EECS 4315 3.0 Mission Critical Systems

Solution to Final exam

9:00–11:00 on April 18, 2018

1 (3 marks)

What are the three E's of the approach to develop dependable software as described in the book "Software for Dependable Systems: Sufficient Evidence?"
Answer:

1. Explicit.
2. Evidence.
3. Expertise.

Marking scheme: 1 mark for each correct answer.

2 (3 marks)

(a) Describe the notion of a deadlock.
Answer: All threads are waiting on each other.
Marking scheme: 0.5 mark for anything similar to the above.

(b) To which language features of Java should you pay close attention when looking for a deadlock?
Answer: The `wait` method.
Marking scheme: 0.5 mark for mentioning `wait` or `notify` or `notifyAll`.

(c) Describe the notion of a race condition.
Answer: A race condition is a flaw that occurs when the timing or ordering of events affects a program's correctness.
Marking scheme: 0.5 mark for mentioning the timing or ordering of events.

(d) To which language features of Java should you pay close attention when looking for a race condition?
Answer: Threads.
Marking scheme: 0.5 mark for any reasonable answer.

(e) Describe the notion of a data race.
Answer: A data race happens when there are two memory accesses in a program where both

- target the same location,

– are performed concurrently by two threads,
– are not reads (at least one is a write),
– are not synchronization operations.

Marking scheme: 0.5 mark for mentioning concurrent accesses.

(f) To which language features of Java should you pay close attention when looking for a data race?
Answer: Shared attributes.
Marking scheme: 0.5 mark for anything reasonable.

3 (4 marks)

(a) Three threads share the attribute `a`. Initially, the value of `a` is 0. The first thread executes the following code.

```
a++;
a--;
```

The second thread executes the following code.

```
a = 1;
a = 2;
```

The third thread executes the following code.

```
a++;
a++;
```

What are the possible final values of `a`? Explain your answer.

Answer: 0, 1, 2, 3, 4, 5. Note that `a++` and `a--` are not atomic. Let me explain how one can get the five different final values. I will name the threads T1, T2 and T3.

– T1 reads the value of `a`, which is zero. T2 and T3 execute all their instructions. T1 increments and writes one to `a`. Finally, T1 executes `a--` resulting in the final value zero.
– T1 executes `a++`, next T3 executes its instructions, after which T2 executes its instructions. At this point the value of `a` is two. Finally, T1 executes `a--` resulting in a final value of one.
– T1 executes all its instructions, T3 executes all its instructions, and then T2 executes all its instructions. The final value is two in this case.
– T1 executes all its instructions, T3 executes `a++`, T2 executes all its instructions, and finally T3 executes `a++`. This results in a final value of three.

- T1 executes all its instructions, T2 executes all its instructions, and then T3 executes all its instructions. In this case the final value is four.
- T2 executes all its instructions, and T1 executes `a++`. T3 reads the value of `a`, which is three. T1 executes `a--`. T3 increments and writes four to `a`. Finally, T3 executes `a++`. This results in a final value of five.

Marking scheme: 1 mark for observing that threads interleave, 1 mark for observing that `a++` and `a--` are not atomic.

(b) There are k threads. Each thread executes n instructions. To how many different executions may this give rise? Explain your answer.

Answer:

$$\begin{aligned}
&\binom{kn}{n}\binom{(k-1)n}{n}\cdots\binom{2n}{n} \\
=\ & \frac{(kn)!}{n!((k-1)n)!}\frac{((k-1)n)!}{n!((k-2)n)!}\cdots\frac{(2n)!}{n!n!} \\
=\ & \frac{(kn)!}{(n!)^k} \\
=\ & \frac{(kn)(kn-1)\cdots(kn-n+1)}{n!}\cdots\frac{2n(2n-1)\cdot(n+1)}{n!} \\
\geq\ & \left(\frac{2n(2n-1)\cdot(n+1)}{n!}\right)^k \\
=\ & \left(\frac{2n(2n-1)\cdot(n+1)}{n(n-1)\cdots 2}\right)^k \\
\geq\ & n^k
\end{aligned}$$

Marking scheme: 2 marks for something similar to the above, 1 mark if some choose or factorial is part of the answer.

