EECS 4315 3.0 Mission Critical Systems

A Solution to the Midterm

9:15–10:15 on February 27, 2019

1 (3 marks)

Fill in the blanks.

(a) A class that contains unit tests is known as a ....

Answer: test case.

Marking scheme: 1 mark for the correct answer.

(b) The code to be tested is known as the ....

Answer: system under test.

Marking scheme: 1 mark for the correct answer.

(c) One of the major challenges in model checking, namely that the state space may become very large, is known as the ....

Answer: state space explosion problem.

Marking scheme: 1 mark for the correct answer.

2 (3 marks)

Give three reasons why we are interested in *randomization*.
Answer:

1. The source code of most computer and video games contains some sort of randomization. This provides games with the ability to surprise players, which is a key factor to their long-term appeal.

2. Randomization may reduce the expected running time or memory usage.

3. Randomization may allow us to solve problems.

Marking scheme: 1 mark for each correct reason.

3 (2 marks)

Nondeterministic code is code that, even for the same input, can exhibit different behaviours in different runs. Give two *limitations* of testing in the context of nondeterministic code.
Answer:

1. No guarantee that all different behaviours have been checked.

2. Errors may be difficult to reproduce.

Marking scheme: 1 mark for each correct limitation.

4 (4 marks)

(a) Complete the following app that prints 1 with probability $\frac{1}{2}$ and prints 2 and 3, each with probability $\frac{1}{4}$.

Answer:

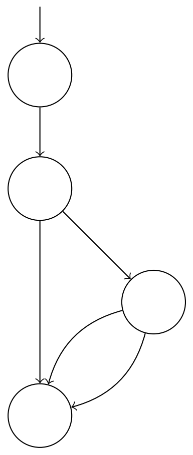
```
import java.util.Random;

public class Example {
  public static void main(String[] args) {
    Random random = new Random();
    if (random.nextBoolean()) {
      System.out.println("1");
    } else {
      if (random.nextBoolean()) {
        System.out.println("2");
      } else {
        System.out.println("3");
      }
    }
  }
}
```

Marking scheme:

- 1 marks for creating a Random object.
- 1 marks for using methods of the Random class.
- 1 mark for getting the probabilities right.

(b) Draw the corresponding state space.

Answer:



Marking scheme: 1 mark if the branching points in the diagram correspond to the branching points in the code.

5 (4 marks)

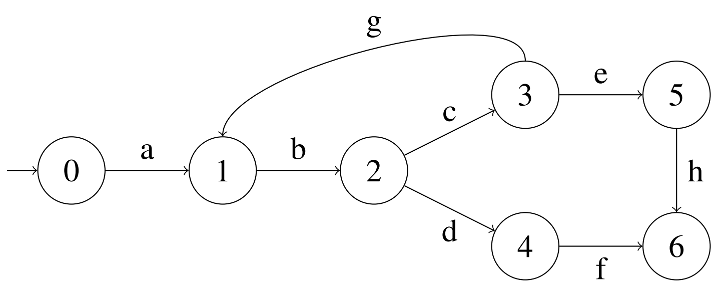
What do a model (labelled transition system) and its mini model (minimized labelled transition system) have in common? (Relevant definitions can be found at the end of this test.)
Answer:

1. The initial state.
2. The final states.
3. The branching structure.
4. The language: (finite and infinite) sequences of actions.

Marking scheme: 1 mark for each correct answer.

6 (2 marks)

Consider the following model (labelled transition system).



Draw the corresponding mini model (minimized labelled transition system). (Relevant definitions can be found at the end of this test.)
Answer:

$$\begin{tikzcd}
 & & & 3 \arrow[ld, bend right, "gb"'] \arrow[dd, "eh"] \\
{} \arrow[r] & 0 \arrow[r, "ab"] & 2 \arrow[ru, bend right, "c"'] \arrow[rd, "df"'] & \\
 & & & 6
\end{tikzcd}$$

Marking scheme: 1 mark for transitions labelled df and eh; 1 mark for the transitions labelled ab and gh.

7 (5 marks)

(a) What does a listener do?

Answer: it handles some of the events generated by JPF's virtual machine and search.

Marking scheme: 1 marks for mentioning events.

(b) List the two types of listeners of Java PathFinder.

Answer:

1. VMListener
2. SearchListener

Marking scheme: 1 marks for each correct answer.

(c) Given an example of each type of listener and describe what the listener does.

