EXAM
System Validation
(192140122)
8:45-12:15
24-06-2011

- The exercises are worth a total of 10 points.
  The final grade for the exam is $\frac{pts}{10}$. The final grade for the course is $\frac{(hw_1+hw_2)/2+exam}{2}$, provided that the grade for the exam is at least 5.0 (otherwise, the final grade is a 4).
- The exam is open book: all *paper* copies of slides, papers, notes etc. are allowed.

Exercise 1: Formal Tools and Techniques (15 points)

Suppose you are advising on the software development for a modern pigsty control system. The pigsty has movable cage walls, so that depending on the total number of pigs, the pigs can be optimally distributed.

A formal model has been build, describing the intended behaviour of the implementation. Several properties have been verified for this model, *e.g.*,

- moving of cage walls will always ensure enough space for the pigs (avoiding in particular that a pig will be killed by a moving cage wall);
- there are never more than 5 pigs together in a cage (unless the total number of pigs exceeds 5 times the maximum number of cages that can be build, if there are more than 5 pigs together in a cage, they might start killing each other);
- the total of pigs is distributed relatively equally over the cages; and
- if no new pigs come in, or go out, the state of the cage walls does not change.

You are asked to give advice on how to move from this formal model to a reliable implementation. How can the desired properties be mapped into properties for the software? How can it be ensured that the implementation has the same behaviour as the formal model?

Give a short description (no more than 200-250 words) how you would approach this problem.

Answer

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

- the high-level temporal properties should be encoded into properties about the code (as we did for the security automata in the course).
- JPF might be usable to check some of the properties automatically, e.g., properties satisfied by a stable configuration
- for many properties, run-time checking is probably sufficient, because consequences if something goes wrong are not that severe.
- the requirement that pigs do not get killed by moving cage walls, and there are never more than 5 pigs together in a cage is more important, because dead pigs should be avoided. Static checking might be more appropriate here.

Exercise 2: Specification (18 points)

Write formal specifications for the following properties of a traffic light system 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 For a traffic light control system:
Whenever a light is red, it will eventually turn green.

b For a traffic light control system:
At most one light is green.

c For a traffic light control system:
The lights follow the sequence red $\rightarrow$ green $\rightarrow$ yellow $\rightarrow$ red $\rightarrow$ ...

d For an implementation of a game of checkers:
White has won the game if no black stones are left on the field. (N.B. a checkers board is 8 $\times$ 8.)

e For a water tank control system, assume that we have two tanks T_1 and T_2, each 500 centimetres high, and that water can be transferred between the two tanks:
A water tank never overflows, if there is still place in the other tank.

f For the same water tank control system:
The amount of water in the two tanks is balanced, *i.e.*, there is never more than 10 centimetres difference in the height of the water levels.

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 G(Red $\rightarrow$ F Green)

b
```
//@ invariant (\forall Light l1; l1.isGreen()) ==>
                    (\forall Light l2; l1 == l2 || l2.isRed()));
```

c Expressed by a series of constraints in the class Light:
```
//@ constraint \old(isGreen()) ==> isGreen() || isOrange();
//@ constraint \old(isOrange()) ==> isOrange || isRed();
//@ constraint \old(isRed()) ==> isRed || isGreen();
```

d
```
//@ invariant white_won <==
                   (\forall int i,j; 0 <= i && i < 8 && 0 <= j && j < 8;
                             board[i][j] == Empty || board[i][j] == White);
```

e G($\neg$ (overflow$_i$ $\wedge\neg$ full$_{i+1}$))

f
```
//@ invariant Math.abs(T1.waterheight(), T2.waterheight()) <= 10;
```
(assuming that waterheight returns the waterlevel height in centimetres).

Exercise 3: Abstraction (7 points)

Consider the class `ManipulateX` below; it contains a few methods to manipulate the variable `x`. For a static analysis, you want to give an abstract specification of it, where instead of specifying the operations in terms of the concrete value of `x`, you only care whether `x` is even or not. Rewrite the specification in this abstract way, by declaring a
`//@ public model boolean isEvenX;`.

