
EXAM
System Validation
(192140122)
13:45 - 17:15
02-07-2012

- The exercises are worth a total of 100 points.
The final grade for the course is $\frac{(hw_1+hw_2)/2+exam/10}{2}$, provided that you obtain at least 50 points for the exam (otherwise, the final grade for the course is a 4).
 - The exam is open book: all *paper* copies of slides, papers, notes etc. are allowed.
-

Exercise 1: Formal Tools and Techniques (10 points)

Suppose you are asked to be a consultant for a team developing a flight control system for a space craft. The team has 1 engineer who is well-trained in the use of formal methods. This person has practical experience with many different formal techniques. The other team members are well-skilled programmers, with a good understanding of the application domain, but without any knowledge of formal methods.

A prototype for the flight control system exists, but has not been tested yet, and it is likely to contain bugs. The deadline for the completion of the flight control system is in a month from now (and no extensions are possible). The flight control system has to fulfill strong requirements. In particular, the space craft should *always* remain controllable, so that the astronauts can get back to Earth.

Give a short description (no more than 200-250 words) of your advice to this team.

Answer

Of course, there is not a single possible solution here. Important ingredients are in my opinion:

- Time is too short to establish full correctness of whole application.
- Identify the small components whose correctness is really essential
- Verify functional correctness of these components individually (I would use static checking here)
- Well-trained programmers should be able to write JML specs for non-critical components, and validate those with run-time checker.
- Abstract model to describe interactions between components, and use model checking to verify this.
- Check for race conditions (using JPF).

Exercise 2: Specification (15 points)

Write formal specifications for the following informal requirements in an appropriate specification formalism of your choice (3 points per item). You may assume that appropriate atomic propositions, query methods and classes exist.

- a You can only open a file when it is closed.
- b All doors of a plane must be closed when the plane is flying.
- c If you roll a dice, the outcome is always between 1 and 6.
- d You are only allowed to live in a country as long as your passport is valid.
- e For a calendar system : the month always follow the sequence January \rightarrow February \rightarrow March ... December \rightarrow January \rightarrow February ...
NB There is no need to spell out all 12 cases, as long as the general idea of your specification is clear.

Answers

Many different formalisations can exist. Also in several cases, both temporal logic and a JML invariant would be an appropriate choice. (Thus answers that differ from the answers below might still be correct.)

- a Declare ghost variable `STATUS` with possible values `OPEN` and `CLOSE`, which is updated inside the methods `open()` and `close()`. Specification for method `open()`:
`requires STATUS == CLOSE;`
- b $G(\textit{flying} \Rightarrow \textit{doorsClosed})$
- c Postcondition of a method `rollDice()`:
`ensures 1 <= \result && \result <= 6;`
or invariant of a class `Dice`:
`invariant 1 <= value && value <= 6;`
- d $\textit{passportValid} \Rightarrow (\textit{allowed} \textit{W} \neg \textit{passportValid})$
- e `constraint \old(month) == January ==> month == January || month == February;`
`constraint \old(month) == February ==> month == February || month == March;`
... `constraint \old(month) == December ==> month == December || month == January;`

Exercise 3: Abstraction (8 points)

Consider interface `GameCharacter` and two classes implementing this interface: `Giant` and `GrowingDwarf` (on the next page).

The interface `GameCharacter` is intended to model a character that can move along a horizontal axis. Initially it starts at position 0, and then it can move left or right. (NB, it is thus okay that position becomes negative).

Different game character implementations can make steps of different sizes, i.e., a giant always makes steps of 10 units, while the step size of the growing dwarf increases with every step he makes.

Write an abstract specification for `GameCharacter`, modelling the intended behaviour of `stepLeft` and `stepRight`. Then add sufficient specifications to `Giant` and `GrowingDwarf` in order to show that these implementations respect your specification of `GameCharacter`.

```

1 package abstraction;
2
3 public interface GameCharacter {
4
5     public void stepLeft();
6
7     public void stepRight();
8 }

```

```

1 package abstraction;
2
3 public class Giant implements GameCharacter {
4
5     private int pos = 0;
6
7     public void stepLeft(){
8         pos = pos - 10;
9     }
10
11    public void stepRight(){
12        pos = pos + 10;
13    }
14 }

