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

June 27, 2013

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:

```
bool minmax(int i, int j, int k,
            int &least, int &most) {
  least = i;
  most = i;
  if (most < j)
    most = j;
  if (most < k)
    most = k;
  if (least > j)
    least = j;
  if (least > k)
    least = k;
  return (least ≤ most);
}
```

Inputs			Outputs		
i	j	k	least	most	result
1	2	3	1	3	true
3	2	1	1	3	true

(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):

	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.

all-c-uses

Inputs			Outputs		
i	j	k	least	most	result

all-p-uses/some-c-uses

Inputs			Outputs		
i	j	k	least	most	result

MC/DC

Inputs			Outputs		
i	j	k	least	most	result

(5 points)

(d) Provide sufficiently many test-cases to guarantee modified condition/decision coverage for the following program fragment:

```
bool foo(int x, int y) {
  return ((x < y) || (y % 2 >= 1));
}
```

Input		Output
x	y	result

(3 points)

2.) Black-Box Testing

The method `valueOf(String s)` of the class `at.forsyte.Float` returns a `Float` object holding the float value represented by the argument `s`.

- If `s` is null, then a `NullPointerException` is thrown.
- Leading and trailing whitespace characters in `s` are ignored.
- If `s` does not represent a valid floating point value, a `NumberFormatException` is thrown.
- If `s` is longer than 10 characters, a `TooLongForExamException` is thrown.

All floating point values are expressed in decimal (base 10). A valid floating point value starts with an *optional* sign (−, +), followed by a whole number part and/or a fraction part (at least one of which has to be present). The whole number part can be either a non-empty sequence of digits (0-9), or the string `NaN` or `Infinity`. In the latter two cases, the whole number part must not be followed by a fraction part. The fraction part starts with a decimal point followed by a non-empty sequence of digits.

(a) **Equivalence Partitioning**

From the conditions above, derive the equivalence classes for the method `valueOf`. 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 `valueOf`. Specify the input as a string. If the expected output is a (valid) `Float` object, use a floating point number to represent it. 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;
assume (a > 0);
while (y < 2013) {
  n = n + 1;
  x = x + a;
  y = y + x;
}
```

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. $x = a \cdot n$

iii. $y = n$

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

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

(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) {

  a = x;

  b = y;

} else {

  a := y;

  b := x;

}

while ((a-b)>0) {

  a = a-1;

  }
```

$\{a = b\}$

(10 points)

4.) Decision procedures

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

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

(5 points)

(b) Consider the following formulas; are they satisfiable? If yes, provide a satisfying assignment, if not, give the reasoning that leads to this conclusion.

i. $(a \rightarrow b) \wedge (\neg a \vee \neg b)$

ii. $\neg a \wedge (b \vee c) \wedge (\neg b \vee c) \wedge (\neg b \vee \neg a) \wedge (\neg c \vee a)$

iii. $(a \rightarrow b) \wedge (\neg a \vee \neg b) \wedge (a \vee b) \wedge (a \wedge \neg b)$

(6 points)

(c) 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. $x_4 = x_7 \ \wedge \ x_2 = x_1 \ \wedge \ x_3 = x_2 \ \wedge \ x_7 \neq x_5 \ \wedge \ x_1 = x_6 \ \wedge \ x_6 \neq x_4 \ \wedge \ x_7 = x_3$

ii. $x_1 = x_3 \ \wedge \ x_1 \neq x_5 \ \wedge \ x_2 = x_4 \ \wedge \ x_2 = x_5 \ \wedge \ x_3 \neq x_4 \ \wedge \ x_3 \neq x_5 \ \wedge \ x_4 = x_5$

(4 points)