SOFTENG 254:
Quality Assurance

Lecture 6a: Logic Coverage

Paramvir Singh
School of Computer Science

Potential Assessment Question

Suppose you have a unit to test for which there are three equivalence partitions. Which of the suggestions below would be most appropriate for the number test cases you would propose based on this information?

(a) 3 (one for each partition)

(b) 5 (one for each partition, 1 between first and second, 1 between second and third)

(c) 9 (1 per partition, 1 each either end of each partition)

(d) Don't know but a number much bigger than 9.

Justify your answer.

Agenda

- Admin
  - Assignment 1
    - Due date approaching (this Saturday)
- Logic coverage
- Decision tables (part 1)

Previously in SOFTENG 254

- Suppose statement is:
  `if (a > 0 && b > 0) {...}`
  but should have been:
  `if (a > 0 || b > 0) {...}`
- This test suite gives full branch coverage, but does not detect the fault

a	a > 0	b	b > 0	Expected	Actual
10	true	30	true	true	true
-1	false	-10	false	false	false

- Need to test other possible combinations of sub-expressions

a	a > 0	b	b > 0	Expected	Actual
10	true	-10	false	true	false
-1	false	30	true	true	false

- Also known as condition coverage

Predicates

- logic (or boolean) expressions appear in many aspects of software development
  - test conditions (e.g. loops, if)
  - descriptions of what code is supposed to do (e.g., pre and post conditions of methods)
  - database queries
  - specifications (formal requirements)
  - decision tables
- a logic expression, or predicate, is an expression that evaluates to a boolean value
- predicates can contain
  - boolean variables
  - relational expressions (using $<, <=, >, >=, ==, !=$ etc)
  - methods that return booleans
  - expressions involving logical operators (!, &&, ||, etc)
- A clause is a logic expression with no logical operators
- $\Rightarrow$ a predicate is a set of 1 or more clauses

Predicates and Testing

- Model the IUT we want to test with a predicate (if possible)
- Test requirements are based on clauses in the predicate
- Predicate Coverage: Given a predicate p, there are two test requirements — p evaluates to true, and p evaluates to false
  - also known as branch coverage
- Clause Coverage: For every clause c in a predicate p, there are two test requirements —- c evaluates to true, and c evaluates to false
- Combinatorial Coverage: For a predicate p, there is a test requirement for every combination of truth values for every clause c in p.
  - also known as condition coverage

Example

- $p = ((a < b)||d)\&\&(m >= n * o)$

  (from *Introduction to Software Testing*, Ammann and Offutt)

- test cases

a	b	$a < b$	d	m	n	o	$m >= n * o$	p
5	10	true	true	1	1	1	true	true
10	5	false	false	1	2	2	false	false

  - these tests provide both full predicate and clause coverage but **not** combinatorial coverage

- Why combinatorial coverage?
  - predicate coverage does not exercise all clauses
  - clause coverage does not always ensure predicate coverage
- but, combinatorial coverage grows exponentially

⇒ subset of combinatorial coverage that tests each clause independently from the other clauses

Active Clauses

- Consider $p = a > 0 \;\&\&\; b < 10$
  - if $a = -1$ then $a > 0$ is false, and so p is false no matter what b is
- For a given predicate p consisting of clauses $c_1, c_2, \ldots c_k$, choose one clause (any one) as the major clause, then all the others are minor clauses
- For p with clauses $c_1, c_2, \ldots c_k$, if c_i is the major clause, then c_i determines p if the values of the minor clauses are such that changing the value of c_i changes the value of p.
- A major clause that determines a predicate is called an active clause
- Example:
  - $p = a > 0 \;\&\&\; b < 10$, major clause—$a > 0$
  - $b = 5 \Rightarrow (b < 10) =$ true
  - $a = 5 \Rightarrow (a > 0) =$ true $\Rightarrow$ $p =$ true
  - $a = -1 \Rightarrow (a > 0) =$ false $\Rightarrow$ $p =$ false
  - $\Rightarrow (a > 0)$ determines p (witness $b = 5$)

Active Clauses and Testing

- Goal: find tests for each clause when that clause determines the value of the predicate
- Active Clause Coverage (ACC): For each predicate p consisting of clauses $c_1, c_2, \ldots c_k$ for each major clause c_i, $0 \leq i \leq k$, choose minor clauses c_j, $i \neq j$ so that c_i determines p: there are two test requirements — c_i evaluates to true and c_i evaluates to false
- Other variations to deal with issues that arise with ACC, in particular should the minor clauses have the same values when changing the major clause?
- Also Inactive Clause Coverage — choose values of minor clauses so that major clause **does not** determine the predicate

