Programm- & Systemverifikation

Testing, Coverage & Invariants: Exercises

Georg Weissenbacher

184.741

TU WIEN

Coverage

```
bool sorted (int a,
      int b, int c)
{
  int i = 0;
  if (a < b)
    i = i + 1;
  if (b < c)
    i = i + 1;
  if (i == 2)
    return true;
  return false;
}
```

Inputs			Output
a	b	c	result
1	2	3	true
3	2	1	false

Coverage

```
bool sorted (int a,
      int b, int c)
{
  int i = 0;
  if (a < b)
    i = i + 1;
  if (b < c)
    i = i + 1;
  return (i == 2);
}
```

Inputs			Output
a	b	c	result
1	2	3	true
3	2	1	false

Control-Flow Coverage

Which of the following coverage criteria are satisfied?

(assume that the term “decision” refers to all Boolean expressions in the program)

	satisfied	
Criterion	yes	no
path coverage		
statement coverage		
branch coverage		
decision coverage		
condition coverage		

Which of the following coverage criteria are satisfied?

(the parameters of the function do not constitute definitions)

	satisfied	
Criterion	yes	no
all-defs		
all-c-uses		
all-p-uses		
all-c-uses/some-p-uses		
all-p-uses/some-c-uses		
all-uses		
all-du-paths		

Complete Coverage

- Augment the test-suite such to achieve full coverage
- If this is not possible, explain why

path-coverage

Inputs			Output
a	b	c	result

all-p-uses/some-c-uses

Inputs			Output
a	b	c	result

Modified Condition/Decision Coverage

Provide a test-suite that achieves full MC/DC coverage:

```
bool bar(int x, int y) {
  return ((x = y) && (y > 5));
}
```

Input		Output
x	y	result

More Coverage

Consider the following program:

```
bool subarr(int i, int j, int k)
  int maxsum = i;
  int lastsum = i;
  if (lastsum < 0)
    lastsum = j;
  else
    lastsum += j;
  if (lastsum > maxsum)
    maxsum = lastsum;
  if (lastsum < 0)
    lastsum = k;
  else
    lastsum += k;
  if (lastsum > maxsum)
    maxsum = lastsum;
  return maxsum;
}
```

Inputs			Output
i	j	k	result
-3	-1	2	2
3	-1	2	4

Which of the following coverage criteria are satisfied?

(assume that the term "decision" refers to all Boolean expressions in the program)

	satisfied	
Criterion	yes	no
path coverage		
statement coverage		
branch coverage		
decision coverage		
condition coverage		

Control-Flow Coverage

Which of the following coverage criteria are satisfied?

(assume that the term “decision” refers to all Boolean expressions in the program)

	satisfied	
Criterion	yes	no
path coverage		✓
statement coverage		
branch coverage		
decision coverage		
condition coverage		

Which of the following coverage criteria are satisfied?

(assume that the term “decision” refers to all Boolean expressions in the program)

	satisfied	
Criterion	yes	no
path coverage		✓
statement coverage	✓	
branch coverage		
decision coverage		
condition coverage		

Control-Flow Coverage

Which of the following coverage criteria are satisfied?

(assume that the term “decision” refers to all Boolean expressions in the program)

	satisfied	
Criterion	yes	no
path coverage		✓
statement coverage	✓	
branch coverage		✓
decision coverage		
condition coverage		

Control-Flow Coverage

Which of the following coverage criteria are satisfied?

(assume that the term “decision” refers to all Boolean expressions in the program)

	satisfied	
Criterion	yes	no
path coverage		✓
statement coverage	✓	
branch coverage		✓
decision coverage		✓
condition coverage		

Which of the following coverage criteria are satisfied?

(assume that the term “decision” refers to all Boolean expressions in the program)

	satisfied	
Criterion	yes	no
path coverage		✓
statement coverage	✓	
branch coverage		✓
decision coverage		✓
condition coverage		✓

Which of the following coverage criteria are satisfied?

(the parameters of the function do not constitute definitions)

	satisfied	
Criterion	yes	no
all-defs		
all-c-uses		
all-p-uses		
all-c-uses/some-p-uses		
all-p-uses/some-c-uses		
all-uses		
all-du-paths		

Data-Flow Coverage

Which of the following coverage criteria are satisfied?

(the parameters of the function do not constitute definitions)

	satisfied	
Criterion	yes	no
all-defs	✓	
all-c-uses		
all-p-uses		
all-c-uses/some-p-uses		
all-p-uses/some-c-uses		
all-uses		
all-du-paths		

Which of the following coverage criteria are satisfied?

(the parameters of the function do not constitute definitions)

	satisfied	
Criterion	yes	no
all-defs	✓	
all-c-uses		✓
all-p-uses		
all-c-uses/some-p-uses		
all-p-uses/some-c-uses		
all-uses		
all-du-paths		

Which of the following coverage criteria are satisfied?