4 (2 marks)

Explain the difference between the `start` and `run` method by means of an example.
Answer: Consider the following code.

```
public class Printer extends Thread {
 public Printer(String name) {
  super(name);
 }

 public void run() {
  while (true) {
```

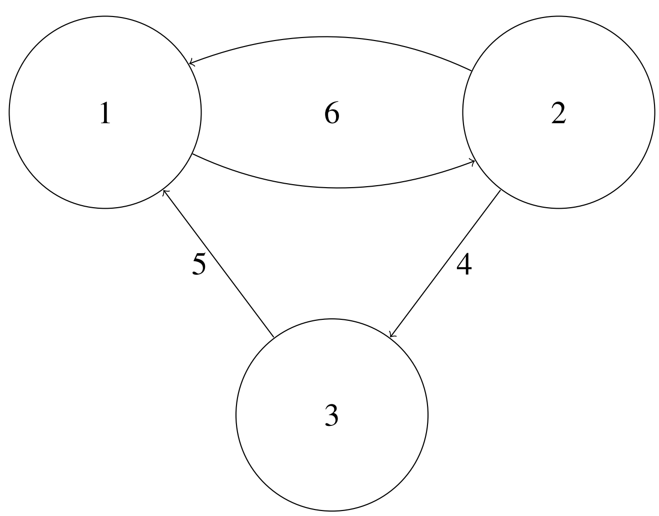
```
      Sustem.out.print(this.getName());
    }
  }
}

public class Main {
  public static void main(String[] args) {
    Printer one = new Printer("1");
    Printer two = new Printer("2");
    one.start() // one.run();
    two.start(); // two.run();
  }
}
```

Since the `start` method starts the new thread and invokes its `run` method, if the `start` method is used an interleaving of ones and twos is printed. If instead the `run` method is used, the new thread is not start and, hence, only ones are printed.
Marking scheme: 1 mark for a reasonable example, 1 mark for an explanation that mentions the starting of the new thread.

5 (3 marks)

Complete the following diagram that captures the states of a thread and the ways in which the state of a thread can be changed by providing the terms associated with 1, 2, 3, 4, 5 and 6 in the list below.



Answer:

1. Runnable.

2. Running.
3. Blocked.
4. `wait`.
5. `notify/notifyAll`.
6. Scheduler.

Marking scheme: 0.5 mark for each correct answer.

6 (1 mark)

Given all the (byte)code of a multi-threaded app, determine for a specific bytecode instruction of a specific thread whether it impacts other threads. Sketch how to solve this problem.
Answer: This problem cannot be solved.
Marking: 1 mark for a similar answer.

7 (4 marks)

A solution to the dining savages problem can be found at the end of this exam.

(a) Can the `while` in line 28 of the `Pot` class be replaced by an `if`? Explain your answer.

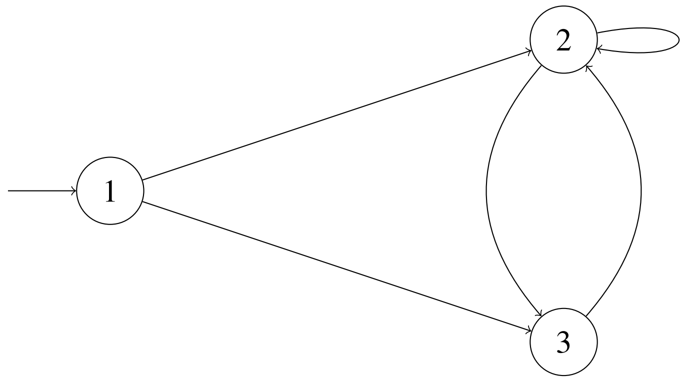
Answer: No. Ten savages could be waiting on line 28 because the pot is empty. All will be notified and become runnable once the cook has prepared the food. Once a savage becomes runnable it will take a portion if an `if` is used. However, there are only five portions.
Marking scheme: 1 mark for the observation that a savage becomes runnable and 1 mark for the observation that the savage can then take a portion.

