Victoria University of Wellington
School of Engineering and Computer Science

SWEN224: Formal Foundations of Programming

Exam Prep Questions

1 Static Analysis

These questions are *illustrative* of the kinds of questions you could expect on static analysis in the exam.

1. **(5 marks)** Briefly, discuss what is meant by the term *static analysis*.
2. **(5 marks)** Briefly, discuss an example of a *static analysis* that is in everyday use.
3. **(5 marks)** Briefly, discuss the difference between *compile-time* and *run-time*.
4. **(5 marks)** Briefly, discuss how the use of static analysis can help to find errors in software.
5. **(5 marks)** Briefly, discuss what *conservatism* means in the context of static analysis.
6. **(6 marks)** For each *parameter*, *return* and *field* in the following program, insert `@NonNull` or `@Nullable` annotations (where appropriate) by writing in the box.

```
public class ArraySet {
    private Object[] items;
    private int count; // counts number of elements currently used.

    public ArraySet(int n) {
        this.items = new Object[n];
        this.count = 0;
    }

    public void add(Object item) {
        items[count] = item;
        count = count + 1;
    }

    public boolean contains(Object o) {
        for(int i=0;i!=items.length;++i) {
            if(items[i].equals(o)) {
                return true;
            }
        }
        return false;
    }
}
```

2 Specification & Verification

These questions are *illustrative* of the kinds of questions you could expect on specification and verification in the exam.

1. **(5 marks)** In your own words, discuss the benefits of providing specifications for code.

2. **(5 marks)** Specifications are often described as contracts between the *client* and *supplier*. Briefly, discuss what is meant by this and how the two roles interact.

3. **(2 marks)** State in English what the following specification says:

```
function add(int x, int y) -> (int z)
requires 0 <= x && x <= 5
requires 0 <= y && y <= 5
ensures 0 <= z && z <= 10:
    //
    return x + y
```

4. **(2 marks)** State in English what the following specification says:

```
function zeroOut(int[] items, int start) -> (int [] r)
requires start >= 0 && start < |items|
ensures all { k in start..|r| | r[k] == 0 }:
    //...
```

5. **(3 marks)** Briefly, discuss whether or not the following implementation of `zeroOut()` meet its specification.

```
function zeroOut(int[] items, int start) -> (int [] r)
requires start >= 0 && start < |items|
ensures all { k in start..|r| | r[k] == 0 }:
    //...
    int i = 0
    while i < |items|:
        items[i] = 0
        i = i + 1
    //
    return items
```

6. **(3 marks)** Provide an appropriate *loop invariant* for the `zeroOut()` function above.

7. **(5 marks)** Loop invariants differ from pre- and post-conditions as they do not form part of a function's specification. Briefly, discuss what the purpose of a loop invariant is.

8. **(5 marks)** Briefly, discuss why the following implementation *does not* meet its specification. You should provide *parameter* and *return* values to illustrate.

```
function isSorted(int[] arr) -> (bool r)
ensures r ==> all { i in 1 .. |arr| | arr[i-1] <= arr[i] }
ensures !r ==> some { i in 1 .. |arr| | arr[i-1] > arr[i] }:
    //
    int i = 1
    int last = arr[0]
    //
    while i < |arr|:
        //
        if last > arr[i]:
            return false
        i = i + 1
        last = arr[i]
    //
    return true
```

9. **(8 marks)** Provide an updated implementation of `isSorted()` which does meet its specification. You should additionally include an appropriate *loop invariant* to ensure that it will verify.

10. **(6 marks)** Next to each numbered comment in the program below, give appropriate logical conditions which are true at that point in the program.

```
function diff(int x, int y) -> (int z)
requires x >= 0 && x <= 255 && y >= 0 && y <= 255
ensures z >= 0
ensures z == (x-y) || z == (y-x):
    //
    int r
    //
    //(1)
    //
    if x > y:
        //
        //(2)
        //
        r = x - y
        //
        //(3)
        //
    else:
        //
        //(4)
        //
        r = y - x
        //
        //(5)
        //
    //
    return r
```