(the parameters of the function do not constitute definitions)

	satisfied	
Criterion	yes	no
all-defs	✓	
all-c-uses		✓
all-p-uses	✓	
all-c-uses/some-p-uses		
all-p-uses/some-c-uses		
all-uses		
all-du-paths		

Which of the following coverage criteria are satisfied?

(the parameters of the function do not constitute definitions)

	satisfied	
Criterion	yes	no
all-defs	✓	
all-c-uses		✓
all-p-uses	✓	
all-c-uses/some-p-uses		✓
all-p-uses/some-c-uses		
all-uses		
all-du-paths		

Which of the following coverage criteria are satisfied?

(the parameters of the function do not constitute definitions)

	satisfied	
Criterion	yes	no
all-defs	✓	
all-c-uses		✓
all-p-uses	✓	
all-c-uses/some-p-uses		✓
all-p-uses/some-c-uses	✓	
all-uses		
all-du-paths		

Which of the following coverage criteria are satisfied?

(the parameters of the function do not constitute definitions)

	satisfied	
Criterion	yes	no
all-defs	✓	
all-c-uses		✓
all-p-uses	✓	
all-c-uses/some-p-uses		✓
all-p-uses/some-c-uses	✓	
all-uses		✓
all-du-paths		

Which of the following coverage criteria are satisfied?

(the parameters of the function do not constitute definitions)

	satisfied	
Criterion	yes	no
all-defs	✓	
all-c-uses		✓
all-p-uses	✓	
all-c-uses/some-p-uses		✓
all-p-uses/some-c-uses	✓	
all-uses		✓
all-du-paths		✓

Complete Coverage

- Augment the test-suite such to achieve full coverage
- If this is not possible, explain why

decision coverage

Inputs			Output
a	b	c	result
0	0	0	0

all-p-uses/some-c-uses

Inputs			Output
a	b	c	result

all-p-uses/some-c-uses already satisfied!

Decision coverage and MC/DC coincide for this example!

Testing

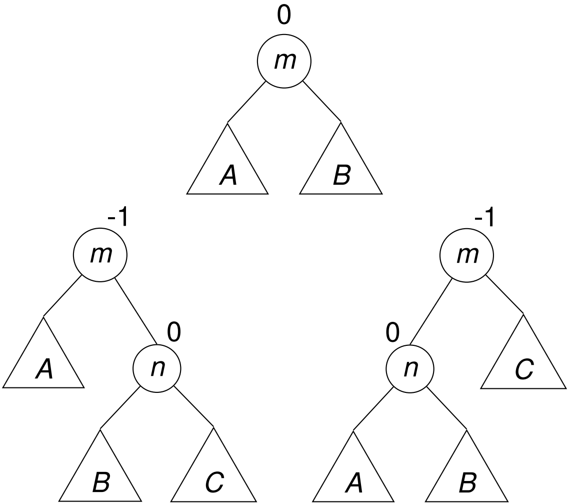
Let's test a balanced tree:

```
/* recursive tree structure            */
typedef struct _tree
{
  struct _tree * left;
  struct _tree * right;
  int element;
  int height;
} Tree;
```

- Test `insert (int e, Tree *t)`
- Conditions
  - Balanced: $|\text{left height} - \text{right height}| \leq 1$
  - Elements in left sub-tree are smaller than elements in right sub-tree

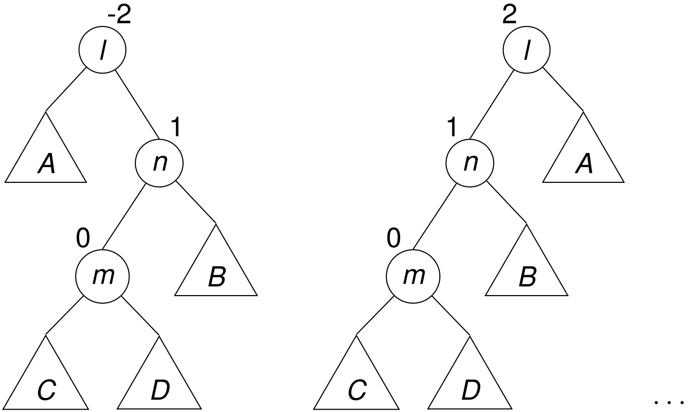
What do Trees Look Like?

Balanced Trees



What do Trees Look Like?

Unbalanced Trees



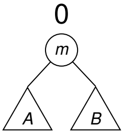
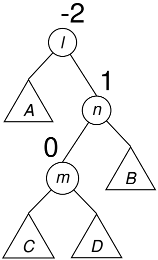
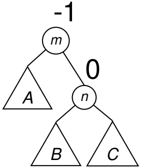
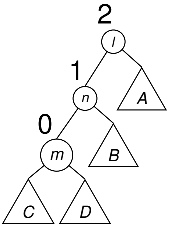
Derive valid and invalid equivalence classes for the function `insert`. Assign a unique number/id to each equivalence class.

