TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI

VUW VICTORIA UNIVERSITY OF WELLINGTON

Student ID: .....

EXAMINATIONS — 2015

TRIMESTER 2

SWEN224

Formal Foundations of Programming

Time Allowed: ONE HOUR

Instructions: Answer all questions
All questions are of equal value

Answer all questions in the boxes provided.
Every box requires an answer.
If additional space is required you may use a separate answer booklet.

No calculators permitted.
Non-electronic Foreign language to English dictionaries are allowed.

Question	Topic	Marks
1.	Static Analysis	20
2.	Model Checking Lifts	20
3.	Model Checking Vending Machines	20
	Total	60

Question 1. Static Analysis [20 marks]

(a) [2 marks] Briefly, discuss what the `@NonNull` annotation means.

This means that the annotated variable or field cannot hold the **`null`** value

(b) [2 marks] Briefly, discuss what the `@Nullable` annotation means.

This means that the annotated variable or field may hold the **`null`** value

(c) [2 marks] Briefly, discuss whether or not `@NonNull` is a *subtype* of `@Nullable`.

Yes, `@NonNull` is a subtype of `@Nullable`. This is because the set of values represented by `@NonNull` is a *subset* of that represented by `@Nullable`

(d) [4 marks] A static analysis, such as non-null analysis, operates at *compile time*. Briefly, discuss the benefits of this over mechanisms which operate at *runtime* (e.g. testing or runtime assertions).

A static analysis operates before the program is run and catch errors prior to deployment. Static analyses explore all possible execution paths and can guarantee that a program are free from a specific kind of error (e.g. `NullPointerException`). In contrast, mechanisms which operate at runtime can only consider a small set of the possible input values and are not guaranteed to find all possible problems. Thus, errors may still occur after deployment.

(e) [6 marks] For each *parameter*, *return* and *field* in the following program, insert `@NonNull` or `@Nullable` annotations (where appropriate) by writing in the box.

```
1  class Employee {
2      private @NonNull String name;      // Every employee has a name
3
4      private int age;                   // Every employee has an age
5
6      private @Nullable String address; // Employees may not have an
   address
7
8      public Employee(@NonNull String name, int age, @Nullable String address)
9
10         this.name = name;
11         this.age = age;
12         this.address = address;
13     }
14
15     public @NonNull String getName() { return name; }
16
17     public int getAge() { return age; }
18
19     public @Nullable String getAddress() { return address; }
20
21     public @NonNull String toString() {
22
23         String r = name + ":" + age;
24         if(address != null) { r = r + ":" + address; }
25         return r;
26     }
27 }
```

(f) [4 marks] `@NonNull` and `@Nullable` annotations are *contra-variant* for parameter types and *co-variant* for return types. Briefly, discuss what this means.

Consider a method in a subclass which overrides a method in a superclass. The parameter types for the subclass's method may only widen (not narrow) those types in the superclass. Thus, a `@NonNull` parameter in the superclass's method may be widened to `@Nullable` in the subclass (but not the other way around).

For return types, the opposite is true. Thus, a `@Nullable` return type in the superclass's method may be narrowed to `@Nullable` in the subclass (but not the other way around)

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Student ID: .....

Question 2. Model Checking Lifts [20 marks]

(a) The LTSA language:

(i) [1 mark] LTSA events are most similar to:

A button pushing events,

B sending an email.

Answer A or B.

A

(ii) [4 marks] Explain the two sorts of requirements using one short sentence (or phrase) each

Safty requirement

nothing bad happens

Liveness requirement

something good happens

(iii) [1 mark] Is it Safety or a Liveness requirements that can be satisfied by processes that do nothing?

Answer **Safety** or **Liveness**.

Safety

This rest of this questions is about specifying a simple Lift system using the event based LTSA language and automata. The Lift system consists of a **Cage** that can move between two floors and two doors, one at each floor. Each of the doors is modeled by separate processes.

(b) The movement of a lift Cage is modeled using only the two events:

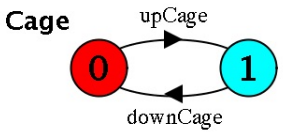
upCage models the movement of the cage from floor one to floor two and

downCage models the movement of the cage from floor two to floor one.

(i) [3 marks] In the LTSA language specify the `Cage` process that starts on floor one

$Cage = (upCage- > downCage- > Cage).$

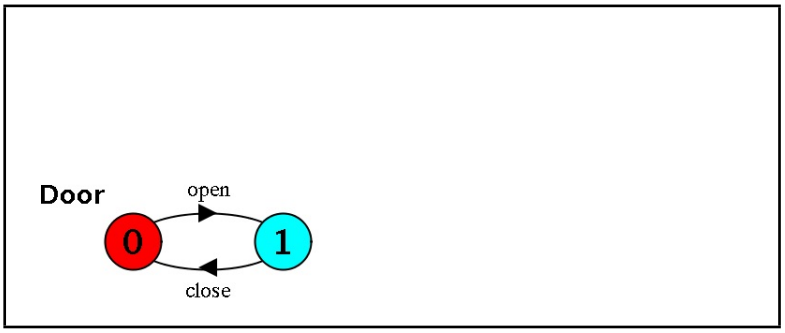
(ii) [3 marks] Draw the automata of the specified **Cage** process



(c) The specification of the doors, on floor one and floor two have a lot in common. Using process labeling we formally model the similarity between the two doors.

