Introduction to Software Testing
Chapter 8.3
Logic Coverage for Source Code

Paul Ammann & Jeff Offutt

http://www.cs.gmu.edu/~offutt/softwaretest/

Logic Expressions from Source

- Predicates are derived from decision statements
  - if, while, for, switch, do-while
- In programs, most predicates have less than four clauses
  - In fact, most have just one clause
- When a predicate only has one clause, CoC, ACC, and CC all collapse to predicate coverage (PC)
  - ACC is only useful with three or more clauses

Finding Values

- *Reachability* : Each test much reach the decision
- *Controllability* : Each test must cause the decision to have specific truth assignment
- *Internal variables :* Predicates variables that are not inputs

```
public int checkVal(int x) {
   y = x*2;
   if (x>0)
      if ((x>10 && x<20) || y==50)
         return 1;
   else
      if ((x<-10 && x>-20) || y<-60)
         return 2;
}
```

y is an internal variable

Reach: x > 0

Control for FFT: x=25

Thermostat (*pg 1 of 2*)

```
 1  // Jeff Offutt & Paul Ammann—September 2014
 2  // Programmable Thermostat
 6  import java.io.*;
10  public class Thermostat
11  {
12     private int curTemp;              // Current temperature reading
13     private int thresholdDiff;        // Temp difference until heater on
14     private int timeSinceLastRun; // Time since heater stopped
15     private int minLag;               // How long I need to wait
16     private boolean Override;         // Has user overridden the program
17     private int overTemp;             // OverridingTemp
18     private int runTime;              // output of turnHeaterOn–how long to run
19     private boolean heaterOn;         // output of turnHeaterOn – whether to run
20     private Period period;            // morning, day, evening, or night
21     private DayType day;              // week day or weekend day
23     // Decide whether to turn the heater on, and for how long.
24     public boolean turnHeaterOn (ProgrammedSettings pSet)
25     {
```

Thermostat (*pg 2 of 2*)

```
26    int dTemp = pSet.getSetting(period, day);
28    if (((curTemp < dTemp - thresholdDiff) ||
29        (Override && curTemp < overTemp - thresholdDiff)) &&
30        (timeSinceLastRun > minLag))
31    {  // Turn on the heater
32       // How long? Assume 1 minute per degree (Fahrenheit)
33       int timeNeeded = Math.abs(dTemp - curTemp);
34       if (Override)
35          timeNeeded = Math.abs(overTemp – curTemp);
36       setRunTime(timeNeeded);
37       setHeaterOn(true);
38       return(true);
39    }
40    else
41    {
42       setHeaterOn(false);
43       return(false);
44    }
45  } // End turnHeaterOn
```

The full class is in the book and on the book website.

Two Thermostat Predicates

28-30 : (((curTemp < dTemp - thresholdDiff) ||
(Override && curTemp < overTemp - thresholdDiff)) &&
timeSinceLastRun > minLag))

34 : (Override)

Simplify

a : curTemp < dTemp - thresholdDiff
b : Override
c : curTemp < overTemp - thresholdDiff
d : timeSinceLastRun > minLag)
28-30 : (a || (b && c)) && d
34 : b

Reachability for Thermostat Predicates

28-30 : True

34 : (a || (b && c)) && d

curTemp < dTemp - thresholdDiff

Need to solve for the internal variable *dTemp*

pSet.getSetting (period, day);

```
setSetting (Period.MORNING, DayType.WEEKDAY, 69);
setPeriod (Period.MORNING);
setDay (DayType.WEEKDAY);
```

Predicate Coverage (*true*)

(8.3.1)

(a || (b && c)) && d

a : true b : true
c : true d : true

a: curTemp < dTemp – thresholdDiff : true
b: Override : true
c: curTemp < overTemp – thresholdDiff : true
d: timeSinceLastRun > (minLag) : true

```
thermo = new Thermostat();  // Needed object
settings = new ProgrammedSettings();  // Needed object
settings.setSetting(Period.MORNING, DayType.WEEKDAY, 69);  // dTemp
thermo.setPeriod(Period.MORNING);  // dTemp
thermo.setDay(DayType.WEEKDAY);  // dTemp
thermo.setCurrentTemp(63);  // clause a
thermo.setThresholdDiff(5);   // clause a
thermo.setOverride(true);  // clause b
thermo.setOverTemp(70);  // clause c
thermo.setMinLag(10);  // clause d
thermo.setTimeSinceLastRun(12);  // clause d
assertTrue (thermo.turnHeaterOn(settings));   // Run test
```

Correlated Active Clause Coverage

(1 of 6) (8.3.3)

Solve for Pa: ((a || (b && c)) && d)

P_a = ((T || (b && c)) && d) ⊕ ((F || (b && c)) && d)

(T && d) ⊕ ((b && c) && d)

d ⊕ ((b && c) && d)

Identity: (X ⊕ y&&X == !y&&X)

!(b && c) && d

(!b || !c) && d

Check with the logic coverage web app

http://cs.gmu.edu:8080/offutt/coverage/LogicCoverage

CACC

(a || (b && c)) && d

	a	b	c	d
P_a :	T	t	f	t
	F	t	f	t
P_b :	f	T	t	t
	f	F	t	t
P_c :	~~f~~	~~t~~	~~T~~	~~t~~
	~~f~~	~~t~~	~~F~~	~~t~~
P_d :	t	t	t	T
	t	t	t	F

duplicates

Six tests needed for CACC on Thermostat

CACC Values for Clauses

	curTemp	dTemp	thresholdDiff
a=t : curTemp < dTemp - thresholdDiff	63	69	5
a=f : !(curTemp < dTemp - thresholdDiff)	66	69	5

dTemp:

```
settings.setSettings (Period.MORNING, DayType.WEEKDAY, 69)
thermo.setPeriod (Period.MORNING);
thermo.setDay (Daytype.WEEKDAY);
```

