VU Advanced Software Engineering WS2025 - Test 1, Group B	Duration: 90min 14.11.25
Last name: First name:	Student ID:

This is only what I remembered from the test and may contain parts, which where not part of the test or misses parts from the test. (so not sound nor complete)

1.) Single Choice Questions (12 points)

In the following, you find some single-choice questions. You get +2 points if you check exactly one answer and this answer is correct.

a) Which abstract domain does not require a widening operator to ensure termination of Abstract Interpretation?

- ☐ the octagon relational domain.
- ☐ the standard interval domain.
- ☐ every domain whose elements are a subset of the elements of the standard interval domain.
- ☐ the parity domain from the lecture.

b) A program analysis can be safely used for verification if it is

- ☐ sound.
- ☐ precise.
- ☐ imprecise.
- ☐ complete.

c) How many binary variables can there be after the bit-blasting step of Bounded Model Checking, assuming that the constraints have 2 integer variables with bit-width 32?

- ☐ 2
- ☐ 32^2
- ☐ 64
- ☐ 2^{32}

d) An unsound analysis

- ☐ is definitely an over-approximation of a program.
- ☐ is definitely an under-approximation of a program.
- ☐ could be neither an over-approximation nor an under-approximation of a program.

e) In Bounded Model Checking, using unrolling assertions in the presence of loop unrolling makes the analysis

- ☐ unsound and precise.
- ☐ unsound and imprecise.
- ☐ sound and precise.
- ☐ sound and imprecise.

f) If an analysis did emit a warning for a correct program, the analysis is considered to be

- ☐ unsound.
- ☐ precise.
- ☐ imprecise.
- ☐ sound.

2.) Abstract Interpretation 1 (26 points)

Consider the modified interval domain INT_d with the following set of abstract elements: $\{[x, x+d] \mid 0 \leq d \leq 2 \text{ and } x \geq 0\} \cup \{\bot, [0, \infty)\}$. INT_d cannot represent negative numbers which is why we restrict ourselves here to programs over unsigned integers.

a) Perform Abstract Interpretation with INT_d on function `check_verified1` below. Provide the abstract control flow graph as in the lecture. What does the analysis reveal about the safety of `check_verified1`? Reason about the semantics of the relevant abstract state(s). (14 points)

```
0   void check_verified1(unsigned int x) {
1       if(x < 3){
2           x = x + x;
3       } else {
4           x = 1;
5       }
6       assert(x <= 4);
7   }
```

b) Perform Abstract Interpretation with INT_d on function `check_verified2` below. Provide the abstract control flow graph as in the lecture. What does the analysis reveal about the safety of `check_verified2`? Reason about the semantics of the relevant abstract state(s). (12 points)

```
0   void check_verified2(unsigned int y) {
1       unsigned int x = 3;
2       if(y <= 1){
3           y++;
4           x = y;
5       }
6       assert(x != 0);
7   }
```

3.) Abstract Interpretation 2 (18 points)

Give a function `precision_loss` (in the C-like syntax used in the lecture) containing an assertion `assert(...)` that shows that the widening operator ∇ for Abstract Interpretation with the standard interval domain as used in the lecture comes with precision loss. Give the abstract state(s) "before" the assertion (i.e. the state(s) for which the assertion is evaluated) when performing Abstract Interpretation both with and without widening.

Hint: Think about which properties the function has to satisfy.

4.) Bounded Model Checking (26 points)

a) Use Bounded Model Checking to verify the correctness of function `f_arith` below. Assume for this analysis that `int` overflows can occur.

- Show the program after the step of converting it into SSA form. (6 points)
- Show the constraints obtained from the SSA form and UNSAT or SAT assignment. (4 points)
- Are the constraints satisfiable? Show that your answer is correct. Recall that satisfiability is shown by a proof argument over the constraints! (10 points)

```
void f_arith(int x, int y) {
    if(y <= 0) {
        if(x > 0) {
            x = -x;
            x = x*x? 2*x; x1
        }
        y = x * 2;
    } else {
        y = -x - y;
    }
    x = -x;
    assert(y < x);
}
```

b) Now assume we want to statically analyze the C-function `f_array` below with Bounded Model Checking. You can assume the existence of assertions check overflow errors using Bounded Model Checking. For each of the following settings, tick what would be the expected result of the SAT-solver and — in case the result is SAT — provide the assignment that the SAT-solver would give for variable `n` (the variable that is used to encode `n` in `f_array`). (6 points)

```
int arr[3];
void f_array(unsigned int n){
    unsigned int i = 0;
    while(i < n){
        arr[i] = 0;
        i++;
    }
}
```

- Unrolling assertions with unrolling bound of 2: ☐ SAT ☐ UNSAT, assignment for `n`:
- Unrolling assumptions with unrolling bound of 2: ☐ SAT ☐ UNSAT, assignment for `n`:

5.) Symbolic Execution (18 points)

Consider function `wh1` below. Perform Symbolic Execution on the path that corresponds to an execution where the loop body is executed exactly two times. Assume for your analysis that no over-/underflows can occur.

```
void wh1(int a, int b) {
    while (a >= 0) {
        b = 42 - b;
        a = a * b;
    }
}
```

a) Complete the table below like in the lecture. Make sure you are using the labels of the CFG nodes in the column "Path". (14 points)

Path	**Symbolic map**	**Path condition**
1	$a \to A, b \to B$	true

b) Use the obtained path constraint and give a possible assignment for variable a and b such that the described path would be executed. For simplicity, you can assume that mathematical integers are used, so no over-/under-flows occur. *(4 points)*