(b) Can the `notifyAll` in line 36 of the `Pot` class be replaced by an `notify`? Explain your answer.

Answer: Yes. At this point, only the cook can be notified and there is only one.
Marking scheme: 2 marks for the observation that it cannot lead to a deadlock. 2 marks for the observation that it is always safe to use `notifyAll` instead of `notify` to avoid deadlocks.

8 (5 marks)

Consider the following transition system



and the following labelling function

$$\begin{array}{rcl} \ell(1) & = & \{a\} \\ \ell(2) & = & \{b\} \\ \ell(3) & = & \{c\} \end{array}$$

For each of the following LTL formulas, determine if that formula holds for the above transition system. A simple yes or no suffices.

Answer:

(a) a Yes.

(b) b No.

(c) $a \wedge \neg b$ Yes.

(d) $\bigcirc(b \vee c)$ Yes.

(e) $\bigcirc \bigcirc \neg a$ Yes.

(f) $\Diamond b$ Yes.

(g) $\Box(a \vee c)$ No.

(h) $a \text{ U } b$ No.

(i) $b \text{ U } a$ Yes.

(j) $a \text{ U } ((b \vee c) \text{ U } c)$ No.

For each of the following CTL formulas, determine if that formula holds for the above transition system. A simple yes or no suffices.

(k) $\exists\Diamond c$ Yes.

(l) $\forall\Diamond c$ No.

(m) $\exists\Diamond b$ Yes.

(n) $\forall \Diamond b$ Yes.

(o) $\exists \Box \neg a$ No.

(p) $\forall \Box \neg a$ No.

(q) $\exists a$ U b Yes.

(r) $\forall a$ U b No.

(s) $\exists a$ U $(\exists b$ U $c)$ Yes.

(t) $\forall a$ U $(\forall b$ U $c)$ No.

Marking scheme: 0.25 mark for each correct answer.

9 (4 marks)

(a) Describe the notion of a safety property.

Answer: "nothing bad ever happens"
Marking scheme: 1 mark for something similar to the above.

(b) Give an example of a safety property expressed in LTL.

Answer: $\Box a$
Marking scheme: 1 mark for a correct answer.

(c) Describe the notion of a liveness property.

Answer: "something good eventually happens"
Marking scheme: 1 mark for something similar to the above.

(d) Give an example of a liveness property expressed in LTL.

Answer: $\Diamond a$
Marking scheme: 1 mark for a correct answer.

10 (3 marks)

(a) The atomic propositions m, r and a represent

- a request is made,
- a request is registered, and
- a request is answered,

respectively. Express "Once a request is made, it will remain registered at least until the request is answered" in LTL.

Answer: $\Box(m \to r \text{ U } a)$
Marking scheme: 0.5 mark for $\Box$, 0.5 mark for r U a and 0.5 mark for $m \to$.

(b) The atomic proposition e_i and f_i represent

- philosopher i is eating and
- philosopher i has just finished eating,

respectively. Express "Whenever philosopher 4 has finished eating, he cannot eat again until philosopher 3 has eaten" in CTL.

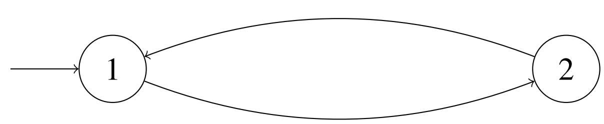
Answer: $\forall\Box(f_4 \to \neg\exists(\neg e_4 \wedge \neg f_3) \text{ U } (e_4 \wedge \neg f_3))$
Marking scheme: 0.5 mark for $\forall$, 0.5 mark for U and 0.5 mark for $\neg e_4$.