Answer:

1. Mnemonics prints the mnemonics of the instructions executed by JPF.
2. StateSpace produces a representation of the state space as a DOT file.

Marking scheme: 1 marks for each correct answer (there are many correct answers).

8 (2 marks)

(a) Why does Java PathFinder contain the class `ListenerAdapter`?

Answer: It provides a default implementation of all methods in SearchListener and VMListener and, hence, saves the implementer of a listener a lot of work.

Marking scheme: 1 mark for mentioning either default implementations or simplifying the implementation of listeners.

(b) Why is the class `ListenerAdapter` abstract?

Answer: So that it cannot be instantiated. Creating a listener that does nothing is pointless.

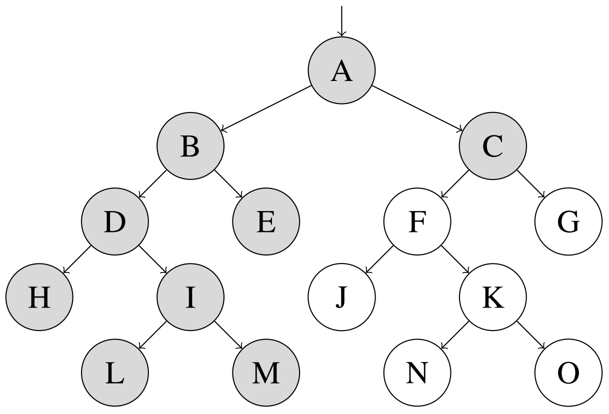
Marking scheme: 1 mark for observing that the class cannot be instantiated.

9 (2 marks)

Explain the difference between `classpath` and `native_classpath`?
Answer: The `classpath` consists of the folders and jar files containing the code that is modelled checked by JPF. The `native_classpath` consists of the folders and jar files containing the code that is executed as part of JPF.
Marking scheme: 1 mark for linking `classpath` with model checking and 1 mark for linking `native_classpath` with running JPF.

10 (6 marks)

The search strategy *beam search* only visits a beam of a particular width. For simplicity, we assume here that the width is 2. At each level, (at most) two states are visited. Consider the following state space.



For example, for the above state space, if started in the initial state A, the states are visited in the following order: A, B, C, D, E, H, I, L, M.
Recall the following methods.

- `public boolean forward()`: tries to move forward along an unexplored transition and returns whether the move is successful.
- `public boolean backtrack()`: tries to backtrack and returns whether the backtrack is successful.

- `private RestorableVMState getRestorableState()`: returns the current state so that it is restorable.
- `private void restoreState(RestorableVMState state)`: restores the given state.
- `public boolean isNew()`: tests whether the current state is new, that is, it has not been visited before.
- `public boolean isEnd()`: tests whether the current state an end (final) state.

(a) For the above state space, provide the sequence of the first 15 calls to the above methods and methods that manipulate appropriate data structures and the value returned by them corresponding to the beam search started in state A.

Answer: As data structures, I use two queues (current and next). The sequence is current.enqueue; current.dequeue; restoreState; forward(true); next.enqueue; backtrack(true); forward(true); next.enqueue; backtrack(true); current.dequeue; restoreState; forward(true); next.enqueue; backtrack(true); forward(true); next.enqueue; . . .

Note that the enqueue uses getRestorableState.

Marking scheme: 1 mark for using appropriate data structures (a queue, an array of size two, etc). 1 mark for the appropriate ordering of forward and backtrack calls. 1 mark restoring the state at the appropriate places.

(b) Give an algorithm (in terms of pseudocode or Java code) using some of the above methods that gives rise to the sequence of part (a) if started in state A.

Answer:

```
private void search(RestorableVMState state) {
  Queue<RestorableVMState> current = new LinkedList<RestorableVMState>()
  current.offer(state);
  while (!current.isEmpty()) {
    Queue<RestorableVMState> next = new LinkedList<RestorableVMState>();
    for (RestorableVMState source : current) {
      this.restoreState(source);
      while (this.forward() && next.size() < 2) {
        if (this.isNewState() && !this.isEndState()) {
          next.offer(this.getRestorableState());
        }
        this.backtrack();
      }
    }
    current = next;
  }
}
```

Marking scheme: 1 mark for the outer loop; 1 mark for the inner loop using forward and restricting the size of the beam; 1 mark for checking whether the state is a new state and not a final state.