Condition	Valid	ID	Invalid	ID

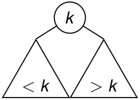
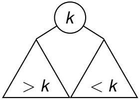
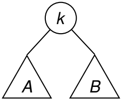
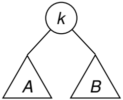
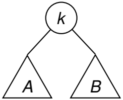
Equivalence Partitioning

- **Invalid** denotes *invalid* inputs (apparently not obvious?)
  - e.g., condition: “Tree is balanced”, invalid: not balanced
- One condition can result in multiple equivalence classes
  - e.g., “Tree is balanced”
  - valid: possible height differences: -1, 0, 1
  - invalid: possible height differences: -2, 2
- Also consider *output* equivalence classes
  - Especially for trees, there many (different balance!)
- Note: variable of type `int` in ANSI-C can’t be
  - a set $\{1, 2\}$
  - outside the range, e.g., $2^{32} + 1$

Equivalence Partitioning

Condition	Valid	ID	Invalid	ID
balanced	 insert $e > m$	1		2
–”–	$e < m$ 	3		4
	. . .			

Equivalence Partitioning

Condition	Valid	ID	Invalid	ID
ordered	0  $e > k$	5	0  $e > k$	6
no duplicates	0  $k \notin A \cup B$ $e > k$	7	0  $k \in A$ $e > k$	8
_"_			0  $k \in B$ $e < k$	9
	...			

Numerous other cases you could consider:

- Try to trigger rotations
  - e smaller than elements in left subtree A
  - e larger than elements in right subtree A
  - ...
- Try to insert elements already contained
  - $e \in A$, $e \in B$
  - Warning! These insertions are *not* invalid!
- Could also consider `null` as separate equivalence class
  - Warning! Insertion into empty tree *not* invalid!
- ...

Boundary Value Testing

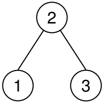
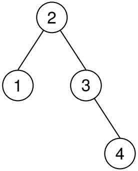
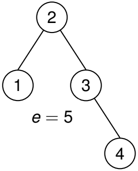
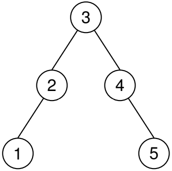
Use *Boundary Value Testing* to derive a test-suite for the method `insert`. Indicate which equivalence classes each test-case covers by referring to the numbers from before.

Input	**Output**	**Classes Covered**

Hint: in exam no points for *redundant* and *non-boundary* test cases

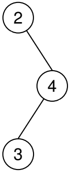
Boundary Value Testing

- “Boundaries” a bit unclear here, requires creativity
  - empty tree (`null`), tree with one element
  - “full” tree (all leaves filled)
  - two elements, leaning left/right
  - …

Input	Output	Classes Covered
0  $e = 4$	-1 	1,5,7
-1  $e = 5$	0 	...

Boundary Value Testing

Cover invalid classes **individually**!

Input	Output	Classes Covered
-2  $e = 5$	exception	**2**

Important:

- Specify **expected result** for test cases
- Test cases need to specify *concrete values*, also for output
- Which equivalence classes are covered? (enumerate them!)
  - Cover as many valid classes as possible with few test cases
  - Cover each invalid class with a *separate* test case
- Also cover output equivalence classes
  - Especially for trees, there many (different balance!)

Invariants

```
n = 0; y = x;
if (x % 2)
  x = x + 1;
else
  skip;
while (x > 42) {
  x = x / 2;
  n = n + 1; }
```

Are the following assertions loop invariants? If not, provide values for x, y, n, x′, y′ and n′ as a counterexample.

1. $n > 0$
2. $x \% 2 == 0$
3. $x \neq y$
4. $x = \lfloor \frac{y}{2^n} \rfloor$

Use Hoare's Calculus to prove the following Hoare Triple (assume that $\mathtt{x} \in \mathbb{N}_0$).

$\{\text{true}\}$

```
if ((x % 2) == 0)
  x = x + 1;
else
  skip;
while (x > 2)
  x = x - 2;
```

$\{\text{x} = 1\}$

Hoare's Axioms: Summary

$$\frac{}{\{P[E/\mathtt{x}]\}\ \mathtt{x:=}E\ \{P\}}$$

$$\frac{\{P\}\ C_1\ \{Q\}\ ,\ \{Q\}\ C_2\ \{R\}}{\{P\}\ C_1\ ;\ C_2\ \{R\}}$$

$$\frac{\{B \wedge P\}\ C_1\ \{Q\}\quad \{\neg B \wedge P\}\ C_2\ \{Q\}}{\{P\}\ \text{if } B \text{ then } C_1 \text{ else } C_2\ \{Q\}}$$

$$\frac{P' \rightarrow P \quad \{P\}\ C\ \{Q\} \quad Q \rightarrow Q'}{\{P'\}\ C\ \{Q'\}}$$

$$\frac{\{P \wedge B\}\ C\ \{P\}}{\{P\}\ \text{while } B \text{ do } C\ \{\neg B \wedge P\}}$$

Exam: June 12

- Solutions for Assignment 3 will be on TUWEL
- Pose questions about content on exam now or in TUWEL forum