Hint: Evenness of `x` can be tested using `(x % 2) == 0`. All logical operators, including the *exclusive or* (`^` in Java), are allowed.

```
1  public class ManipulateX {
2
3          /*@ spec_public */ private int x;
4
5          /*@ requires dx >= 0;
6              ensures x == \old(x) + dx;
7           */
8          public void moveX(int dx) {
9                  x = x + dx;
10         }
11
12         /*@ ensures x == \old(x) * 2;
13          */
14         public void doubleX() {
15                 x = 2 * x;
16         }
17
18 }
```

Answer

```
1  public class ManipulateX {
2
3          /*@ spec_public */ private int x;
4
5          //@ public model boolean isEvenX;
6
7          //@ represents isEvenX <- (x % 2) == 0;
8
9          /*@ requires dx >= 0;
10             ensures isEvenX == (\old(isEvenX) ^ (dx % 2) == 1);
11          */
12         public void moveX(int dx) {
13                 x = x + dx;
14         }
15
16         /*@ ensures isEvenX;
17          */
```

```
18          public void doubleX() {
19                  x = 2 * x;
20                  //@ assert (x % 2) == 0;
21          }
22
23  }
```

Unfortunately, it does not pass ESC/Java; I guess modulo calculations are too complex.

Exercise 4: Run-time Checking (15 points)

Each subquestion is worth 3 points. Explain your answers.

a Consider class `Pin` in Figure 1. What will happen when the run-time checker executes the `main` method of this class? Explain why.

```
1  package exercises;
2
3  public class Pin {
4
5          private /*@ spec_public */ byte pin[]=new byte[8];
6
7          /*@
8          // pin is defined
9             invariant pin != null;
10
11         // the pin only contain digits (between 0 and 9)
12            invariant (\forall int i; 0 <= i && i < pin.length;
13                                              0 <= pin[i] && pin[i] <= 9);
14         */
15
16     /*@ ensures (\forall int i; 0 <= i && i < pin.length; pin[i] == 0);
17         signals (Exception) false;
18         */
19         public Pin(){
20                 for(int i=0;i<=8;i++) this.pin[i]=0;
21         }
22
23
24         public static void main(String[] args) {
25                 Pin r = new Pin();
26         }
27
28 }
```

Figure 1: Class `Pin`

b Consider class `Game` in Figure 2. What will happen when the run-time checker executes the `main` method of this class? Explain why.

```
1  package exercises;
2
3  public class Game {
4
5          //@ ghost public int state = UNINIT;
6          //@ ghost public static final int UNINIT = 1;
7          //@ ghost public static final int IN_PROGRESS = 2;
8          //@ ghost public static final int FINISHED = 3;
9
10         //@ constraint \old(state) == UNINIT ==> state == IN_PROGRESS;
11         //@ constraint \old(state) == IN_PROGRESS ==> state == FINISHED;
12
13         //@ ensures state == IN_PROGRESS;
14         public Game() {
15                 //@ set state = IN_PROGRESS;
16         }
17
18         //@ requires state == IN_PROGRESS;
19         public void makeMove() {
20
21         }
22
23         //@ requires state == IN_PROGRESS;
24         public void makeFinalMove() {
25                 //@ set state = FINISHED;
26         }
27
28         public static void main (String [] args) {
29                 Game g = new Game();
30                 g.makeFinalMove();
31                 System.out.println("you have won!");
32         }
33 }
```

Figure 2: Class `Game`

c Consider class Append in Figure 3. Method append does not respect its specification. Explain which part of the specification is not respected, and explain how the *JML run-time checker* can be used to demonstrate this.

```
1  package exercises;
2
3  public class Append {
4
5      /*@ spec_public */ private Object[] a = new Object[16];
6
7      /*@ requires a != null;
8          requires o != null;
9          requires (\exists int i; 0 <= i && i <= a.length;
10                     (\forall int j; 0 <= j && j < i; a[j] != null) &&
11                     (\forall int k; i <= k && k < a.length; a[k] == null));
12         ensures (\exists int i; 0 <= i && i < a.length;
13                     (\forall int j; 0 <= j && j < i;
14                                     a[j] != null && a[j] == \old(a[j])) &&
15                     o.equals(a[i]) &&
16                     (\forall int k; i < k && k < a.length; a[k] == null));
17         ensures \old(a.length) <= a.length;
18         signals (Exception) false;
19     */
20     public void append(Object o){
21         int i;
22         for(i=0;i<a.length;i++){
23                 if (a[i]==null) break;
24         }
25         Object[] temp = a;
26             if (i>a.length) {
27                 a=new Object[a.length*2];
28                 for(int k=0;k<temp.length;k++){
29                         a[k]=temp[k];
30                 }
31         };
32         a[i]=o;
33     }
34
35
36 }
```