Example

- $p = (a \;\&\&\; e) || (b \;\&\&\; f)$

	Major	a	e	b	f	p
1	a	true	true	true	false	true
2	a	false	true	true	false	false
3	e	true	true	false	false	true
4	e	true	false	false	false	false
5	b	false	true	true	true	true
6	b	false	true	false	true	false
7	f	false	true	true	true	true
8	f	false	true	true	false	false

- 4 clauses $\Rightarrow$ 16 combinations
- ACC maximum 8, and usually fewer due to duplication (e.g. 2 and 8)

Infeasible Test Requirements

- As for other forms of coverage, logic coverage criteria may lead to infeasible test requirements, especially predicates found in code
- E.g. short-circuit boolean evaluation

```
while (i < a.length && a[i] > 0) { i++; }
```

If `i < a.length` then `a[i]>0` will never be evaluated

Decision Tables

- In some cases can use *decision tables* to model the IUT (or rather, the requirements)
- Decision tables use predicates in the models that dictate actions that need to be taken
- Once a model is created, test cases can be *generated* from the model
- Needs for decision tables:
  - one of several distinct responses is to be selected according to distinct cases of input variables
  - cases can be modelled by boolean expressions on the input variables
  - actions do not depend on order of inputs
  - actions do not depend on state, i.e, depends only on inputs

Process

1. Model the implementation with a decision table
   (a) identify decision variables and conditions
   (b) identify resultant actions
   (c) identify actions for each combination of conditions
   (d) verify completeness and consistency
2. derive logic function (optional)
3. choose test suite generation strategy
4. generate test suite

Decision Tables

- two parts: condition section, action section
- *condition section*: lists input variables, boolean expression consisting of input variables
- *action section*: lists actions when conditions are true
- each combination of conditions and actions is a *variant*
- each combination must be unique
- the same actions may apply to multiple combinations

Example: Car Insurance Renewal

- Determine what happens to car insurance on annual renewal of policy
- What happens depends on person's age, number of claims
- What happens is how much the premium increases or whether or not the policy is cancelled
- (Adapted from Binder)

Example: Car Insurance Renewal

- If the person is 25 or younger and they have zero claims they will receive a $50 increase in their insurance premium
- If the person is 26 or older and they have zero claims they will receive a $25 increase in their insurance premium
- If the person has one claim and they are:
  - 25 or younger ⇒ $100 increase, warning letter
  - 26 or older ⇒ $50 increase
- If the person has 2-4 claims, they get a warning letter, and if they are:
  - 25 or younger ⇒ $400 increase
  - 26 or older ⇒ $200 increase
- If the person has more than 5 claims, their policy is cancelled (and no increase to premium)

Example: Designing a decision table

1. Identify decision variables and conditions
   - decision variables: age, number of claims
   - conditions: age ≤ 25 or ≥ 26, claims 0, 1, 2-4, ≥ 5

     operator: and
2. Identify actions
   - premium increase amount, warning, cancel policy

