Test models: Combinational Models

Błażej Pietrzak
blazej.pietrzak@cs.put.poznan.pl

Combinational model representations

- Control list
- Decision table
- Decision tree
- Graphs
- Activity Diagram

Software testing with combinational models

1. Develop a model of capability or implementation being tested with decision table.
2. Validate decision table model.
3. Derive the logic function.
4. Choose a combinational test strategy.
5. Generate tests.

When to use decision tables

- ✓ One of several response is selected based on distinct input variables
- ✓ Cases can be modeled as mutually exclusive Boolean expressions (*pol. wzajemnie wykluczających się wyrażeń boolowskich)* of input variables
- ✓ Response does not depend on the order, in which input variables are initiated or computed
- ✓ Response does not depend upon prior input or output

How to develop a decision table

1. Identify decision variables and conditions.
2. Identify resultant actions to be selected.
3. Identify the actions to be produced in response to condition combinations.
4. Derive logic function to validate the completeness and cohesion of the model.

An example

6 conditions

Condition section			Action section		
Variant	Claims no.	Insured age	Premium Increase by (USD)	Send warning	Cancel
1	0	25 or less	50	No	No
2	1	26 or more	25	No	No
3	1	25 or less	100	Yes	No
4	0	26 or more	50	No	No
5	2 - 4	25 or less	400	Yes	No
6	2 - 4	26 or more	200	Yes	No
7	5 or more	any	0	No	Yes

Weinberg V.: Structured Analysis. Englewood Cliffs, NJ, Prentice-Hall 1978

Explicit and implicit variants

Decision table with n conditions can have up to 2^n variants

The example can have up to 64 variants

Explicit variant – a case that is modeled in decision table or truth table

The example have 7 explicit variants

Implicit variant – a case absent in the decision table or truth table, but can be deduced.

Decision tree

- Powiedzieć, że drzewa decyzyjne mogą być łatwiej zrozumiałe
- Mają tendencję do ukrywania wariantów
- Tabele decyzyjne są lepszym wyborem

Truth table

T – true; F – false; DC – don't care

	Decision variable	Variant							
		Condition	1	2	3	4	5	6	7
Condition section	Claims no.	0	T	T	F	F	F	F	F
		1	F	F	T	T	F	F	F
		2 – 4	F	F	F	F	T	T	F
		5 or more	F	F	F	F	F	F	T
	Insured age	25 or less	P	F	T	F	T	F	DC
		26 or more	F	T	F	T	F	T	DC
Action section	Premium Increase by 0		F	F	F	F	F	F	T
	Premium Increase by 25		F	T	F	F	F	F	F
	Premium Increase by 50		T	F	F	T	F	F	F
	Premium Increase by 100		F	F	T	F	F	F	F
	Premium Increase by 200		F	F	F	F	F	T	F
	Premium Increase by 400		F	F	F	F	T	F	F
	Send warning		F	F	T	F	T	T	F
	Cancel		F	F	F	F	F	F	T

Don’t care condition

- May be either true or false without changing the action
- Simplifies the decision table

Spot a don't care condition

```
if ((w > x) || (w > (y / z)))
```

Which can be don't care conditions?

Spot a don't care condition

```
if ((w > x) || (w > (y / z)))
```

If w > x then y and z can be don't care conditions.

Type-safe exclusion

```
int claims;
...
if (claims == 0) { ... }
else if (claims == 1) { ... }
else if ((claims >= 2) && (claims <= 4)) { ... }
else if (claims >= 5) { ... }
```

Type- safe exclusion - conditions defined for nonbinary decision variable, which fulfills each one of them with only one value.

Claims cannot be simultaneously 0 and 3.

Don’t know conditions

- Reflects incomplete model
- Example:
  Decision table does not define an action for Insured age = 300.
  It is unknown what will happen.

Can’t happen conditions

- Some inputs are mutually exclusive
- Some inputs cannot be produced by the environment
- Implementation is structured so as to prevent evaluation

Can't happen conditions – cont.

Number of claims can be simultaneously 0, 1, 2, 3, 4, 5 or more

```
if (isZeroClaims) { ... }
if (isOneClaim) { ... }
if (isTwoToFourClaims) { ... }
if (isFiveOrMoreClaims) { ... }
```

Decision tables in OO development

- Testing
- Requirements engineering
- Conditional statements (if .. else .., switch etc.)
- Class responsibilities

```
class Policy {
    public Policy(Date birth);
    public void makeClaim();
    public void annualRenewal();
    public Money getPremiumRate();
    public boolean isCanceled();
    public boolean isActive();
}
```

Decision table vs. Truth table

- Often taken to mean the same thing, they are not interchangeable.
- A truth table is a special case of decision table – all cells in a decision table must be resolvable to true or false

Deriving the logic function

- Boolean expressions
- Karnaugh-Veitch matrix
- Other

Decision table validation

Decision table to be used must be:

- Testable
- Complete
- Consistent
- Error free

Decision table validation – cont.