Figure 3: Class Append

d Consider class `Hotel` in Figure 4 (on the next page). What will happen when the run-time checker executes the `main` method of this class? Explain why.

```
1 public class Hotel {
2     /*@ spec_public */ private int nr_of_rooms;
3     /*@ spec_public */ private String [] reservations;
4     /*@ spec_public */ private String [] rooms;
5     //@ invariant rooms != null;
6     //@ invariant reservations != null;
7
8     //@ requires nr >= 0;
9     public Hotel (int nr) {
10        nr_of_rooms = nr;
11        reservations = new String [nr_of_rooms];
12        rooms = new String [nr_of_rooms]; }
13
14    public void removeReservation (String name) {
15        for (int i = 0; i < nr_of_rooms; i++) {
16            if (name.equals(reservations[i])) {
17                reservations[i] = null; }}}
18
19
20    //@ requires name != null;
21    public void addReservation(String name) {
22        for (int i = 0; i < nr_of_rooms; i++) {
23            if (reservations[i] == null) {
24                reservations[i] = name;
25                return; }}}
26
27    /*@ requires name != null;
28        requires 0 <= nr && nr < nr_of_rooms;
29        ensures name.equals(rooms[nr]);
30        ensures (\forall int i; 0 <= i && i < nr_of_rooms && i != nr;
31                                    rooms[i] == \old(rooms[i]));
32        ensures !(\exists int i; 0 <= i && i < nr_of_rooms;
33                                     name.equals(reservations[i]));
34     */
35    public void checkIn (String name, int nr) {
36        rooms[nr] = name;
37        removeReservation(name); }
38
39    public static void main (String [] args) {
40        Hotel h = new Hotel (5);
41        h.addReservation("A");
42        h.addReservation("B");
43        h.checkIn("C", 3);
44        h.checkIn("A", 2); }
45 }
```

Figure 4: Class `Hotel`

e Consider class `Computations` in Figure 5. What will happen when the run-time checker executes the `main` method of this class? Explain why.

```
1  package exercises;
2
3  public class Computations {
4
5          /*@ spec_public */ private int v;
6          //@ invariant v >= 0;
7
8          //@ requires x >= 0;
9          Computations(int x) {
10                 v = x;
11         }
12
13         void square () {
14                 v =  v * v;
15         }
16
17         void squaredDistance(int u) {
18                 v = v - u;
19                 square();
20         }
21
22         int getV() {
23                 return v;
24         }
25
26         public static void main (String [] args) {
27                 Computations c = new Computations(2);
28                 c.square();
29                 c.squaredDistance(5);
30                 System.out.println(c.getV());
31         }
32 }
```

Figure 5: Class `Computations`

Answers

a A JMLInternalExceptionalPostconditionError will be returned. The method ends with an `ArrayIndexOutOfBoundsException` and the method specification says `signals false;`, *i.e.*, the method should not terminate exceptionally.

b The run-time checker will not give any warnings, because the methods that are executed (the constructor and `makeFinalMove`) respect the specifications of the class - including the constraint.

c The exceptional postcondition is violated. The following `main` method would make the run-time checker give a message about this, *i.e.*, return a JMLInternalExceptionalPostcondition-Error.

```
1
2       public static void main (String[] args) {
3           Append r = new Append();
4           Object o = new Object();
5           for (int i = 0; i < 16; i++) {
6                   r.append(o);
7           }
8           Object o2 = new Object();
9           r.append(o2);
10      }
```

d Nothing will go wrong, because no method specification is violated during the execution.

e The call to `square` from within `squaredDistance` will throw a JMLInvariantError, because the invariant that `v` should be positive is violated by the assignment `v = v - u` (where `v` is 4 and `u` is 5).