```

```

1 package abstraction;
2
3 public class GrowingDwarf implements GameCharacter {
4
5     private int pos = 0;
6     private int steps_taken = 0;
7
8     public void stepLeft() {
9         pos = pos - steps_taken;
10        steps_taken++;
11    }
12
13    public void stepRight() {
14        pos = pos + steps_taken;
15        steps_taken++;
16    }
17 }

```

Answer

Division of points: 2 for correct declaration of model variables, 2 for specifications of `stepLeft()` and `stepRight()`, and 2 for each correct represents clause.

```

1 package abstractionSpecs;
2
3 public interface GameCharacter {
4
5     //@ public model instance int _stepSize;
6     //@ public model instance int _pos;
7
8     //@ ensures _pos == \old(_pos) - \old(_stepSize);
9     public void stepLeft();
10
11     //@ ensures _pos == \old(_pos) + \old(_stepSize);
12     public void stepRight();
13 }

```

```

1 package abstractionSpecs;
2
3 public class Giant implements GameCharacter {
4
5     //@ represents _pos = pos;
6     //@ represents _stepSize = 10;
7
8     /*@ spec_public */ private int pos = 0;
9
10    public void stepLeft(){
11        pos = pos - 10;
12    }
13
14    public void stepRight(){
15        pos = pos + 10;
16    }
17 }

```

```

1 package abstractionSpecs;
2
3 public class GrowingDwarf implements GameCharacter {
4
5     //@ represents _pos = pos;
6     //@ represents _stepSize = steps_taken;
7
8     private int pos = 0;
9     private int steps_taken = 0;
10
11    public void stepLeft() {
12        pos = pos - steps_taken;
13        steps_taken++;
14    }
15
16    public void stepRight() {
17        pos = pos + steps_taken;
18        steps_taken++;

```

```
19     }  
20 }
```

No points subtracted if specification of `stepLeft()` and `stepRight()` do not use old value of `stepSize` (this is probably too subtle to notice without tool support).

Exercise 4: Run-time Checking (15 points)

Each subquestion is worth 3 points. Explain your answers (without explanation no points will be awarded).

- a Consider class `Computation`. What will happen when the JML run-time checker is used to validate this class?

```
1 package rc;
2
3 public class Computation {
4
5     /*@ spec_public */ private int val;
6
7     /*@ ensures val == \old(val) * \old(val);
8     */
9     public void compute() {
10         val = 2 * val;
11     }
12
13     public static void main (String [] args) {
14         Computation c = new Computation();
15         c.compute();
16     }
17 }
```

b Consider class Screen. What will happen when the JML run-time checker is used to validate this class?

```
1 package rc;
2
3 public class Screen {
4
5     /*@ spec_public */ int x_size;
6     /*@ spec_public */ int y_size;
7
8     //@ invariant (x_size % 2) == 0;
9     //@ invariant (y_size % 2) == 0;
10
11     public void updateScreen() {
12     }
13
14     public void updateHorizontal() {
15         x_size++;
16         updateScreen();
17         x_size++;
18         updateScreen();
19     }
20
21     public static void main (String [] args) {
22         Screen s = new Screen();
23         s.updateScreen();
24         s.updateHorizontal();
25     }
26 }
```


c Consider class `Game`. What will happen when the JML run-time checker is used to validate this class?

```
1 package rc;
2
3 class Player {}
4
5 public class Game {
6
7     /*@ spec_public */ private Player player1 = new Player();
8     /*@ spec_public */ private Player player2 = new Player();
9
10    //@ invariant player1 != player2;
11
12    public void battle(){
13        warmUp(player1);
14        warmUp(player2);
15        fight();
16    }
17
18    /*@ requires player1 != player2;
19       ensures false;
20       signals (Exception) false;
21    */
22    public void fight() {
23    }
24
25    public void warmUp(Player p){
26    }
27
28    public static void main (String [] args) {
29        Game g = new Game();
30        g.battle();
31    }
32 }
```