- ✓ Content checklist
- ✓ Logic function checking
  - Manually transcribing truth tables is easy to make errors (e.g. spreadsheet)
  - Spreadsheet
- ✓ Reasonability view
  - Scrutinized for omissions and inconsistencies

Fault Model

Why are these properties tested with this technique?

Answer can be based on common sense, experience, assumption, analysis or experiment.

Fault Model shows associations and components of the system being tested that have highest probability for faults to occur.

Bug hazard is a circumstance that increase the chance of a bug.

Fault Model

Generally there are two kinds of fault models:

- conformance – directed testing
  - to prove conformance with requirements or specificati
  - Tests are representative enough for essential features of the system being tested
  - Every fault makes the system not conformant (Nonspecific fault model)
- fault-directed testing
  - To prove implementation faults
  - It is based on observation that conformance can be proved for implementation containing bugs.
  - Specific fault model

Fault Model
Combinational Models

- Incorrect or missing
  - Assignment to a decision variable
  - Operator or variable in a predicate
  - Structure in a predicate
  - Default case
  - Action(s)
  - Class or method in composition
- Extra action(s)
- Structural error in decision table implementation
- General errors (e.g. ambiguous reqs.)

Test generation strategies

Strategies:

- All explicit variants
- All variants, All true, All false, All primes
- Each condition/All conditions
- Binary Decision Diagram Determinants
- Variable negation
- Nonbinary Variable Domain Analysis
- Additional Heuristics

Test generation strategies

- All explicit variants
  - Test each explicit variant at least once
  - All true strategy for binary decision variable
  - Appropriate to non-binary variable
  - If implicit variant results from type-safe exclusion it can generate acceptable coverage
  - Very inefficient when can't happen conditions or undefined domain boundaries result in implicit variants

Test generation strategies – cont.

All variants

- Test each variant once
- Feasible for small tables (7, 8 variables)
- Number of tests = 2^n

Test generation strategies – cont.

- All true
  - Test each variant, which produces true
- All false
  - Test each variant, which produces false
- All primes
  - Each prime implicate of the function is tested at least once

Test generation strategies – cont.

Each condition/All conditions

- Based on heuristic to reduce # of tests
- Each variable is made true once with all other variables false
- All variables true (and heuristic)
- All variables false (or heuristic)
- Assumes independence of condition evaluation and absence of faults that would mask an error
- Doesn’ t test don’t know conditions
- Number of tests = n + 1

Test generation strategies – cont.

S = P + Q + R   (or heuristic)

P	Q	R	S
False	False	True	True
False	True	False	True
True	False	False	True
False	False	False	False

Test generation strategies – cont.

S = PQR (and heuristic)

P	Q	R	S
False	False	True	False
False	True	False	False
True	False	False	False
True	True	True	True

Test generation strategies Each condition/All conditions

Z = ABC + AD?