	Override
b=t : Override	T
b=f : !Override	F

These values need to be placed into calls to turnHeaterOn() to satisfy the 6 tests for CACC

	curTemp	overTemp	thresholdDiff
c=t : curTemp < overTemp - thresholdDiff	63	72	5
c=f : !(curTemp < overTemp - thresholdDiff)	66	67	5

	timeSinceLastRun	minLag
d=t : timeSinceLastRun > minLag	12	10
d=f : !(timeSinceLastRun > minLag)	8	10

CACC Tests 1 & 2

dTemp = 69 (period = MORNING, daytype = WEEKDAY)

1. T t f t

```
thermo.setCurrentTemp (63);
thermo.setThresholdDiff (5);
thermo.setOverride (true);
thermo.setOverTemp (67); // c is false
thermo.setMinLag (10);
thermo.setTimeSinceLastRun (12);
```

2. F t f t

```
thermo.setCurrentTemp (66); // a is false
thermo.setThresholdDiff (5);
thermo.setOverride (true);
thermo.setOverTemp (67); // c is false
thermo.setMinLag (10);
thermo.setTimeSinceLastRun (12);
```

CACC Tests 3 & 4

dTemp = 69 (period = MORNING, daytype = WEEKDAY)

3. f T t t

```
thermo.setCurrentTemp (66); // a is false
thermo.setThresholdDiff (5);
thermo.setOverride (true);
thermo.setOverTemp (72); // to make c true
thermo.setMinLag (10);
thermo.setTimeSinceLastRun (12);
```

4. F f T t

```
thermo.setCurrentTemp (66); // a is false
thermo.setThresholdDiff (5);
thermo.setOverride (false); // b is false
thermo.setOverTemp (72);
thermo.setMinLag (10);
thermo.setTimeSinceLastRun (12);
```

CACC Tests 5 & 6

dTemp = 69 (period = MORNING, daytype = WEEKDAY)

5. t t t T

```
thermo.setCurrentTemp (63);
thermo.setThresholdDiff (5);
thermo.setOverride (true);
thermo.setOverTemp (72);
thermo.setMinLag (10);
thermo.setTimeSinceLastRun (12);
```

6. t t t F

```
thermo.setCurrentTemp (63);
thermo.setThresholdDiff (5);
thermo.setOverride (true);
thermo.setOverTemp (72);
thermo.setMinLag (10);
thermo.setTimeSinceLastRun (8); // d is false
```

Predicate Transformation

- ACC criteria are considered to be expensive for testers, and attempts have been made to reduce the cost
- One approach is to rewrite the program to eliminate multi-clause predicates, thus reducing the problem to branch testing (i.e., one clause per each predicate)
- A conjecture is that the resulting tests will be equivalent to ACC
- However, we explicitly advise against this approach for two reasons

Predicate Transformation Issues

1. The resulting rewritten program may have substantially more complicated control structure than the original (including repeated statements)
   – Endangering both reliability and maintainability. Second, as the following
2. The transformed program may not require tests that are equivalent to the tests for ACC on the original program

Program Transformation Issues

(8.3.4)

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

Transform (1) ?

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

Problems With Transformation 1

- We trade one problem for two problems :
  - Maintenance becomes harder
  - Reachability becomes harder
- Consider coverage :
  - CACC on the original requires four rows marked in the table
  - PC on the transformed version requires five different rows

a	b	c	(a∧b)∨c	CACC	PC_T
T	T	T	T		X
T	T	F	T	X	
T	F	T	T	X	X
T	F	F	F	X	X
F	T	T	T		X
F	T	F	F	X	
F	F	T	T		
F	F	F	F		X

- PC on the transformed version has two problems :
  1. It does not satisfy CACC on the original
  2. It is more expensive (more tests)

Program Transformation Issue 2

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

Transform (2) ?

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

Problems With Transformation 2

- We move complexity into computations
  - Logic criteria are not effective at testing computations
- Consider coverage :
  - CACC on the original requires four rows marked in the table
  - PC on the transformed version requires only two

a	b	c	(a∧b)∨c	CACC	PC_T
T	T	T	T		X
T	T	F	T	X	
T	F	T	T	X	
T	F	F	F	X	
F	T	T	T		
F	T	F	F	X	
F	F	T	T		
F	F	F	F		X

- PC on the transformed version becomes equivalent to clause coverage on the original
  - Not an effective testing technique

Transforming Does Not Work

Logic coverage criteria exist to help us develop better software

Circumventing the criteria is unsafe

Side Effects in Predicates (8.3.5)

- Side effects occur when a value is changed while evaluating a predicate
  - A clause appears twice in the same predicate
  - A clause in between changes the value of the clause that appears twice
- Example :

A && (B || A)	B is : changeVar (A)

B is a method call, changeVar (A), which has a side effect of changing the value of A

  - Evaluation : Runtime system checks *A*, then *B*, if *B* is false, check *A* again
  - But now *A* has a different value!
  - How do we write a test that has two different values for A in the same predicate?
- No clear answers to this controllability problem

We suggest a social solution : Go ask the programmer

Summary : Logic Coverage for Source Code

- Predicates from decision statements (if, while, for, etc.)
- Most predicates have less than four clauses
  - But some programs have a few predicates with many clauses
- The challenge is resolving internal variables
- Don't forget non-local variables
- If an input variable is changed within a method, it is treated as an internal variable thereafter
- Avoid transformations that hide predicate structure