d Consider class `Student`. What will happen when the JML run-time checker is used to validate this class?

```
1 package rc;
2 import java.util.HashMap;
3 import java.util.Map;
4
5 public class Student {
6
7     public static final int numCourses = 35;
8     /*@ spec_public */ private int[] grades = new int[numCourses];
9
10    /*@ invariant grades != null;
11    /*@ invariant (\forall int i; 0 <= i && i < numCourses;
12                0 <= grades[i] && grades[i] <= 10);
13    */
14
15    /*@ requires 0 <= grade && grade <= 10;
16        requires 0 <= course && course < numCourses;
17        ensures grades[course] == grade;
18    */
19    public void newResult(int course, int grade) {
20        grades[course] = grade;
21    }
22
23    public static void main (String [] args) {
24        Student s = new Student();
25        s.newResult(22, 86);
26    }
27 }
```

e Consider class Resources. What will happen when the JML run-time checker is used to validate this class?

```
1 package rc;
2
3 class Thing {
4     //@ public static ghost int countThings= 0;
5
6     //@ ensures countThings == \old(countThings) + 1;
7     public Thing() {
8         //@ set countThings = countThings + 1;
9     }
10
11 }
12
13 public class Resources {
14
15     public static final int MAX = 26;
16
17     //@ invariant Thing.countThings <= MAX;
18
19     /*@ requires p > 0;
20         ensures Thing.countThings == \old(Thing.countThings) + p;
21     */
22     public void makeThings(int p) {
23         Thing[] storage = new Thing[p];
24         //@ loop_invariant 0 <= i && i <= storage.length;
25         /*@ loop_invariant Thing.countThings ==
26             \old(Thing.countThings) + i;
27         */
28         for (int i = 0; i < storage.length; i++) {
29             new Thing();
30         }
31     }
32
33     public static void main (String [] args) {
34         Resources r= new Resources();
35         r.makeThings(13);
36     }
37 }
```

Answers

- a Nothing will happen, because this particular execution (where `val` is 0) will not violate the specification – even though the method `compute` clearly does not respect its specification.
- b An invariant violation will be reported, because between the call to `updateHorizontal` and `updateScreen`, `x_size` has been increased by one. The (first) call to `updateScreen` thus is a visible state, and at this point the invariant that `x_size` is always even is violated.
- c It will signal that the normal postcondition of `fight()`, which states that `fight()` never should terminate normally is violated – because `fight()` does terminate.
- d It will signal a precondition error, because `newResult` is called with values that are out of bounds for the precondition.
- e Nothing will happen, because this particular execution does not create more than `MAX` things.

Exercise 5: Static Checking (15 points)

Subquestions a, b, c are worth 3 points, subquestion d is worth 6 points. Explain your answers (without explanation no points will be awarded).

- a Consider again class `Resources` from exercise 4e. What will happen when ESC/Java is used to validate this class.

b Consider class Channel. Add a loop invariant.

```
1 package sc;
2
3 class InputException extends Exception {}
4 class NotEnoughInputException extends Exception {}
5
6 class Token{}
7
8 public class Channel {
9
10     /*@ spec_public */ private Token [] channel = new Token[4];
11     /*@ spec_public */ private int [] processed = new int[4];
12
13     /*@ invariant channel != null;
14     /*@ invariant processed != null;
15
16     /*@ pure */ public boolean illegalToken(Token s) {
17         return false;
18     }
19
20     /*@ pure */ public int process (Token s) {
21         return 0;
22     }
23
24     /*@ requires n > 0;
25         ensures (\forall int i; 0 <= i && i < n;
26                 processed[i] == process(channel[i]));
27         signals (NotEnoughInputException) false;
28         signals (InputException) false;
29     */
30     public void readTokens(int n) throws NotEnoughInputException ,
31         InputException {
32         if (n > channel.length) {
33             throw new NotEnoughInputException();
34         }
35         processed = new int [n];
36         for (int i = 0; i < processed.length; i++) {
37             if (illegalToken(channel[i])) {
38                 throw new InputException();
39             }
40             else processed[i] = process(channel[i]);
41         }
42     }
43 }
```

- c Assuming that your loop invariant is correct, what will happen when ESC/Java is applied on Channel?
- d Consider class GenderBalance. When using ESC/Java to validate this class, two different errors will be signalled. What are these errors, and what causes them (3 points per error).

```
1 package sc;
2
3 public class GenderBalance {
4
5     /*@ spec_public */ private int nr_of_men;
6     /*@ spec_public */ private int nr_of_women;
7
8     //@ invariant nr_of_men <= nr_of_women;
9
10    public int addMen(int m) {
11        return nr_of_men + m;
12    }
13
14    public int addWomen(int w) {
15        return nr_of_women + w;
16    }
17
18    /*@ requires m <= w;
19        ensures nr_of_men == \old(nr_of_men) + m;
20        ensures nr_of_women == \old(nr_of_women) + m;
21    */
22    public void recruitment(int m, int w) {
23        nr_of_men = addMen(m);
24        nr_of_women = addWomen(w);
25    }
26 }
```