11 (4 marks)

Let f and g be arbitrary LTL formulas. Which of the following equivalences hold? If the equivalence holds, give a proof. If the equivalence does not hold, provide a transition system (and a specific choice for f and g) for which one of the two LTL formulas holds and the other LTL formula does not hold.

(a) $\Box\Diamond f \equiv \Diamond\Box f$

Answer: The equivalence does not hold. Let $f = a$. Consider the following transition system



and the following labelling function

$$\begin{aligned} \ell(1) &= \{a\} \\ \ell(2) &= \emptyset \end{aligned}$$

In this transition system, $\Box\Diamond a$ holds, but $\Diamond\Box a$ does not hold.
Marking scheme: 1 mark for a correct counterexample.

(b) $\neg(f \text{ U } g) \equiv (\neg g \text{ U } (\neg f \wedge \neg g)) \vee \Box\neg g$

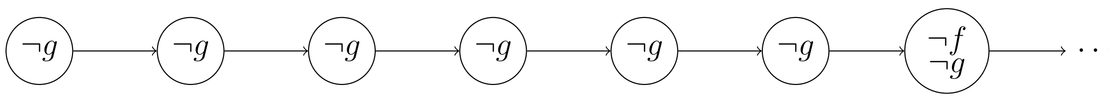
Answer: Note that

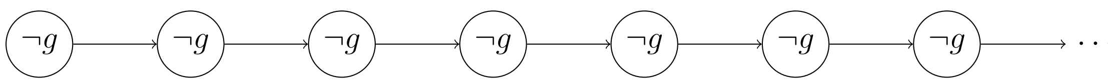
$$\begin{aligned}
&p \models \neg(f \,\mathrm{U}\, g) \\
&\quad \text{iff} \quad \text{not}(p \models f \,\mathrm{U}\, g) \\
&\quad \text{iff} \quad \text{not}(\exists i \geq 0 : p[i..] \models g \text{ and } \forall 0 \leq j < i : p[j..] \models f) \\
&\quad \text{iff} \quad \forall i \geq 0 : \text{not}(p[i..] \models g) \text{ or } \exists 0 \leq j < i : \text{not}(p[j..] \models f) \\
&\quad \text{iff} \quad \forall i \geq 0 : p[i..] \models \neg g \text{ or } \exists 0 \leq j < i : p[j..] \models \neg f
\end{aligned} \tag{1}$$

and

$$\begin{aligned}
&p \models (\neg g \,\mathrm{U}\, (\neg f \wedge \neg g)) \vee \Box\neg g \\
&\quad \text{iff} \quad (\exists m \geq 0 : p[m..] \models \neg f \wedge \neg g \text{ and } \forall 0 \leq n < m : p[n..] \models \neg g) \text{ or } \forall k \geq 0 : p[k..] \models \neg g \\
&\quad \text{iff} \quad (\exists m \geq 0 : p[m..] \models \neg f \text{ and } p[m..] \models \neg g) \text{ and } \forall 0 \leq n < m : p[n..] \models \neg g) \text{ or} \\
&\qquad\quad \forall k \geq 0 : p[k..] \models \neg g
\end{aligned} \tag{2}$$

It remains to show that (1) and (2) are equivalent. In both cases, executions satisfying the formula have the following shape.





Marking scheme: 1 mark for an (almost) correct proof.

Let f and g be arbitrary CTL formulas. Which of the following equivalences hold? If the equivalence holds, give a proof. If the equivalence does not hold, provide a transition system (and a specific choice for f and g) for which one of the two CTL formulas holds and the other CTL formula does not hold.

