Software Testing

Sibylle Schupp[1]

[1]Institute for Software Systems/Institut für Softwaresysteme
Hamburg University of Technology (TUHH)

Spring 2022

Lecture 8

Outline

Recall

- Clause (predicate without logic operators)
- Semantic coverage
- Logic coverage criteria (semantic):
  - PC, CC for predicate and clause coverage
  - CoC for complete clause coverage
  - ACC (GACC, CACC, RACC) for active clause coverage
    - A major clause c of a predicate p determines p if the minor clauses have values so that changing the value of c changes the value of p.
    - Active clause coverage (ACC) is formulated in terms of determination but ambiguous.
    - Three unambiguous interpretations: general, correlated, restricted
  - ICC for inactive clause coverage

Outline

1. Logic coverage II
   - Semantic logic coverage of programs
   - Logic coverage for specifications
   - Semantic logic coverage of FSMs

Logic coverage for source code

- Predicates are derived from test expressions (decisions)
  - In programs, most predicates have not more than 3 or 4 clauses.
  - If a predicate has one clause only, CoC, ACC, ICC, CC, PC coincide.
- Applying logic criteria to program source code is not trivial:
  - Reachability: how to get to the test expression?
  - Controllability: which input values assign the right values to the variables in the predicate?
  - Variables of a predicate that are not input variables are called internal variables.

Example (semantic coverage of source code)

```
// Introduction to Software Testing
// Authors: Paul Ammann & Jeff Offutt
public class Thermostat
{
   private int curTemp;             // current temperature reading
   private int thresholdDiff;       // temp difference until we turn
        heater on
   private int timeSinceLastRun;    // time since heater stopped
   private int minLag;              // how long I need to wait
   private boolean override;        // has user overridden the program
   private int overTemp;            // overriding temperature
   private int runTime;             // output of turnHeaterOn — how
        long to run
   private boolean heaterOn;        // output of turnHeaterOn —
        whether to run
   private Period period;           // morning, day, evening, or night
   private DayType day;             // week day or weekend day

   // Decide whether to turn the heater on, and for how long.
   public boolean turnHeaterOn (ProgrammedSettings pSet) {..}
```

Example (cont'd)

```
// Introduction to Software Testing
// Authors: Paul Ammann & Jeff Offutt
// Decide whether to turn the heater on, and for how long.
   public boolean turnHeaterOn (ProgrammedSettings pSet) {
      int dTemp = pSet.getSetting (period, day);
      if (((curTemp < dTemp - thresholdDiff)
               || (override && curTemp < overTemp - thresholdDiff))
          && (timeSinceLastRun > minLag))
      {  // Turn on the heater
         // How long? Assume 1 minute per degree (Fahrenheit)
         int timeNeeded = curTemp - dTemp;
         if (override)
            timeNeeded = curTemp - overTemp;
         setRunTime (timeNeeded);
         setHeaterOn (true);
         return (true);
      }
      else {
         setHeaterOn (false);
         return (false);
      }
   } // End turnHeaterOn
```

Example (predicates)

The example code contains two predicates:

```
// predicate p1
if (((curTemp < dTemp − thresholdDiff)
        || (override && curTemp < overTemp − thresholdDiff))
    && (timeSinceLastRun > minLag))

// predicate p2
if (override)
```

We introduce abbrevations for the clauses:

a	curTemp < dTemp - thresholdDiff
b	override
c	curTemp < overTemp - thresholdDiff
d	timeSinceLastRun > minLag

Thus,

$$p_1 \equiv (a \,||\, (b \,\&\&\, c)) \,\&\&\, d) \quad \text{and} \quad p_2 \equiv b$$

Reachability

Reachability: when are p_1, p_2 reached?

	Condition
p_1	true ("p_1 is always reached")
p_2	$(a \,||\, (b \,\&\&\, c)) \,\&\&\, d)$ ("p_2 depends on p_1")

Determine reachability conditions before defining TRs (for a certain coverage criterion).

Controllability

Assume predicate coverage.

- Consider the true cases of p_1, p_2. Set as TR: a = b = c = d = true (other truth assignments exist as well)
- Find test values (other test values exist as well):

Clause		Test Values
a	curTemp < dTemp - thresholdDiff	63, 69, 5
b	override	true
c	curTemp < overTemp - thresholdDiff	63, 70, 5
d	timeSinceLastRun > minLag	12, 10

- Problem: controllability.
  Predicate p_1 depends on the local variable *dTemp*.