Answers

a ESC/Java will signal that the invariant that countThings should always be below MAX might be violated (as there is no bound on how many things can be added).

```
b //@ loop_invariant processed != null && channel != null;
   //@ loop_invariant 0 <= i && i <= processed.length;
   //@ loop_invariant processed.length <= channel.length;
   //@ loop_invariant (\forall int j; 0 <= j && j < i; processed[j] ==
   process(channel[j]));
```

c The exceptional postconditions might not be established.

d The postcondition of recruitment might not be established, because the methods addMen and addWomen do not have specifications.

The object invariant might be broken when addWomen is called from the method recruitment, because men are added before women are added.

Exercise 6: Modeling (22 points)

Consider the Bean Can Game.

You and a friend are given a can containing N black beans and M white beans initially. The can is emptied according to the following repeated process:

- Select two beans from the can
- If the beans are:
 - the same color: put a black bean back in the can
 - different colors: put a white bean back in the can

The player who chooses the color of the remaining bean wins the game.

-- Bean Can Algorithm pseudocode

```
while (num-beans-in-can > 1) do {
  pick 2 beans randomly
  if bean1-color == bean2-color
  then put-back black bean
  else put-back white bean
}
```

- (12 pts.) Write a NuSMV model for this game.
- (3 pts.) Write a property to specify that the game always terminates.
- (3 pts.) Write a property to specify that black always is able to win the game.
- (4 pts.) Suppose you want to be sure that the model has no executions where it always picks only black beans. What change would you have to make to the model to ensure this.

Answer

```
1 MODULE main
2 VAR
3
4   M : 4 .. 4;
5   N : 13 .. 13;
6   black : 0 .. 2 * (M + N + 1);
7   white : 0 .. 2 * (M + N + 1);
8
9   bean1 : {B, W};
10  bean2 : {B, W};
11
12
13 ASSIGN
14   init(black) := M;
15   init(white) := N;
16
17
```

```

18 next(bean1) := case
19     black + white = 1 : {B, W};
20     black = 0 : W;
21     white = 0 : B;
22     TRUE : {B, W};
23     esac;
24 next(bean2) := case
25     black + white = 1 : {B, W};
26     black = 0 : W;
27     white = 0 : B;
28     black = 1 & next(bean1) = B : W;
29     white = 1 & next(bean1) = W : B;
30     TRUE : {B, W};
31     esac;
32
33 next(black) := case
34     black + white > 1 & black + white <= 2 * (M + N):
35         case bean1 = B & bean2 = B : black - 1 > 0 ? black -
36             bean1 = W & bean2 = W : black + 1;
37             TRUE : black - 1 > 0 ? black - 1 : 0;
38         esac;
39     TRUE : black;
40     esac;
41
42 next(white) := case
43     black + white > 1 & black + white <= 2 * (M + N):
44         case bean1 = W & bean2 = W : white - 2 > 0 ? white -
45             TRUE : white;
46         esac;
47     TRUE : white;
48     esac;
49
50 FAIRNESS bean1 = W | bean2 = W;
51
52 CTLSPEC AF (black + white = 1);
53
54 CTLSPEC EF ((black + white = 1) & black = 1);
55 CTLSPEC EF ((black + white = 1) & white = 1);

```

Don't subtract points related with the range of the variables `black` and `white`.

b See above.

c See above.

d Add a fairness constraint:

`bean1 = W | bean2 = W`; This ensures that only executions are considered where at least one of the beans at some points is white. In addition, one has to make sure that when the game is finished, always arbitrary beans are chosen.