Test cases for ABC

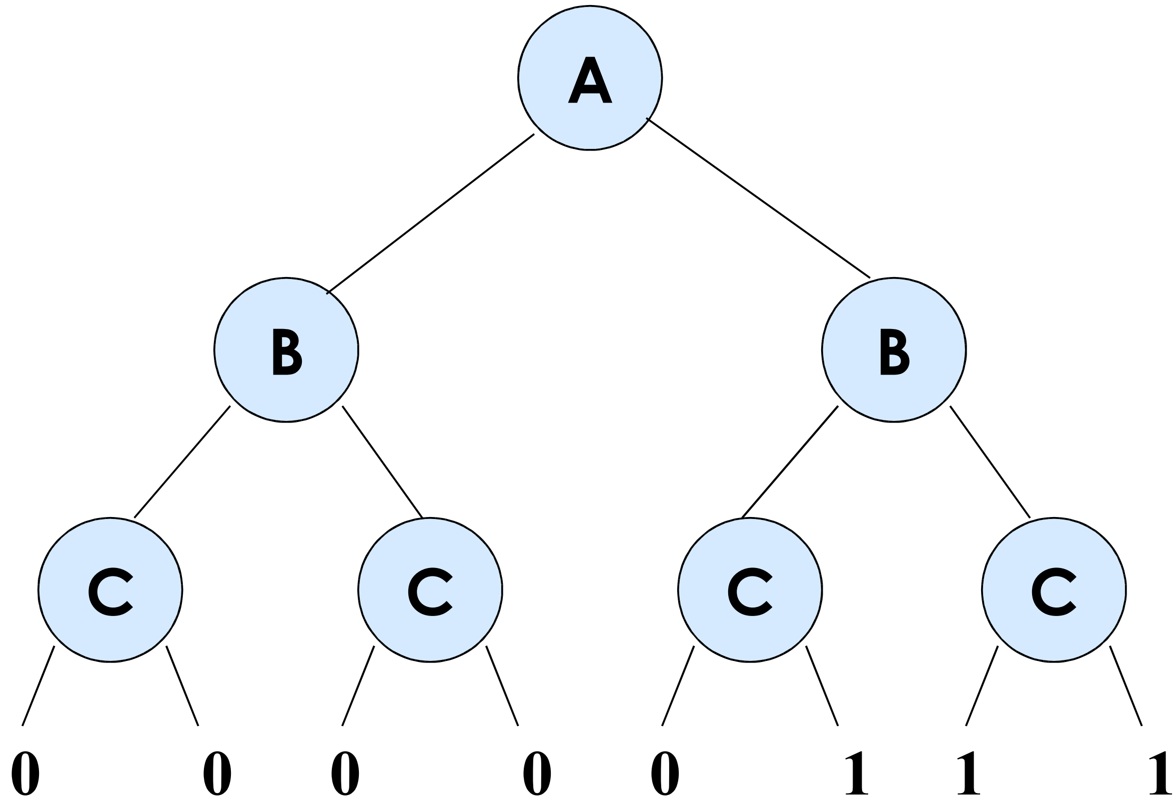
Test cases for AD

Binary Decision Diagram Determinants

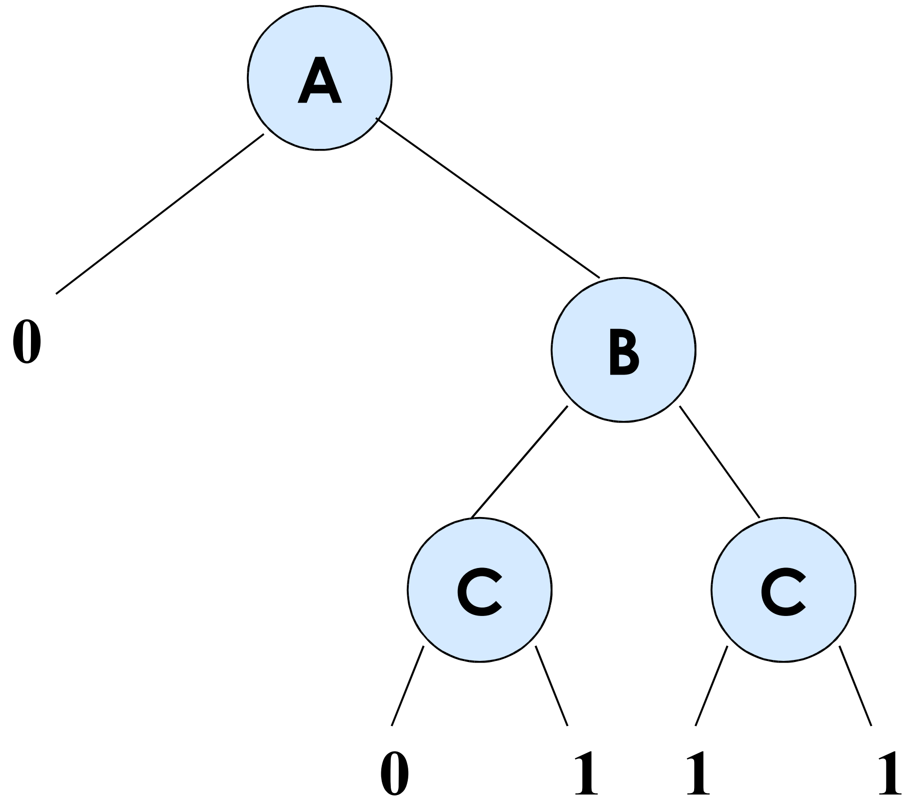
Z = A(B + C)

2. Create BDD diagram upon truth table
   a) Create decision tree upon truth table
      - Nodes represent Boolean variables
      - Left branch is always false
      - Right branch is always true
      - Each leaf represents resultant value for conditions on the path from root to leaf
   b) Bring the decision tree to BDD diagram
      - from left to right replace leafs with equivalent constants or variables and prune branches (reduce the tree)