Predicate coverage (true case)

```
@Test
  public void turnHeaterOn_True (){
  // a true
   thermo.setCurrentTemp (63);
   thermo.setThresholdDiff (5);
   // b true .. c true ..  d true
   thermo.setMinLag (10);
   thermo.setTimeSinceLastRun(12);

   assertTrue(thermo.turnHeaterOn(settings));
}
@BeforeEach
public void setUp() {
    thermo   = new Thermostat();
    settings = new ProgrammedSettings();

    settings.setSetting(Period.MORNING, DayType.WEEKDAY, 69);
    thermo.setPeriod(Period.MORNING); // param. for getSetting
    thermo.setDay(DayType.WEEKDAY);
}
```

Selected laws of the Boolean algebra

Mathematical notation

Active clause coverage depends on determining values. The computation of determining values uses laws of the Boolean algebra.

- de Morgan laws

$$\begin{aligned} \neg(a \vee b) &\equiv \neg a \wedge \neg b \\ \neg(a \wedge b) &\equiv \neg a \vee \neg b \end{aligned}$$

- $\oplus$ laws

$$\begin{aligned} \mathit{true} \oplus a &\equiv \neg a \\ \mathit{false} \oplus a &\equiv a \\ a \oplus b &\equiv (a \wedge \neg b) \vee (\neg a \wedge b) \end{aligned}$$

- $\vee$ laws and $\wedge$ laws

$$\begin{aligned} a \vee a &\equiv a \\ a \wedge a &\equiv a \\ a \vee \neg a &\equiv \mathit{true} \\ a \wedge \neg a &\equiv \mathit{false} \end{aligned}$$

Finding values for minor clauses

- Let c be a clause in p. We assume that each clause occurs only once.
- Denote by $p_{c=true}$ the predicate in which the occurence of c in p has been replaced by true. Similarly, $p_{c=false}$ denotes the predicate that one obtains if one replaces c in p by false.
- Set

$$p_c = p_{c=true} \oplus p_{c=false}$$

For all values for which p_c=true, c determines p.

Example (correlated active clause coverage, CACC)

Consider again

$$p \equiv (a \,||\, (b \,\&\&\, c)) \,\&\&\, d)$$

Compute determining values for major clause a:

$$\begin{array}{lll} p_a & \equiv & (true \,||\, (b \,\&\&\, c)) \,\&\&\, d) \oplus (false \,||\, (b \,\&\&\, c)) \,\&\&\, d) \\ & = & (true \&\& \, d) \oplus ((b \,\&\&\, c) \;\&\&\; d) \\ & = & d \oplus ((b \,\&\&\, c) \;\&\&\; d) \\ & (= & true \oplus ((b \,\&\&\, c) \;\&\&\; true)) \\ & = & !(b \,\&\&\, c) \\ & = & (!b \,||\, !c) \end{array}$$

- The clause a determines the predicate p iff d=true and b or c=false.
- Similarly, $p_b =!a \,\&\&\, c \,\&\&\, d$, $p_c =!a \,\&\&\, b \,\&\&\, d$, $p_d = a \,||\, (b \,\&\&\, c)$

Example (TRs for correlated active clause coverage)

$$(a \,||\, (b \,\&\&\, c)) \,\&\&\, d)$$

	a	b	c	d	p
a, p_a	true	true	false	true	true
	false	true	false	true	false
b, p_b	false	true	true	true	true
	false	false	true	true	false
c, p_c	false	true	true	true	true
	false	true	false	true	false
d, p_d	true	true	true	true	true
	true	true	true	false	false

- For CACC, six tests suffice (duplicates for p_b, p_c).
- In the TRs, we capitalize the value of the major clause

Ttft FtFt fTTt fFtt tttT tttF

Example (cont'd): test cases

```
@Test
public void turnHeaterOn_PaF (){
    // Ftft
    // a false
    thermo.setCurrentTemp (66);
    thermo.setThresholdDiff (5);

    // b true
    thermo.setOverride(true);

    // c false
    thermo.setOverTemp(65);

    // d true
    thermo.setMinLag (10);
    thermo.setTimeSinceLastRun(12);

    assertFalse(thermo.turnHeaterOn(settings));
}
```

- For each of the six test cases: set variables and oracle appropriately.

Program transformation: motivation (?)