Exercise 7: Traces (15 points)

Consider the following Java program:

```
1 import gov.nasa.jpj.jvm.Verify;
2 public class Workers {
3     public static Barrier barrier;
4     public static Object lock1=new Object();
5     public static Object lock2=new Object();
6
7     public static void main(String args[]) throws Exception {
8         barrier=new Barrier(2);
9         Thread t1 = new Thread(new Runner());
10        Thread t2 = new Thread(new Runner());
11        Thread t3 = new Thread(new Runner());
12        Verify.beginAtomic();
13        t1.start();
14        t2.start();
15        t3.start();
16        Verify.endAtomic();
17        t1.join();
18        t2.join();
19        t3.join();
20    }
21
22    public static class Runner implements Runnable {
23        public void run(){
24            synchronized(lock1){
25                synchronized(lock2){
26                    // do some work.
27                }
28            }
29            barrier.sync();
30            synchronized(lock2){
31                synchronized(lock1){
32                    // do more work.
33                }
34            }
35        }
36    }
37 }
```

The code for the barrier used is:

```
1 /**
2  A barrier is a way of synchronizing N threads.
3  It makes a group of threads wait until the last one
4  has reached the barrier (called sync).
5  */
6
7  public class Barrier {
8
9      private int threads;
10     private int blocked;
11     private boolean round;
12
13     public Barrier(int threads){
14         this.threads = threads;
15         blocked = 0;
16     }
17
18     public synchronized void sync(){
19         blocked++;
20         if (blocked < threads){
21             boolean expect = !round;
22             while (round != expect) {
23                 try { wait(); } catch (InterruptedException e) {}
24             }
25         } else {
26             round = !round;
27             blocked = 0;
28             notifyAll();
29         }
30     }
31 }
32 }
```

To analyse the presence of deadlocks for this program, we have used JPF. The input for JPF is:

```
1 # class to verify
2 target = Workers
3
4 # where to find byte code and source
5 classpath=.
6 sourcepath=.
7
8 # set heuristic
9 search.class = .search.heuristic.BFSHeuristic
10 search.multiple_errors = true
11
12 # how do we want the result?
13 report.console.property_violation=error , snapshot , trace
```

```
14
15 # what are we looking for?
16 listener=.listener.DeadlockAnalyzer
17
18 # how to display deadlocks
19 deadlock.format=essential
```

JPF signals that there is a problem with this code. Analyse the counter example (starting on the next page) and explain why the assertion was triggered.

When you describe the scenario, it is important to state which thread is running and precisely what the first and/or last action of the running thread during its turn is. (for example: "thread 37 starts by reading 7 from variable x and stops just before reading variable y.") The description of the other things a thread does during its turn can and *should* be much less detailed.

```

1 ===== error #71
2 gov.nasa.jpf.jvm.NotDeadlockedProperty
3 deadlock encountered:
4   thread java.lang.Thread:{ id:0,name:main,status:WAITING,priority:5,lockCount:0,
      suspendCount:0}
5   thread java.lang.Thread:{ id:1,name:Thread-1,status:WAITING,priority:5,lockCount:1,
      suspendCount:0}
6   thread java.lang.Thread:{ id:2,name:Thread-2,status:TERMINATED,priority:5,lockCount:0,
      suspendCount:0}
7   thread java.lang.Thread:{ id:3,name:Thread-3,status:TERMINATED,priority:5,lockCount:0,
      suspendCount:0}
8
9
10 ===== snapshot #71
11 thread java.lang.Thread:{ id:0,name:main,status:WAITING,priority:5,lockCount:0,
      suspendCount:0}
12   waiting on: java.lang.Thread@143
13   call stack:
14     at java.lang.Thread.join(Thread.java:-1)
15     at Workers.main(Workers.java:17)
16
17 thread java.lang.Thread:{ id:1,name:Thread-1,status:WAITING,priority:5,lockCount:1,
      suspendCount:0}
18   waiting on: Barrier@142
19   call stack:
20     at java.lang.Object.wait(Object.java:-1)
21     at Barrier.sync(Barrier.java:23)
22     at Workers$Runner.run(Workers.java:28)
23
24
25 ===== trace #71
26 transition #0 thread: 0
27 gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {id:"<root>" ,1/1,isCascaded:false}
28   [2895 insn w/o sources]
29   Workers.java:4      : public static Object lock1=new Object();
30   [1 insn w/o sources]
31   Workers.java:4      : public static Object lock1=new Object();
32   Workers.java:5      : public static Object lock2=new Object();
33   [1 insn w/o sources]
34   Workers.java:5      : public static Object lock2=new Object();
35   [2 insn w/o sources]
36   Workers.java:8      : barrier=new Barrier(2);
37   Barrier.java:13     : public Barrier(int threads){
38   [1 insn w/o sources]
39   Barrier.java:14     : this.threads=threads;
40   Barrier.java:15     : blocked=0;
41   Barrier.java:16     : }
42   Workers.java:8      : barrier=new Barrier(2);
43   Workers.java:9      : Thread t1=new Thread(new Runner());
44   Workers.java:21     : public static class Runner implements Runnable {
45   [1 insn w/o sources]
46   Workers.java:21     : public static class Runner implements Runnable {
47   Workers.java:9      : Thread t1=new Thread(new Runner());
48   [188 insn w/o sources]
49   Workers.java:9      : Thread t1=new Thread(new Runner());
50   Workers.java:10     : Thread t2=new Thread(new Runner());
51   Workers.java:21     : public static class Runner implements Runnable {
52   [1 insn w/o sources]
53   Workers.java:21     : public static class Runner implements Runnable {
54   Workers.java:10     : Thread t2=new Thread(new Runner());
55   [141 insn w/o sources]
56   Workers.java:10     : Thread t2=new Thread(new Runner());
57   Workers.java:11     : Thread t3=new Thread(new Runner());
58   Workers.java:21     : public static class Runner implements Runnable {
59   [1 insn w/o sources]
60   Workers.java:21     : public static class Runner implements Runnable {
61   Workers.java:11     : Thread t3=new Thread(new Runner());
62   [141 insn w/o sources]
63   Workers.java:11     : Thread t3=new Thread(new Runner());
64   Workers.java:12     : Verify.beginAtomic();