Exercise 5: Static Checking (15 points)

Each subquestion is worth 3 points. Explain your answers.

a Consider again class `Game` in Figure 2 (used in exercise 4b). Explain what happens when the static checker is applied to this class.

b Consider again class `Hotel` in Figure 4 (used in exercise 4d). Explain how the class should be adapted so that the static checker does not signal any errors in the class. Why are these changes necessary?

c Consider class `Point` in Figure 6. What will happen when the static checker is applied to this class? Explain why.

```
1  package exercises;
2
3  public class Point {
4
5          /*@ spec_public */ private int x;
6          /*@ spec_public */ private int y;
7
8          /*@ assignable x,y;
9              ensures x == \old(x) + dx;
10          */
11         void moveX(int dx) {
12                 x = x + dx;
13         }
14
15         /*@ assignable x, y;
16             ensures y == \old(y) + dy;
17          */
18         void moveY(int dy) {
19                 y = y + dy;
20         }
21
22         /*@ assignable x, y;
23             ensures x == \old(x) + dx;
24             ensures y == \old(y) + dy;
25          */
26         void move(int dx, int dy) {
27                 moveX(dx);
28                 moveY(dy);
29         }
30 }
```

Figure 6: Class `Point`

d Consider class `Car` in Figure 7. What will happen when the static checker is applied to this class? Explain why.

```
1  package exercises;
2
3  public class Car {
4
5          /*@ spec_public */ private int speed;
6          /*@ spec_public */ private int trip_counter;
7          /*@ spec_public */ private int km_counter;
8
9          //@ invariant 0 <= speed && speed <= 200;
10         //@ invariant 0 <= trip_counter && trip_counter < 1000;
11         //@ invariant 0 <= km_counter && km_counter < 1000000;
12
13         public Car () {
14                 speed = 0;
15                 trip_counter = 0;
16                 km_counter = 0;
17         }
18
19         public void resetTripCounter () {
20                 trip_counter = 0;
21         }
22
23         public void makeTrip(int distance) {
24                 trip_counter = trip_counter + distance;
25                 km_counter = km_counter + distance;
26         }
27 }
```

Figure 7: Class `Car`

e Consider class `Insert` in Figure 8. What would bean appropriate loop invariant to make this method pass verification with the static checker (using the ESC/Java loopsafe option).

```
1  package exercises;
2
3  public class Insert {
4
5          /*@ spec_public */ private int[] a;
6
7          // a is sorted
8          //@ invariant a != null;
9          //@ invariant (\forall int i; 0 <= i && i < a.length - 1;
10         //@                            a[i] <= a[i + 1]);
11
12         //@ requires a != null;
13         //@ requires (\forall int i; 0 <= i && i < a.length - 1;
14         //@                           a[i] <= a[i + 1]);
15         Insert(int [] a) {
16                 this.a = a;
17         }
18
19         /*@
20             ensures a[\result] == elem;
21             */
22         public int insert (int elem) {
23                 int j = 0;
24                 for (;j < a.length - 1 && a[j] < elem; j++) {};
25                 a[j] = elem;
26                 return j;
27         }
28
29 }
```

Figure 8: Class `Insert`

Answers

a The static checker checks for all methods whether they respect the specification. It finds that `makeMove` violates the constraint, because it does not change the `state` of the game.

b A specification for `removeReservations` has to be given, as shown below. This is necessary, because the static checker verifies in a modular way, *i.e.*, abstracting every method calls by its specification.

```
1 //@ requires name != null;
2 //@ ensures !(\exists int i; 0 <= i && i < nr_of_rooms;
3 //@                           name.equals(reservations[i]));
4 public void removeReservation (String name) {
5     for (int i = 0; i < nr_of_rooms; i++) {
6         if (name.equals(reservations[i])) {
7                 reservations[i] = null;
8         }
9     }
10 }
```

c It says that the postcondition of `move` is possibly not established. The `assignable` clause of `moveY` specifies that `x` may be changed by this method, without actually specifying how, thus verification of `move` - that uses only the specification of `move` - cannot guarantee anything about the final value of `x` anymore.