Consider the following example:

```
if ((a && b) || c) {
       S1;
}
else   {
       S2;
}
```

- ACC criteria are comparatively expensive. In the example, CACC requires 4 tests:

	a	b	c	$(a \wedge b) \vee c$	CACC
1	true	true	true	true	
2	true	true	false	true	x
3	true	false	true	true	x
4	true	false	false	false	x
5	false	true	true	false	
6	false	true	false	false	x
7	false	false	true	false	
8	false	false	false	false	

Example (program transformation)

- Could one transform the code into code with predicates that have fewer clauses?

```
if (a) {
        if (b)
              S1;
        else {
              if (c)
                   S1;
              else
                   S2;
        }
}
else {
        if (c)
              S1;
        else
              S2;
}
```

Example (program transformation), cont'd

- Predicate coverage requires 5 tests:

	a	*b*	*c*	PC (other choices exist)
1	true	true	true	x
2	true	true	false	
3	true	false	true	x
4	true	false	false	x
5	false	true	true	x
6	false	true	false	
7	false	false	true	
8	false	false	false	x

- Problems:
  - More tests (need to reach each predicate)
  - CACC of the original problem not necessarily satisfied
  - Readability and maintainability hampered

Predicates with side effects

Another problem:

- When a value changes while the predicate is evaluated, the program has a side effect.
- Conditions for a side effect:
  1. A clause occurs twice; and
  2. a clause in between changes one of its variables.
- Ex.

```
if (a && (b || a))
```

  where *b* could be the return value of the function `changeVar(a)` (and thus requires its invocation).
- Problem: cannot write a test that has two different values for the same predicate.

Summary (logic coverage for source code)

- In source code, predicates occur frequently (while, if, for)
- The hard part in testing: reachability
  - Internal variables
- Avoid transformations that hide the structure

Outline

1. Logic coverage II
   - Semantic logic coverage of programs
   - Logic coverage for specifications
   - Semantic logic coverage of FSMs

Logic coverage for specifications

- Specifications can be formal and informal
  - Formal: mathematical logic ($\rightsquigarrow$ software verification)
  - Informal: natural languages
- Formal specifications can be used (almost) directly
  - Informal specifications need to be formalized
  - Common example: preconditions

Preconditions

- Preconditions are often expressed as comments in method headers.

```
void saveAddress(string name,  string state, int zip,
                 string street, string city)
// name must not be empty
// state must be valid
// zip must be 5 numeric digits
// street must not be empty
// city must not be empty
```

- As a Boolean expression: name $!=""$ $\wedge$ state in stateList $\wedge$ zip $\geq 0000 \wedge$ zip $\leq 99999 \wedge$ street $!= ""$ $\wedge$ city $!= ""$

Conjunctive Normal Form

Definition

A predicate is in conjunctive normal form (CNF) if it consists of clauses or disjunctions, connected by the $\wedge$ operator.

- Ex.: $a \wedge b \wedge c \wedge d$ (where a, b, c, d can be disjunctions)
- In CNF, a major clause is active (determines) when all other clauses are true.
- TR for ACC coverage: all true and the diagonal of "false" values

	a	b	c	..	..
1	true	true	true	true	
2	false	true	true	true	
3	true	false	true	true	
4	true	true	false	true	
5	true	true	true	false	
6	...				

Disjunctive Normal Form

Definition

A predicate is in disjunctive normal form (DNF) if it consists of clauses or conjunctions, connected by the $\vee$ operator.

- $a \vee b \vee c \vee d$ where a, b, c, d could be conjunctions.
- In DNF, a major clause is active (determines) when all other clauses are false.
- TR for ACC coverage: "all false" and the diagonal of "true" values

	a	b	c	..	..
1	false	false	false	false	
2	true	false	false	false	
3	false	true	false	false	
4	false	false	true	false	
5	false	false	false	true	
6	...				

Summary (logic coverage of specifications)

- Logic specifications are quite frequent.
  - Preconditions, asserts, OCL, design-by-contract
  - Formal languages: SAT, LTL
- Available at different times in the software life cycle
  - Methods and classes (unit testing)
  - Dependencies between classes and components (module testing)
  - System (system testing)
- CNF or DNF simplify finding TRs.