```

```

65     [2 insn w/o sources]
66 Workers.java:13           : t1.start();
67     [2 insn w/o sources]
68 Workers.java:14           : t2.start();
69     [2 insn w/o sources]
70 Workers.java:15           : t3.start();
71     [2 insn w/o sources]
72 Workers.java:16           : Verify.endAtomic();
73     [1 insn w/o sources]
74 ----- transition #1 thread: 0
75 gov.nasa.jpjf.jvm.choice.ThreadChoiceFromSet {id:"end atomic" ,1/4,isCascaded:false}
76     [2 insn w/o sources]
77 Workers.java:17           : t1.join();
78     [1 insn w/o sources]
79 ----- transition #2 thread: 3
80 gov.nasa.jpjf.jvm.choice.ThreadChoiceFromSet {id:"wait" ,3/3,isCascaded:false}
81     [2 insn w/o sources]
82 Workers.java:23           : synchronized(lock1){
83 Workers.java:24           :   synchronized(lock2){
84 Workers.java:26           :   }
85 Workers.java:27           : }
86 ----- transition #3 thread: 3
87 gov.nasa.jpjf.jvm.choice.ThreadChoiceFromSet {id:"sharedField" ,3/3,isCascaded:false}
88 Workers.java:28           : barrier.sync();
89 Barrier.java:19           : blocked++;
90 Barrier.java:20           : if (blocked<threads){
91 Barrier.java:21           :   boolean expect=!round;
92 Barrier.java:22           :   while (round!=expect) {
93 Barrier.java:23           :     try { wait(); } catch (InterruptedException e) {}
94     [1 insn w/o sources]
95 ----- transition #4 thread: 1
96 gov.nasa.jpjf.jvm.choice.ThreadChoiceFromSet {id:"wait" ,1/2,isCascaded:false}
97     [2 insn w/o sources]
98 Workers.java:23           : synchronized(lock1){
99 Workers.java:24           :   synchronized(lock2){
100 Workers.java:26           :   }
101 Workers.java:27           : }
102 ----- transition #5 thread: 2
103 gov.nasa.jpjf.jvm.choice.ThreadChoiceFromSet {id:"sharedField" ,2/2,isCascaded:false}
104     [2 insn w/o sources]
105 Workers.java:23           : synchronized(lock1){
106 ----- transition #6 thread: 2
107 gov.nasa.jpjf.jvm.choice.ThreadChoiceFromSet {id:"monitorEnter" ,2/2,isCascaded:false}
108 Workers.java:23           : synchronized(lock1){
109 Workers.java:24           :   synchronized(lock2){
110 ----- transition #7 thread: 2
111 gov.nasa.jpjf.jvm.choice.ThreadChoiceFromSet {id:"monitorEnter" ,2/2,isCascaded:false}
112 Workers.java:24           : synchronized(lock2){
113 Workers.java:26           : }
114 Workers.java:27           : }
115 ----- transition #8 thread: 2
116 gov.nasa.jpjf.jvm.choice.ThreadChoiceFromSet {id:"sharedField" ,2/2,isCascaded:false}
117 Workers.java:28           : barrier.sync();
118 Barrier.java:19           : blocked++;
119 Barrier.java:20           : if (blocked<threads){
120 Barrier.java:26           :   round=!round;
121 Barrier.java:27           :   blocked=0;
122 Barrier.java:28           :   notifyAll();
123     [2 insn w/o sources]
124 Barrier.java:30           : }
125 Workers.java:29           :   synchronized(lock2){
126 ----- transition #9 thread: 1
127 gov.nasa.jpjf.jvm.choice.ThreadChoiceFromSet {id:"monitorEnter" ,1/3,isCascaded:false}
128 Workers.java:28           : barrier.sync();
129 ----- transition #10 thread: 1
130 gov.nasa.jpjf.jvm.choice.ThreadChoiceFromSet {id:"monitorEnter" ,1/3,isCascaded:false}
131 Workers.java:28           : barrier.sync();
132 Barrier.java:19           : blocked++;
133 Barrier.java:20           : if (blocked<threads){
134 Barrier.java:21           :   boolean expect=!round;