(c) $\forall \bigcirc f \equiv \neg\exists \bigcirc \neg f$

Answer:

$$\begin{aligned}
s \models \forall \bigcirc f \quad &\text{iff} \quad \forall p \in \mathit{Paths}(s) : p \models \bigcirc f \\
&\text{iff} \quad \forall p \in \mathit{Paths}(s) : p[1] \models f \\
&\text{iff} \quad \text{not}(\exists p \in \mathit{Paths}(s) : \text{not}(p[1] \models f)) \\
&\text{iff} \quad \text{not}(\exists p \in \mathit{Paths}(s) : p[1] \models \neg f) \\
&\text{iff} \quad \text{not}(\exists p \in \mathit{Paths}(s) : p \models \bigcirc \neg f) \\
&\text{iff} \quad \text{not}(s \models \exists \bigcirc \neg f) \\
&\text{iff} \quad s \models \neg\exists \bigcirc \neg f
\end{aligned}$$

Marking scheme: 1 mark for an (almost) correct proof.

(d) $\forall\Box f \equiv \forall \bigcirc \forall\Box f$.

Answer: The equivalence does not hold. Let $f = a$. Consider the following transition system

and the following labelling function

$$\begin{array}{rcl} \ell(1) & = & \emptyset \\ \ell(2) & = & \{a\} \end{array}$$

In this transition system, $\forall \bigcirc \forall\Box a$ holds, but $\forall\Box a$ does not hold.
Marking scheme: 1 mark for a correct counterexample.

12 (2 marks)

Recall that

$$Sat(f) = \{ s \in S \mid s \models f \}.$$

(a) Prove that $Sat(f \wedge g) = Sat(f) \cap Sat(g)$.

Answer:

$$\begin{array}{rcl} Sat(f \wedge g) & = & \{ s \in S \mid s \models f \wedge g \} \\ & = & \{ s \in S \mid s \models f \text{ and } s \models g \} \\ & = & \{ s \in S \mid s \models f \} \cap \{ s \in S \mid s \models g \} \\ & = & Sat(f) \cap Sat(g) \end{array}$$

Marking scheme: 1 mark for an (almost) correct proof.

(b) Explain how computing Sat can be used to solve the CTL model checking problem.

Answer: A transition system $\langle S, L, I, \rightarrow, \ell \rangle$ satisfies CTL formula f iff $I \subseteq Sat(f)$. The set $Sat(f)$ can be computed recursively on the syntactic structure of f.
Marking scheme: 1 mark for the mentioning $I \subseteq Sat(f)$.

Pot class

```
public class Pot {
  private int size;
  private int content;

  public Pot(int size) {
    this.size = size;
    this.content = 0;
  }

  private synchronized void beginCook() throws InterruptedException {
    while (this.content > 0) {
      this.wait();
    }
  }

  private synchronized void endCook() {
    this.content = this.size;
    this.notifyAll();
  }

  public void cook() throws InterruptedException {
    this.beginCook();
    // cook
    this.endCook();
  }

  public synchronized void take() {
    while (this.content == 0) {
      try {
        this.wait();
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
    this.content--;
    this.notifyAll();
  }
}
```

Cook class

```
public class Cook extends Thread {
  private Pot pot;

  public Cook(Pot pot) {
    super();
    this.pot = pot;
  }

  public void run() {
    try {
      while (true) {
        this.pot.cook();
      }
    } catch (InterruptedException e) {
      // do nothing
    }
  }
}
```

Savage class

```
public class Savage extends Thread {
  private Pot pot;

  public Savage(Pot pot) {
    super();
    this.pot = pot;
  }

  public void run() {
    this.pot.take();
  }
}
```

DiningSavages class

```
public class DiningSavages {
  public static void main(String[] args) {
    final int SIZE = 4;
    final int SAVAGES = 10;
```

```

  Pot pot = new Pot(SIZE);
  Cook cook = new Cook(pot);
  Savage[] savage = new Savage[SAVAGES];
  for (int i = 0; i < SAVAGES; i++) {
   savage[i] = new Savage(pot);
  }
  cook.start();
  for (int i = 0; i < SAVAGES; i++) {
   savage[i].start();
  }
  for (int i = 0; i < SAVAGES; i++) {
   try {
    savage[i].join();
   } catch (InterruptedException e) {
    e.printStackTrace();
   }
  }
  cook.interrupt();
  try {
   cook.join();
  } catch (InterruptedException e) {
   e.printStackTrace();
  }
 }
}
```

Definitions

Definition 1. A transition system is a tuple $\langle S, L, I, \rightarrow, \ell \rangle$ consisting of

- a set S of states,
- a set L of labels,
- a set $I \subseteq S$ of initial states,
- a transition relation $\rightarrow \subseteq S \times S$, and
- a labelling function $\ell : S \rightarrow 2^L$.

