6.0 ECTS/4.5h VU Programm- und Systemverifikation (184.741)

June 18, 2014

Kennzahl (study id)	Matrikelnummer (student id)	Familienname (family name)	Vorname (first name)	Gruppe (version)
				A

1.) Coverage

Consider the following program fragment and test suite:

```
int longest (unsigned a, unsigned b, unsigned c) {
  int result = a;
  if ((a + b > c) && (a + c > b) && (b + c > a)) {
    if (b > result)
        result = b;
    if (c > result)
        result = c;
    return result;
  } else {
    return -1;
  }
}
```

Inputs			Outputs
a	b	c	result
6	4	3	6
3	6	4	6
7	16	8	-1

(a) **Control-Flow-Based Coverage Criteria**

Indicate (✓) which of the following coverage criteria are satisfied by the test-suite above (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/decision coverage		

(5 points)

(b) **Data-Flow-Based Coverage Criteria**

Indicate (✓) which of the following coverage criteria are satisfied by the test-suite above (here, the parameters of the function do not constitute definitions, and the return statement `return result` is a c-use):

	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		

(7 points)

(c) *If* the test-suite from above does not satisfy the coverage criteria listed below, augment it with test-cases such that these criteria are satisfied. If full coverage cannot be achieved for one or more of these criteria, explain why.

MC/DC

Inputs			Outputs
a	b	c	result

all-p-uses/some-c-uses

Inputs			Outputs
a	b	c	result

(5 points)

(d) Provide sufficiently many test-cases to guarantee modified condition/decision coverage for the following program fragment, or explain why this coverage metric can't be satisfied:

```
bool foo(unsigned a, unsigned b, unsigned c) {
  return ((a + b > c) || (a > c));
}
```

Input			Output
a	b	c	result

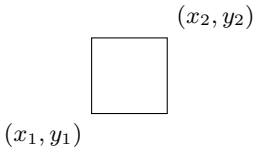
(3 points)

2.) Black-Box Testing

The function

`int inside(int` x_0`, int` y_0`, int` x_1`, int` y_1`, int` x_2`, int` y_2`)`

determines whether the point (x_0, y_0) lies inside, on, or outside the following rectangle:



- The point (x_1, y_1) is the lower left corner, and (x_2, y_2) is the upper right corner.
- The sides of the rectangle must have a non-zero length.
- `inside` returns
  - 1 if (x_0, y_0) lies *inside* the rectangle
  - 0 if (x_0, y_0) lies *on* a side of the rectangle
  - -1 if (x_0, y_0) lies *outside* the rectangle

(a) **Equivalence Partitioning**

From the conditions above, derive the equivalence classes for the method `inside`. Use the table below to partition them into *valid equivalence classes* (valid inputs) and *invalid equivalence classes* (invalid inputs). Label each of the equivalence classes clearly with a number (in the according column). For each correct *equivalence class* you can score one point (up to 10 points).
(Do not provide test-cases here – that's task (b))

Condition	**Valid**	**ID**	**Invalid**	**ID**

(10 points)

(b) **Boundary Value Testing**

Use *Boundary Value Testing* to derive a test-suite for the method `inside`. Specify the inputs points $p_0 = (x_0, y_1), p_1 = (x_1, y_2), p_2 = (x_2, y_2)$ (e.g., $p_1 = (1, 2), p_2 = (0, 0), p_3 = (4, 4)$). Indicate clearly which equivalence classes each test-case covers by referring to the numbers from task (a). You receive one point for each *correct* test-case (up to 15 points, where test-cases that do not represent boundary values do not count).

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

(15 points)

3.) (a) **Invariants**

Given the code example:

```
int a, x, y, n;
x = 0;
y = 0;
n = 0;
if (a > 0) {
  while (y < 2013) {
    n = n + 1;
    y = y + a;
    x = x + y;
  }
}
```

Consider the following formulas; are they loop invariants? If not, give the values of a, x, y, n, a', x', y', and n' that show that it is not an invariant.

i. $a > 0$

ii. $y = a \cdot n$

iii. $y = n$

iv. $x = a \cdot y \ \wedge \ y = n \cdot a$

v. $x = \frac{a \cdot n \cdot (n+1)}{2} \ \wedge \ y = a \cdot n$

(10 points)

(b) **Hoare Logic**

Prove the Hoare Triple below (assume that the domain of all variables in the program are the natural numbers including 0). You need to find a sufficiently strong loop invariant. Annotate the following code directly with the required assertions. Justify each assertion by stating which Hoare rule you used to derive it.

{true}

```
if (x > y) {

  t := x;

  x := y;

  y := t;

} else {

  skip;

}

while (x<y) {

  x := x + 1;

}
```

$\{x = y\}$

(10 points)

4.) Decision procedures

(a) Bring the following formula into conjunctive normal form (CNF) using *Tseitin transformation*.

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

(5 points)

(b) Consider the following formulas in Equality Logic; are they satisfiable? If yes, provide a satisfying assignment over integers, if not, give the reasoning based on equivalence classes that leads to this conclusion.

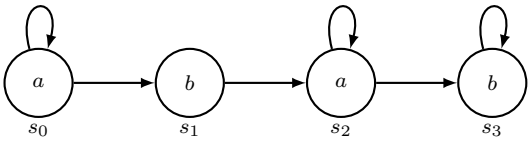
i. $a = b \wedge c \neq d \wedge a = d \wedge e = c \wedge e \neq b \wedge c \neq b \wedge d = b$

ii. $a \neq b \wedge c \neq d \wedge c = g \wedge b = c \wedge e = f \wedge f = a \wedge g = e$

(6 points)

5.) Temporal Logic

Consider the following Kripke Structure:



(a) Fix s_0 as the initial state and give the computation tree for three steps. **(6 points)**

(b) For each formula, in which states of the Kripke structure does it hold? (Note that we do not consider s_0 as special initial state here.)

i. $b \wedge \mathbf{A}\,\mathbf{X}\,a$

ii. $\mathbf{E}\,\mathbf{F}\,\mathbf{G}\,a$

iii. $\mathbf{AF}\,\mathbf{AG}\,a \vee \mathbf{AF}\,\mathbf{AG}\,b$

iv. $\mathbf{E}\,\mathbf{X}\,b \wedge (\mathbf{E}\,\mathbf{X}\,(\mathbf{E}\,\mathbf{X}\,b))$

(8 points)