Outline

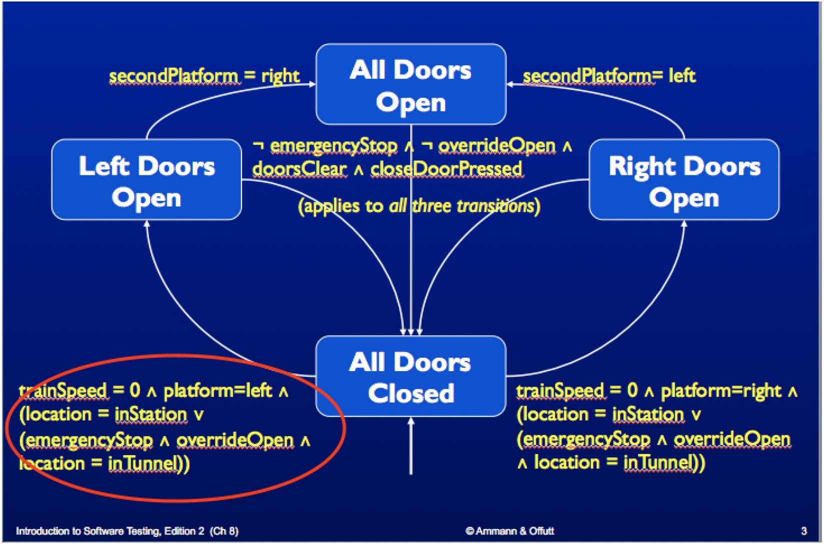
1 Logic coverage II

- Semantic logic coverage of programs
- Logic coverage for specifications
- Semantic logic coverage of FSMs

Logic coverage for finite state machines

- Recall: FSMs can be considered graphs
  - Nodes represent states
  - Edges represent transitions among states
- Transitions are often guarded by a logical expression
- Testing: cover "all" logical expressions

Example (FSM)


All Doors Open
secondPlatform = right
secondPlatform= left
Left Doors Open
Right Doors Open
¬ emergencyStop ∧ ¬ overrideOpen ∧
doorsClear ∧ closeDoorPressed
(applies to *all* three transitions)
All Doors Closed
trainSpeed = 0 ∧ platform=left ∧
(location = inStation ∨
(emergencyStop ∧ overrideOpen ∧
location = inTunnel))
trainSpeed = 0 ∧ platform=right ∧
(location = inStation ∨
(emergencyStop ∧ overrideOpen
∧ location = inTunnel))
Introduction to Software Testing, Edition 2 (Ch 8)
© Ammann & Offutt
3

Example (determination)

The predicate p

$$trainSpeed = 0 \wedge platform{=}left \wedge (location = inStation \vee (emergencyStop \wedge overrideOpen \wedge location = inTunnel))$$

- For each of the 6 clauses, find the truth assignments that let the clause determine the value of the predicate.
- Compute $p_{trainSpeed=0}$, $p_{platform=left}$, ...

Example (determination), cont'd

The predicate p

$$\text{trainSpeed} = 0 \land \text{platform=left} \land (\text{location} = \text{inStation} \lor (\text{emergencyStop} \land \text{overrideOpen} \land \text{location} = \text{inTunnel}))$$

- $p_{trainspeed=0} =$
  $$\text{platform=left} \land (\text{location} = \text{inStation} \lor (\text{emergencyStop} \land \text{overrideOpen} \land \text{location} = \text{inTunnel}))$$
- $p_{platform=left} =$
  $$\text{trainSpeed} = 0 \land (\text{location} = \text{inStation} \lor (\text{emergencyStop} \land \text{overrideOpen} \land \text{location} = \text{inTunnel}))$$
- $p_{location=inStation} =$
  $$\text{trainSpeed} = 0 \land \text{platform=left} \land (\neg\, \text{emergencyStop} \lor \neg\, \text{overrideOpen} \lor \neg\, (\text{location} = \text{inTunnel})))$$

Example (CACC)

	trainSpeed = 0	platform = left	location = in-Station	emergen-cyStop	Over-ride-Open	locatio = in-Tunne
trainSpeed=0						
trainSpeed $\neq$ 0						
platform=left						
platform$\neq$left						
inStation						
$\neg$inStation						
emergencyStop						
$\neg$emergencyStop						
overrideOpen						
$\neg$overrideOpen						
inTunnel						
$\neg$inTunnel						

Example (CACC), cont'd

	trainSpeed = 0	platform = left	location = in-Station	emergen-cyStop	Over-ride-Open	locatio = in-Tunnel
trainSpeed=0	T					
trainSpeed ≠ 0	F					
platform=left		T				
platform≠left		F				
inStation			T			
¬inStation			F			
emergencyStop				T		
¬emergencyStop				F		
overrideOpen					T	
¬overrideOpen					F	
inTunnel						T
¬inTunnel						F