Definition 2. Linear temporal logic (LTL) is defined by the grammar

$$f ::= a \mid f \wedge f \mid \neg f \mid \bigcirc f \mid f \,\mathrm{U}\, f$$

where $a \in L$.

We use the following syntactic sugar.

$$\begin{array}{rcll} f \vee g & = & \neg(\neg f \wedge \neg g) & \\ \text{true} & = & a \vee \neg a & \\ \Diamond f & = & \text{true} \, \mathrm{U} \, f & \text{(eventually } f\text{)} \\ \Box f & = & \neg \Diamond \neg f & \text{(always } f\text{)} \end{array}$$

Definition 3. $\mathit{Paths}(s)$ is the set of (execution) paths starting in state s. Let $p \in \mathit{Paths}(s)$ and $n \geq 0$. Then $p[n]$ is the $(n+1)^{\text{th}}$ state of the path p and $p[n..]$ is the suffix of p starting with the $(n+1)^{\text{th}}$ state.

Definition 4. The relation $\models$ is defined by

$$\begin{array}{rcl} p \models a & \text{iff} & a \in \ell(p[0]) \\ p \models f \wedge g & \text{iff} & p \models f \text{ and } p \models g \\ p \models \neg f & \text{iff} & \text{not}(p \models f) \\ p \models \bigcirc f & \text{iff} & p[1..] \models f \\ p \models f \, \mathrm{U} \, g & \text{iff} & \exists i \geq 0 : p[i..] \models g \text{ and } \forall 0 \leq j < i : p[j..] \models f \end{array}$$

and

$$\langle S, L, I, \rightarrow, \ell \rangle \models f \text{ iff } \forall s \in I : \forall p \in \mathit{Paths}(s) : p \models f$$

Definition 5. Computation tree logic (CTL) is defined as follows. The state formulas are defined by

$$f ::= a \mid f \wedge f \mid \neg f \mid \exists g \mid \forall g$$

where $a \in L$. The path formulas are defined by

$$g ::= \bigcirc f \mid f \, \mathrm{U} \, f$$

Definition 6. The relation $\models$ is defined by

$$\begin{array}{rcl} s \models a & \text{iff} & a \in \ell(s) \\ s \models f \wedge g & \text{iff} & s \models f \text{ and } s \models g \\ s \models \neg f & \text{iff} & \text{not}(s \models f) \\ s \models \exists g & \text{iff} & \exists p \in \mathit{Paths}(s) : p \models g \\ s \models \forall g & \text{iff} & \forall p \in \mathit{Paths}(s) : p \models g \end{array}$$

and

$$\begin{array}{rcl} p \models \bigcirc f & \text{iff} & p[1] \models f \\ p \models f \, \mathrm{U} \, g & \text{iff} & \exists i \geq 0 : p[i] \models g \text{ and } \forall 0 \leq j < i : p[j] \models f \end{array}$$

and

$$\langle S, L, I, \rightarrow, \ell \rangle \models f \text{ iff } \forall s \in I : s \models f$$

Definition 7. The satisfaction set $\mathit{Sat}(f)$ is defined by

$$\mathit{Sat}(f) = \{ \, s \in S \mid s \models f \, \}.$$

Definition 8. LTL/CTL formulas f and g are equivalent, denoted $f \equiv g$, if $\langle S, L, I, \rightarrow, \ell \rangle \models f$ iff $\langle S, L, I, \rightarrow, \ell \rangle \models g$ for all transition systems $\langle S, L, I, \rightarrow, \ell \rangle$.