Logic Coverage for Source Code

Idaho State University | Computer Science

Isaac Griffith

CS 4422 and CS 5599
Department of Computer Science
Idaho State University

Outcomes

At the end of Today's Lecture you will be able to:

- Apply logic coverage to source code
- Understand special issues that source transforms pose

Inspiration

“I am pretty sure there is a difference between ‘this has not been proven’ and ‘this is false.’ ” – Ron Jeffries

Logic Expressions from Source

- Predicates are derived from **decision** statements
- In programs, most predicates have **less than four** clauses
  - Wise programmers actively strive to keep predicates simple
- When a predicate only has one clause, COC, ACC, ICC and CC all collapse to **predicate coverage** (PC)
- Applying logic criteria to program source is hard because of **reachability** and **controllability**:
  - **Reachability**: Before applying the criteria on a predicate at a particular statement, we have to **get to** that statement
  - **Controllability**: We have to **find input values** that indirectly assign values to the variables in the predicates
  - Variables in the predicates that are not inputs to the program are called **internal variables**
- Illustrated through an example in the following slides...

Thermostat

```
// Jeff Offutt & Paul Ammann-September 2014
// Programmable Thermostat
import java.io.*;
public class Thermostat
{
    private int curTemp;          // Current temperature reading
    private int thresholdDiff;    // Temp difference until 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;         // OverridingTemp
    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)
    {
```

Thermostat (pg 2)

```
     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
}
```

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
```

- for a in 34: `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)

```
(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

$$\begin{aligned}
P_a &= ((a \,||\, (b \,\&\&\, c)) \,\&\&\, d) \oplus ((a \,||\, (b \,\&\&\, c)) \,\&\&\, d) \\
&((T \,||\, (b \,\&\&\, c)) \,\&\&\, d) \oplus ((F \,||\, (b \,\&\&\, c)) \,\&\&\, d) \\
&(T \,\&\&\, d) \oplus ((b \,\&\&\, c) \,\&\&\, d) \\
&d \oplus ((b \,\&\&\, c) \,\&\&\, d) \\
&T \oplus ((b \,\&\&\, c) \,\&\&\, T) \\
&!(b \,\&\&\, c) \,\&\&\, d \\
&(!b \,||\, !c) \,\&\&\, d
\end{aligned}$$

Check with the logic coverage web app at

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

Correlated Active Clause Coverage

```
(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**

Note: P_c are duplicates

Six tests needed for CACC on Thermostat

Correlated Active Clause Coverage

	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**

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

Correlated Active Clause Coverage

```
dTemp = 69 (period = MORNING, dayType = WEEKDAY)
```

❶ T t f t

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

❷ 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);
```

Correlated Active Clause Coverage

```
dTemp = 69 (period = MORNING, dayType = WEEKDAY)
```

❸ f T t t

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

❹ 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);
```

Correlated Active Clause Coverage

```
dTemp = 69 (period = MORNING, dayType = WEEKDAY)
```

❺ t t t T

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

❻ 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
```

Program Transformation Issues

Transform (1)?

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

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

a	b	c	$(a \wedge b) \vee c$	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

Program Transformation Issue 2

Transform (2)?

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

```
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 table
  - **PC on the transformed** version requires only two
- PC on the transformed version becomes equivalent to **clause coverage** on the original
  - **Not** an **effective** testing technique

a	b	c	$(a \wedge b) \vee c$	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

Transforming Does Not Work

Logic coverage criteria exist to help us make better software

Circumventing the criteria is unsafe

Side Effects in Predicates

- 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)`
  - 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 the same predicate?
- No clear answers to this controllability problem

We suggest a social solution: Go ask the programmer

Summary

- **Predicates** appear in decision statements (if, while, for, etc.)
- Most predicates have less than **four clauses**
  - But some programs have a few predicates with many clauses
- The hard part of applying logic criteria to source is usually resolving the **internal variables**
  - Sometimes setting variables requires calling **other methods**
- **Non-local variables** (class, global, etc.) are also input variables if they are used
- If an input variable is changed within a method, it is treated as an **internal variable** thereafter
- Avoid transformations that hide predicate structure

Are there any questions?