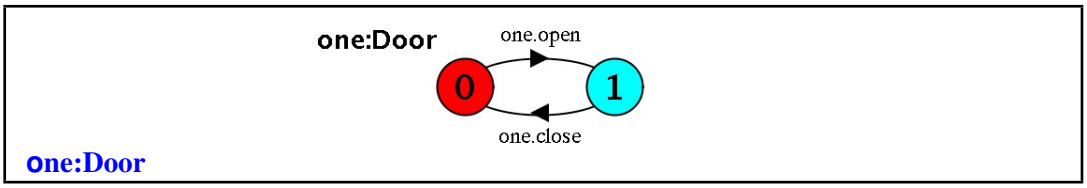
(i) [3 marks] Draw the automata defined by the LTSA specification of a door:

```
Door = (open->close->Door).
```



(ii) [3 marks] By applying labels **one** and **two** to the above **Door** process you can specify the two doors.

Write LTSA command labeling the **Door** process with the label **one** and draw the automata of this process.



(iii) [2 marks] There are two requirements for the Lift system:

Requirement 1 the door on each of the floors only opens when the lift is on the same floor.

Requirement 2 the lift must not get stuck on either of the floors

Is **Requirement 1** a Safety or a Liveness requirement?

Answer **Safety** or **Liveness**.

Safety

Is **Requirement 2** a Safety or a Liveness requirement?

Answer **Safety** or **Liveness**.

Liveness

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Student ID: .....

SPARE PAGE FOR EXTRA ANSWERS

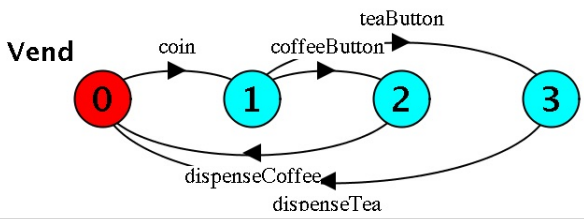
Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Question 3. Model Checking Vending Machines [10 marks]

(a) A specification for the simple vending machine **Vend** is given below.

```
Vend = (coin -> Ready),
   Ready = (teaButton -> dispenseTea -> Vend
           | coffeeButton -> dispenseCoffee -> Vend).
```

(i) [3 marks] Draw the automata for **Vend**:



(ii) [2 marks] Specify, in LTSA, the requirement that: always at some point in the future **Vend** will accept a **coin**

$progress\ Coin = \{coin\}$

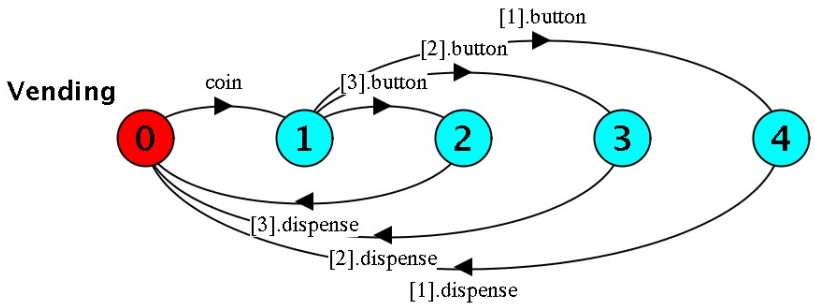
(iii) [2 marks] Specify, in LTSA, the requirement that **Vend** is always fair. That is, each drink is paid for and you always get a drink after you pay.

$property\ Fair = (coin- > (dispenseTea- > Fair|dispenseCoffee- > Fair)).$

(b) [4 marks] Specify a process **Vending** that has **N** buttons, each button dispensing a different drink.

$$Vending = (coin- > Ready),$$
$$Ready = ([i : 1..N].button- > [i].dispense- > Vending).$$

(c) [3 marks] Draw the automata for the process **Vending** when **const N = 3**



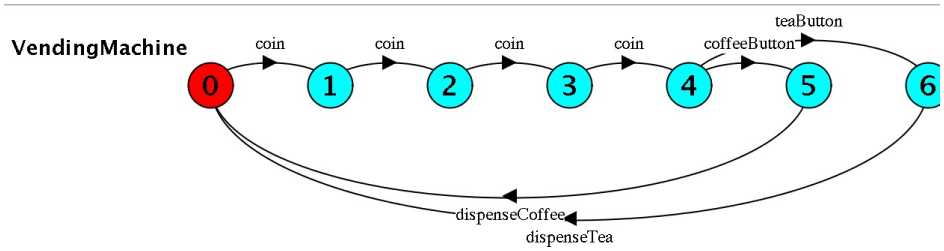
(d) [3 marks] Specify a process **VendingMachine** that needs **M** coins before it responds to **teaButton** by dispensing tea or responds to **coffeeButton** by dispensing coffee.

$$VendingMachine = (coin->R[0]),$$
$$R[j:0..M-1] = (coin->R[j+1]),$$
$$R[M] = (teaButton->dispenseTea->VendingMachine$$
$$|coffeeButton->dispenseCoffee->VendingMachine).$$

OR

$$VendingMachine2 = (coin->R[0]),$$
$$R[j:0..M] = (when(j<M)coin->R[j+1]$$
$$|when(j==M)teaButton->dispenseTea->VendingMachine2$$
$$|when(j==M)coffeeButton->dispenseCoffee->VendingMachine2).$$

(e) [3 marks] Draw the automata for the process **Vending** when **const M = 4**



Student ID: .....

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

****