Example (CACC), cont'd

	trainSpeed = 0	platform = left	location = inStation	emergencyStop	OverrideOpen	location = inTunnel
trainSpeed=0	T	t	t	t	t	t
trainSpeed ≠ 0	F	t	t	t	t	t
platform=left	t	T	t	t	t	t
platform≠left	t	F	t	t	t	t
inStation	t	t	T	f	f	f
¬inStation	t	t	F	f	f	f
emergencyStop	t	t	f	T	t	t
¬emergencyStop	t	t	f	F	t	t
overrideOpen	t	t	f	t	T	t
¬overrideOpen	t	t	f	t	F	t
inTunnel	t	t	f	t	t	T
¬inTunnel	t	t	f	t	t	F

Example (CACC): problem

	trainSpeed = 0	platform = left	location = inStation	emergencyStop	OverrideOpen	location = inTunnel
inStation	t	t	T	f	f	f
¬inStation	t	t	F	f	f	f

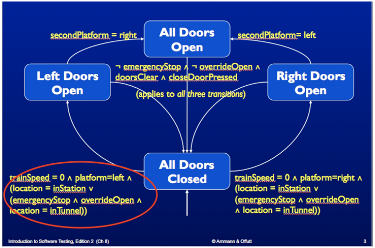
- The model contains two locations for the train: inStation and inTunnel.
- Thus, the two predicates cannot both be false (or true).
    - "Dependent clauses"
- Options
    - Rewrite the predicate to eliminate dependencies (not always possible)
    - Change truth assignment. In the example, change t t F f f f

Example (CACC): revised truth assignments

	trainSpeed = 0	platform = left	location = in-Station	emergen-cyStop	Over-ride-Open	locatio = in-Tunnel
trainSpeed=0	T	t	t	t	t	f
trainSpeed ≠ 0	F	t	t	t	t	f
platform=left	t	T	t	t	t	f
platform≠left	t	F	t	t	t	f
inStation	t	t	T	f	f	f
¬inStation	t	t	F	f	f	t
emergencyStop	t	t	f	T	t	t
¬emergencyStop	t	t	f	F	t	t
overrideOpen	t	t	f	t	T	t
¬overrideOpen	t	t	f	t	F	t
inTunnel	t	t	f	t	t	T
¬inTunnel	t	t	f	t	t	F
infeasible						

Example (accidential transitions)

When the major clause is true, the transition is taken.



Major Clause	Expected Output
trainSpeed=0	Left Doors Open
trainSpeed $\neq$ 0	All Doors Closed
platform=left	Left Doors Open
platform≠left	All Doors Closed
inStation	Left Doors Open
¬inStation	All Doors Closed
emergencyStop	Left Doors Open
¬emergencyStop	All Doors Closed
overrideOpen	Left Doors Open
¬overrideOpen	All Doors Closed
inTunnel	Left Doors Open
¬inTunnel	All Doors Closed

Accidental transitions

- In the general case: when the major clause is true, the transition is taken. When the major clause is false, no transition is taken.
- Now consider

Major Clause	Expected Output
platform=left	Left Doors Open
platform≠left	All Doors Closed ??

- If platform≠left, it holds that platform=right. The expected output (transition) should be “Right Doors Open”.
- The transition is <u>accidental</u> since it is implicit.
- Obviously, accidential transitions must be recognized by hand.

Complicating issues for test automation

- We have seen:
  - Dependent clauses
  - Accidental transitions
  - Reachability
    - The tests must reach the state where the transition starts ("prefix").
- Additional issues:
  - Some tests must reach particular final states.
  - Mapping
    - The predicates in the FSM may not match the predicates in the program.
    - The predicates in the FSM might encapsulate sequences of actions in the program. Ex: trainspeed $= 0$.

Summary (FSM logic testing)

- FSM is one of the most widely used notation.
  - Used at all levels of the software development process
  - Used in many domains, in particular embedded software
- Many languages exist
  - UML state diagrams, Petri Nets, decision tables, Z, . . .
- Safety
  - Often used in guards, often as safety constraints

In-class exercise

remove

`void remove()`

Removes from the underlying collection the last element returned by this iterator (optional operation). This method can be called only once per call to `next()`. T
behavior of an iterator is unspecified if the underlying collection is modified while the iteration is in progress in any way other than by calling this method.

Throws:

`UnsupportedOperationException` - if the `remove` operation is not supported by this iterator

`IllegalStateException` - if the `next` method has not yet been called, or the `remove` method has already been called after the last call to the `next` meth

In-class exercise (cont'd)

References

- AO, Ch. 8.3, 8.4, 8.5