignitionEnable = true;
} else {
          ignitionEnable = false;
}
```


4. Variable Negation Test Strategy

-Various test strategies can be used to generate test suites from a valid decision table; the ***variable negation test strategy*** is one of the most powerful available.

-Variable negation is a sophisticated strategy that generates test suites based on characteristics of special kinds of variants, called ***test candidate sets*** (*the rows from the decision/truth tables*).

-Two special kinds of variants are considered: ***unique true points*** and ***near false points***:

- ***Unique true point***: a variant that makes a ***product term true but no other term true*** in a sum-of-products boolean formula. **F = xyz + b!xy + a!x!y**
- ***Near false point***: a variant, containing ***a negated literal from a product term*** which makes ***the logic function evaluates to 0*** for the negated term.

-Given a term in a sum-of-products, the near false points are identified by negating each literal in the term, one-by-one.

!xyz
x!yz
xy!z

-Test candidate sets are generated for each product term, and used to select test suites.

-The steps for test suite generation are:

1. Identify the ***sum-of-products*** for the logic function.
2. For each term in the sum-of-products, identify the ***unique true points*** and the ***near false points***.
3. Label the ***candidate test sets*** corresponding to the generated points.
4. Test suites are generated by ***selecting at least one variant from each candidate test set***. Selection can be done randomly, or using some heuristics.

Given F = xyz + b!xy + a!x!y we will have four possible tests for each product (for example, xyz) consisting of the unique true cases and then three unique negations (see previous page). Therefore 12 unique tests.

Variant/Candidate matrix

-Represent a systematic way to ***organize test cases*** generated using ***variable negation strategy***.

-The matrix includes three kinds of columns:

- a column listing variants,
- a set of columns listing the different test candidate sets,
- a column listing selected test cases.

Variant	Test Candidate Set						Test Case
	1	2	3	…	q-1	q	
0							
1			x				√
…							…
p-1	x						√
p	x	x					

-The cells may be empty or include checkmarks in case where some variants belong to some candidate sets

Example: Boiler Control

Z= AB~C + AD

Product term AB~C

Product term negation	Test candidate sets	Candidate set number
A B ~C	12 (UTP)	1
A B C	14 (NFP)	2
A ~B ~C	8 (NFP)	3
~A B ~C	4, 5 (NFP)	4

Product term AD

Product term negation	Test candidate sets	Candidate set number
A D	9, 11, 15 (UTP)	5
A ~D	8, 10, 14 (NFP)	6
~A D	1, 3, 5, 7 (NFP)	7

Note:
UTP: Unique true Point
NFP: Near False Point

Variant/ Candidate Matrix

Variant	Test Candidate Set							Test Case
	1	2	3	4	5	6	7	
0								
1							x	
2								
3							x	
4				x				
5				x			x	√
6								
7							x	
8			x			x		√
9					x			√
10						x		
11					x			
12	x							√
13								
14		x				x		√
15					x			

Note:
Unique true Points:
-AB~C: 12
-AD: 9, 11, 15

5. Non-binary Variable Domain Analysis

-Test data may be generated for non-binary decision table variables.

- Each variant with non-binary decision variables defines a ***sub-domain***.
- Typically each sub-domain ***corresponds to the conditions defining the variant*** (although this is not an absolute requirement).
- The minimal domain test strategy is to pick ***one on point*** and ***one off point*** per boundary.