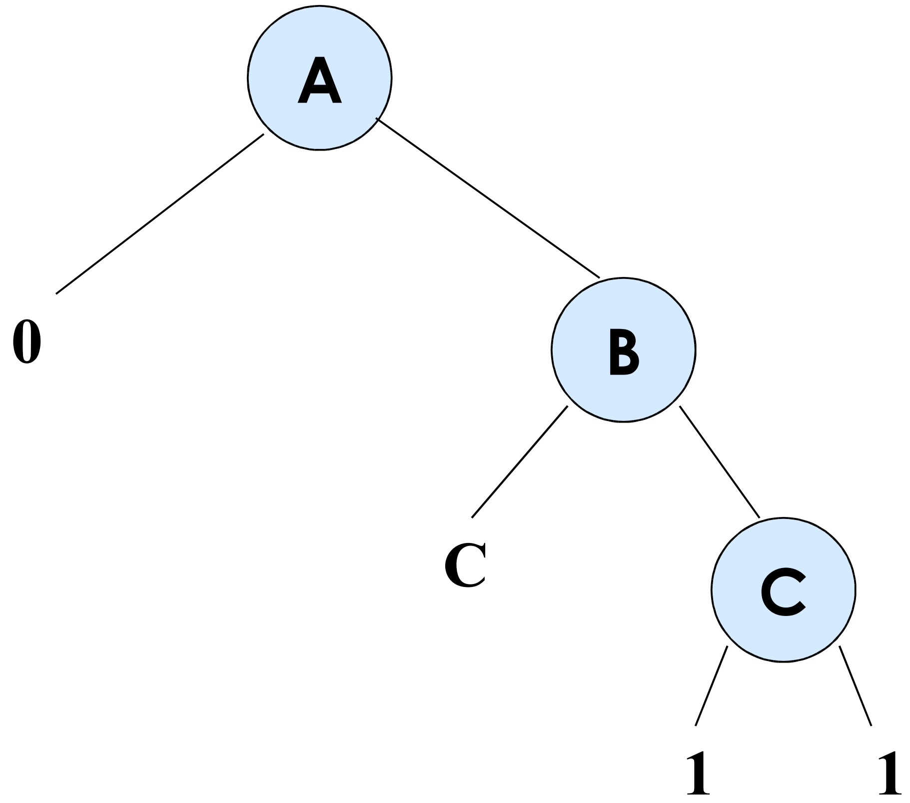
Binary Decision Diagram Determinants – cont.



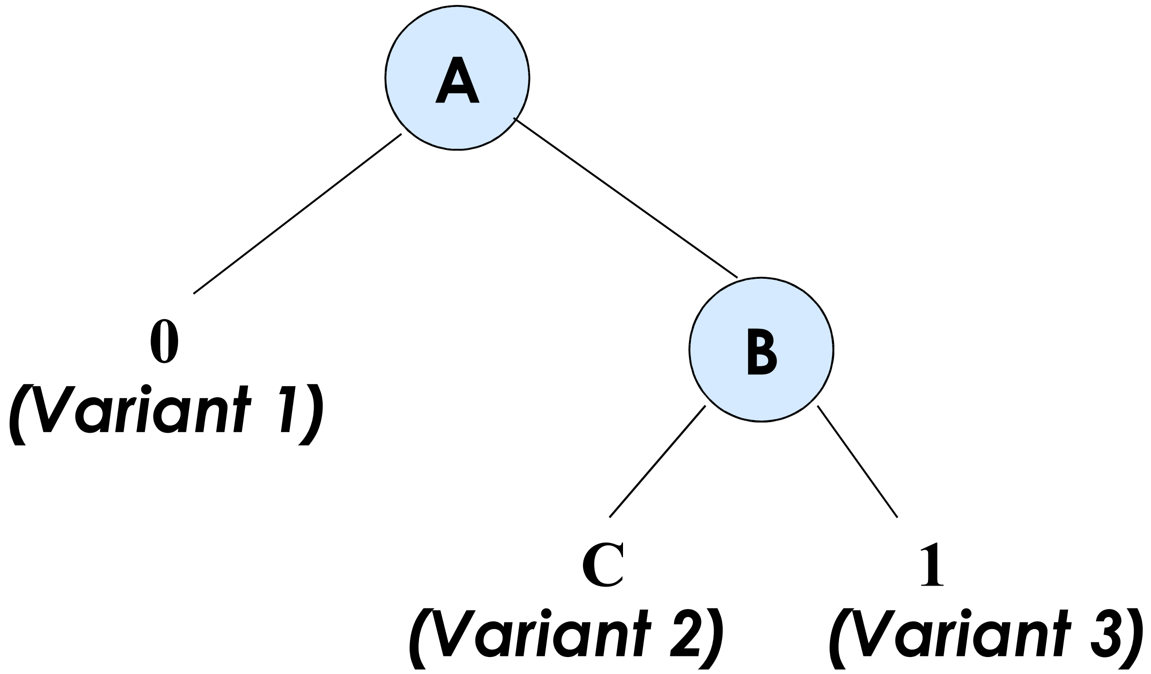
Binary Decision Diagram Determinants – cont.