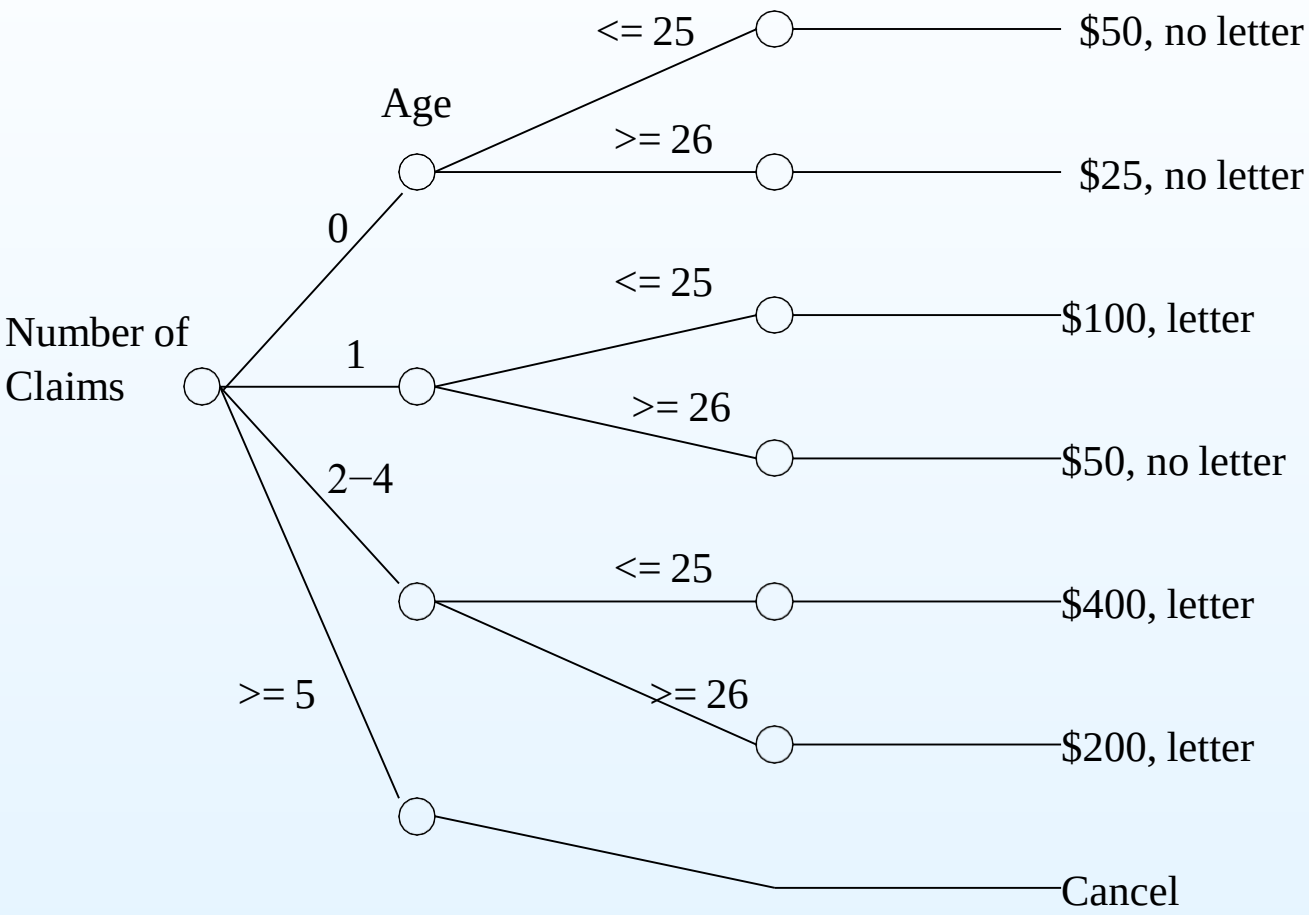
Example: Decision Table

Condition Section			Action Section		
Variant	Number of Claims	Age	Increase	Warning	Cancel
1	0	≤ 25	50	No	No
2	0	≥ 26	25	No	No
3	1	≤ 25	100	Yes	No
4	1	≥ 26	50	No	No
5	2–4	≤ 25	400	Yes	No
6	2–4	≥ 26	200	Yes	No
7	≥ 5	Any	0	No	Yes

Decision Table Representations

- Number of different ways decision tables could be represented:
  - row or column
  - tree
  - truth table
- some are easier for humans to deal with, some are easier for machines (or for converting to machine use)

Decision Tree



Truth Table

	Decision Variable	Condition	Variant 1 2 3 4 5 6 7
Condition Section	Number of Claims	0 1 2-4 ≥ 5	T T F F F F F F F T T F F F F F F F T T F F F F F F F T
			T F T F T F DC F T F T F T DC
	Age	≤ 25 ≥ 26	F F F F F F T F T F F F F F T F F T F F F F F T F F F F F F
Action Section	Increase = 0 Increase = 25 Increase = 50 Increase = 100 Increase = 200 Increase = 400 Send warning Cancel		F F F T F F F F F T F F F F T F T T F F F F F F F T

How Many Variants?

- Insurance example has 6 conditions: 4 different values for number of claims, 2 different values for age

$\Rightarrow$ $2^6 = 64$ different variants are possible, but only 7 are shown in the decision table/tree

Implicit Variants

- Explicit Variants — listed explicitly in the model
- Implicit Variants — not listed for various reasons:
  - don’t need to be as they are implied by explicit variants
    - don’t cares
    - type-safe exclusions
  - missing due to incorrect modelling
    - can’t happen
    - don’t know

Key Points

- For predicates, combinatorial coverage gives the best test suite, but the number of test grows exponentially ⇒ need a way to find a subset that still gives a good quality test suite
- An active clause maximises the variation of the value of the predicate while restricting the number of variation of clauses needed
- Decision tables can be used to model functionality that can be expressed as predicates