Example: Insurance Renewal System

-Assuming that the age must be between 16 and 85, and that no more than 10 claims would ever be paid, seven sub-domains are identified:

- Exactly 0 claims, age 16-25
- Exactly 0 claims, age 26-85
- Exactly 1 claim, age 16-25
- Exactly 1 claim, age 26-85
- Two, 3, or 4 claims, age 16-25
- Two, 3, or 4 claims, age 26-85
- Five to 10 claims, age 16-85

V1

Variable	Condition	Type	Test cases						
			1		2	3	4	5	
Number of claims	==0	On	0						
		Off (above)		1					
		Off (below)			-1				
	Typical	In				0	0	0	0
Insured age	≥16	On				16			
		Off					15		
	≤25	On						25	
		Off							26
	Typical	In	20	20	20				
Expected Result			Acc	V3	Rej	Acc	Rej	Acc	V2
Premium increase			50	100		50		50	25
Send warning			No	Yes		No		No	No
Cancel			No	No		No		No	No

V2

Variable	Condition	Type	Test cases						
			6		7	8		9	10
Number of claims	==0	On	0						
		Off (above)		1					
		Off (below)			-1				
	Typical	In				0	0	0	0
Insured age	≥26	On				26			
		Off					25		
	≤85	On						85	
		Off							86
	Typical	In	32	49	64				
Expected Result			Acc	V4	Rej	Acc	V1	Acc	Rej
Premium increase			25	50		25	50	25	
Send warning			No	No		No	No	No	
Cancel			No	No		No	No	No	

V3

Variable	Condition	Type	Test cases						
			11			**12**	**13**	**14**	
Number of claims	==1	On	1						
		Off (above)		2					
		Off (below)			0				
	Typical	In				1	1	1	1
Insured age	≥16	On				16			
		Off					15		
	≤25	On						25	
		Off							26
	Typical	In	19	19	19				
Expected Result			Acc	V5	V5	Acc	Rej	Acc	V4
Premium increase			100	400	400	100		100	50
Send warning			Yes	Yes	Yes	Yes		Yes	No
Cancel			No	No	No	No		No	No

V4

Variable	Condition	Type	Test cases						
			15			**16**		**17**	**18**
Number of claims	==1	On	1						
		Off (above)		2					
		Off (below)			0				
	Typical	In				1	1	1	1
Insured age	≥26	On				26			
		Off					25		
	≤85	On						85	
		Off							86
	Typical	In	55	55	55				
Expected Result			Acc	V6	V2	Acc	V3	Acc	Rej
Premium increase			50	200	25	50	100	50	
Send warning			No	Yes	No	No	Yes	No	
Cancel			No	No	No	No	No	No	

V5

Variable	Condition	Type	Test cases							
			19		**20**		**21**	**22**	**23**	
Number of claims	>=2	On	2							
		Off		1						
	<=4	On			4					
		off				5				
	Typical	In					3	3	3	3
Insured age	≥16	On					16			
		Off						15		
	≤25	On							50	
		Off								26
	Typical	In	23	17	22	21				
Expected Result			Acc	V3	Acc	V7	Acc	Acc	Acc	V6
Premium increase			400	100	400	0	400	400	400	200
Send warning			Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
Cancel			No	No	No	Yes	No	No	No	No

V6

Variable	Condition	Type	Test cases							
			24		**25**		**26**	**27**	**28**	**29**
Number of claims	>=2	On	2							
		Off		1						
	<=4	On			4					
		off				5				
	Typical	In					3	3	3	3
Insured age	≥26	On					26			
		Off						50		
	≤85	On							85	
		Off								86
	Typical	In	83	27	36	44				
Expected Result			Acc	V4	Acc	V7	Acc	Acc	Acc	rej
Premium increase			200	50	200	0	200	200	200	
Send warning			Yes	No	Yes	No	Yes	Yes	Yes	
Cancel			No	No	No	yes	No	No	No	

V7

Variable	Condition	Type	Test cases							
			30		**31**	**32**	**33**	**34**	**35**	**36**
Number of claims	>=5	On	5							
		Off		4						
	<=10	On			10					
		off				11				
	Typical	In					6	7	8	9
Insured age	≥16	On								
		Off						15		
	≤85	On							85	
		Off								86
	Typical	In	58	18	43	29				
Expected Result			Acc	V5	Acc	Rej	Acc	Rej	Acc	Rej
Premium increase			0	400	0		0		0	
Send warning			No	Yes	No		No		No	
Cancel			Yes	No	Yes		Yes		Yes	