Binary Decision Diagram Determinants – cont.



Binary Decision Diagram Determinants – cont.



Binary Decision Diagram Determinants – cont.

1. Map BDD diagram into BDD determinants table.

BDD determinant

path from root to leaf in BDD diagram

BDD Variant	A	B	C	Z
1	0	x	x	0
2	1	0	C	C
3	1	1	x	1

Binary Decision Diagram Determinants – cont.

1. Generate BDD test suite

BDD Variant	A	B	C	Z
1	0	x	x	0
2	1	0	0	0
2	1	0	1	1
3	1	1	x	1

What about don't care variables?

Variable negation

- BDD determinant strategy doesn’t check don’t care variables
- Variable negation detects faults in don’t care variables implementation
- Wymaga postaci boolowskiej sumy iloczynów
- Very effective – 97% of accuracy
- Very small – 6% of all possible combinations

Variable negation – cont.

- Variable negation generates sets of test candidates:
  1. One test case for each product term *(pol. składnik iloczynowy)*. Only one product term in the test case is true, other are false.
  2. One test case for each element, which is created by negation of every literal in each product term in a way that Z = 0 (false)

Variable negation – cont.

$Z = A(B + C) = AB + AC$

2. AB = true; AC = false
   { A = 1; B = 1; C = 0; }
   AB = false; AC = true
   { A = 1; B = 0; C = 1; }

Variable negation – cont.

Z = A(B + C) = AB + AC

2. AB

   ~AB { A = 0; B = 1; C = 1; }

   A~B { A = 1; B = 0; C = 0; }

   AC

   ~AC { A = 0; B = 0; C = 1; }

   A~C { A = 1; B = 0; C = 0; }

Variable negation – cont.

- Test suite must contain at least one test case from each set of candidates
- Candidates can be chosen at random or by intuition of tester

Nonbinary Variable Domain Analysis

- Equivalence classes
- Boundary values

Equivalence class

- A set of inputs that a tester believes will be treated similarly by reasonable algorithms
- Input is divided into classes
- For each class the result of the test for any value within this class is representative for every value from this class
- Correct equivalence classes
- Incorrect equivalence classes

Defining test cases

- From each equivalence class choose one value
- Correct test case covers possibly many uncovered equivalence classes
- Incorrect test case covers only one incorrect equivalence class

Boundary Conditions

- Test cases that use values on boundaries of equivalence classes are more effective
- Instead of choosing any value within equivalence class choose values on class's boundaries
- Overflows (e.g. Integer Overflow)
- Number of tests = 2b x 2 x n = 4n
  b is a number of constraints

An Example

- Claims = -1, Insured age 0 - 15
- Claims = 0, Insured age 16 – 25
- Claims = 0, Insured age 26 – 85
- Claims = 1, Insured age 16 – 25
- Claims = 1, Insured age 26 – 85
- Claims = 2 - 4, Insured age 16 – 25
- Claims = 2 - 4, Insured age 26 – 85
- Claims = 5 - 10, Insured age 16 – 85
- Claims = 11 and more, Insured age 86 and more

Additional heuristics

- Change the order in input data
  Correct implementation of decision table should be „order independent”
- Change the order in which tests are run
- Add tests with implicit variants

Choosing a combinational test strategy

- Small function <0; 6> variables
  - All explicit variants when nonbinary
  - All variants ☺
  - Each condition/All conditions
- Medium function < 7;11> variables
  - All variants ☺
  - Each condition/All conditions
  - BDD diagrams
- Big function <12; infinity>
  - Each condition/All conditions
  - Variable negation

Literature

- Robert V. Binder: Testowanie systemów obiektowych. Modele wzorce i narzędzia, WNT 2003

Quality Assessment

Thank You for your attention ☺

- What is your general impression (1-6)
- Was it too slow or too fast?
- What important did you learn during the lecture?
- What to improve and how?

ment

Input Vector No.	Normal Pressure	Call For Heat	Damper Shut	Manual Mode	Ignition Enable
	A	B	C	D	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