```

```

135 Barrier.java:22          : while (round!=expect) {
136 Barrier.java:23          : try { wait(); } catch (InterruptedException e) {}
137 [1 insn w/o sources]
138 ----- transition #11 thread: 2
139 gov.nasa.jp.f.jvm.choice.ThreadChoiceFromSet {id:" wait" ,1/2,isCascaded:false}
140 Workers.java:29          : synchronized(lock2){
141 Workers.java:30          : synchronized(lock1){
142 ----- transition #12 thread: 2
143 gov.nasa.jp.f.jvm.choice.ThreadChoiceFromSet {id:" monitorEnter" ,1/2,isCascaded:false}
144 Workers.java:30          : synchronized(lock1){
145 Workers.java:32          : }
146 Workers.java:33          : }
147 Workers.java:34          : }
148 [1 insn w/o sources]
149 ----- transition #13 thread: 3
150 gov.nasa.jp.f.jvm.choice.ThreadChoiceFromSet {id:" terminate" ,1/1,isCascaded:false}
151 [2 insn w/o sources]
152 Barrier.java:23          : try { wait(); } catch (InterruptedException e) {}
153 Barrier.java:22          : while (round!=expect) {
154 Barrier.java:25          : } else {
155 Barrier.java:30          : }
156 Workers.java:29          : synchronized(lock2){
157 Workers.java:30          : synchronized(lock1){
158 Workers.java:32          : }
159 Workers.java:33          : }
160 Workers.java:34          : }
161 [1 insn w/o sources]
162
163 ===== thread ops #71
164 1      3      2      0      trans      insn      loc      : stmt
165 -----
166 |      |      |      |      1354      directcallreturn [synthetic] [run]
167 W:322 |      |      |      |      930      invokevirtual Barrier.java:23 : try { wait()
168 |      |      |      |      930      invokevirtual Barrier.java:23 : try { wait()
169 L:322 |      |      |      |      930      invokevirtual Workers.java:28 : barrier.sync
170 |      |      |      |      507      invokevirtual Barrier.java:28 : notifyAll();
171 |      |      |      |      1      invokevirtual Workers.java:17 : t1.join();
172 |      |      |      |      0
173 |      |      |      |      0
174 S      |      |      |      0

```

Answer

trans. 0-1 The main thread starts the 3 worker threads and start waiting.

trans. 2-3 Thread 3 executes the first work block and enters the barrier.

trans. 4 Thread 1 executes the first work block.

trans. 5-8 Thread 2 executes the first work block and passes through the barrier.

trans. 9-10 Thread 1 enters the Barrier.

trans. 11-12 Thread 2 executes the second work block.

trans. 13 Thread 3 leaves the barrier and executes the second work block.

This leaves Thread 1 waiting at the barrier and thus the program deadlocks.