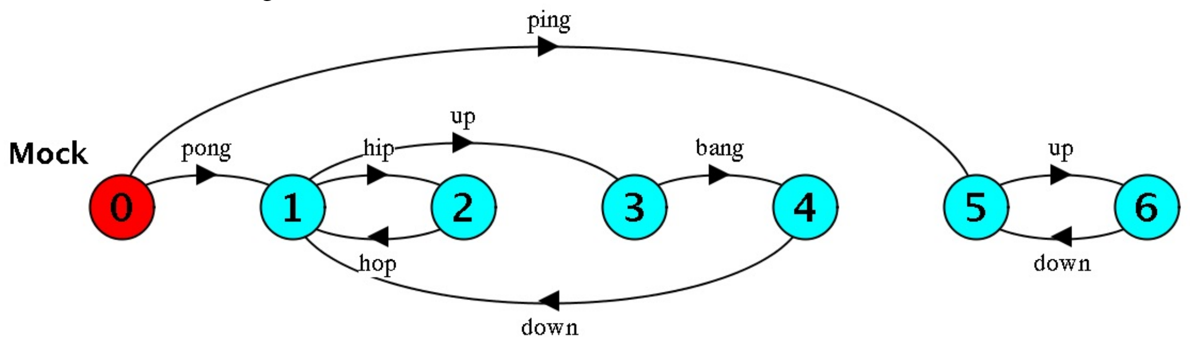
11. **(5 marks)** Briefly, discuss why the precondition given for the following function is not the *weakest precondition*.

```
function lastIndexOf(int[] items, int item) -> (int r)
requires all { k in 0..|items| | items[k] >= 0 }
ensures r >= 0 ==> items[r] == item
ensures r >= 0 ==> all { i in r+1 .. |items| | items[i] != item }
ensures r < 0 ==> all { i in 0 .. |items| | items[i] != item }:
    //...
    int i = |items|
    while i > 0
    where i <= |items|
    where all { k in i .. |items| | items[k] != item }:
        i = i - 1
        if items[i] == item:
            return i
    //
    return -1
```

12. **(5 marks)** The Whiley verification system ensures that functions are *partially correct* but it does not ensure they are *totally correct*. Briefly, discuss what this means.

3 Checking Requirements:

These questions are *illustrative* of the kinds of questions you could expect on requirements checking in the exam. Consider the following automaton.



Will **Mock**, above, satisfy the following statements (Yes/No)

1. **(2 marks)** `progress T = {hip}`
2. **(2 marks)** `progress T = {hop,down}.`
3. **(2 marks)** `property Sh = (pong->hip->down->Sh).`
4. **(2 marks)** `property St = (up->down->St)..`

4 Modeling

Cook Modeling a Simple cooker

When a cooker, **Cook** is switched on, **switchOn** you can assume that the cooker is cold and the element will be turned **on** and the cooker will **heat** up. Once the cooker is hot it will turn **off** and **cool** before turning itself **on** again.

Requirement. Once the cooker **Cook** is switched on it can not be switched off and forever cycles around being cold then hot then cold, ...

Only use events: **switchOn**, **on**, **heat**, **off** and **cool**.

1. **(2 marks)** Draw the automaton of the **Cook** automaton.
2. **(2 marks)** Specify the **Cook** process using the LTSA language.
3. **(2 marks)** Briefly, discuss whether the **requirement** given above is a *safety* or *liveness* requirement.

Cooker a simple cooker with an off switch.

Build a **Cooker** process by extending the **Cook** process in the previous question, by adding the ability to switch off the cooker. After the user switches off the cooker, the **switchOff** event, the cooker must turn itself off if it is currently on. When off the cooker may still need to cool down before it returns to it initial state.

Only use events: **switchOn**, **switchOff**, **on**, **heat**, **off** and **cool**.

1. **(3 marks)** Draw the automaton of the **Cooker** automaton.
2. **(3 marks)** Specify the **Cooker** process using the LTSA language.

Car a simple specification

A **CarSmp** is initially at rest and not running. It can **start** after which it is running. when running it can **stop**. In addition the car can **acc**elerate and then **break**. Before it has started it cannot accelerate.

1. **(3 marks)** Draw the automaton of the **CarSmp** specified above using only the events **start**, **stop**, **acc** and **break**.
2. **(3 marks)** Specify the **CarSmp** process using the LTSA language.

Car again

Car is like the previous **CarSmp** model except that is can travel at speeds 0,1,2,3,....N and to accelerate from rest to speed i it must **acc**elerate i times, and to stop from speed i it must **brake** i times.

1. **(2 marks)** Draw the automaton of the **CarSmp** specified above using only the events **start**, **stop**, **acc** and **brake** when **N=3**
2. **(2 marks)** Specify the **Car** process using the LTSA language.

CarMax and indexed process.

The **CarMax** process is the **Car** process from the previous question with the ability to set the maximum speed of the car to some input parameter.

Only use the events **start**, **stop**, **acc**, **brake** and **setMax**

1. **(5 marks)** Draw the automaton of the **CarMax** specified above when the constant **N = 2**.
2. **(5 marks)** Specify the **CarMax** process using the LTSA language.