d Method `makeTrip` might violate the invariant of `Car`, because the method does not have a precondition to constrain the possible values of the parameter `distance` - or alternatively, the method implementation does not check that the counters get over their limit (and should be reset to 0). Notice that both options would be okay individually; it does not make sense to have them both.

e
```
1 //@ loop_invariant (\forall int i; 0 <= i && i < a.length - 1;
2 //@                                 a[i] <= a[i + 1]);
3 //@ loop_invariant (\forall int i; 0 <= i && i < j; a[i] < elem);
4 //@ loop_invariant 0 <= j && j <= a.length - 1;
5 //@ loop_invariant a != null;
```

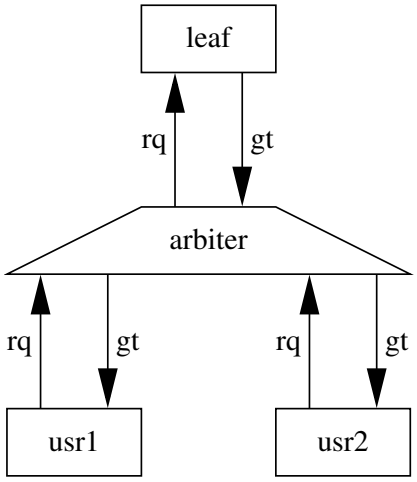

Figure 9: The lock hierarchy

Exercise 6: Modeling (15 points)

In this exercise, we use a locking protocol for access to resources that is inspired by hardware.

The protocol works as follows:
Communication between a server and a client uses two variables: `rq` (ReQuest) and `gt` (GranT). Initially both variables are false. When the client wants access to the resource, the client sets `rq` to true. When the server sees the request and is able to acknowledge it, it will set `gt` to true. When the client sees that `gt` has become true, it knows that it has access to the resource. If the client is finished, it sets `rq` to false. When the server detects this, it will also set `gt` to false.

The client

- is not required to free the resource;
- may not set `rq` to false before the arbiter has set `gt` to true;
- may not set `rq` to true before the arbiter has set `gt` to false.

Figure 10 contains part of an SMV model that uses this protocol in a hierarchical way. There are two users and there is a single protected resource accessed via the Leaf model. The two users compete for access to this resource. To control this, an intermediate Arbiter is used. The Arbiter receives requests from the two users and forwards these requests to the Leaf module. Both for the receiving of the requests (from user to Arbiter) and for the forwarding of the request (from Arbiter to Leaf), the protocol described above is used, see Fig.9).

When the Arbiter receives the grant from the Leaf module, it passes it on to one of the two competing users, ensuring that at most one of the users at the same time has access. Thus the following properties have to be respected by the Arbiter module:

```
LTLSPEC G ( gt_1 -> !gt_2 )
LTLSPEC G ( gt_2 -> !gt_1 )
```

The arbiter is biased towards the first user. That is, if the first user requests access and the second did not yet request access at that moment, then access is eventually granted to the first user.

```
LTLSPEC G ( ( !rq_2 & rq_1 ) -> F gt_1 )
```

```
MODULE Leaf(request,grant)
ASSIGN
    init(grant) := FALSE;
    next(grant) := request;

MODULE User(request,grant)
VAR
    state:{safe,enter,critical,leave};
ASSIGN
    init(state) := safe;
    next(state) := case
        state=safe           : {safe,enter};
        state=enter & !grant : enter;
        state=enter & grant  : critical;
        state=critical       : {critical,leave};
        state=leave & grant  : leave;
        state=leave & !grant : safe;
    esac;
    init(request) := FALSE;
    next(request) := case
        state=enter : TRUE;
        state=leave : FALSE;
        TRUE        : request;
    esac;

MODULE main
VAR
    rq      : boolean;
    gt      : boolean;
    rq_1    : boolean;
    gt_1    : boolean;
    rq_2    : boolean;
    gt_2    : boolean;
    nil     : Leaf(rq,gt);
    arb     : Arbiter(rq,gt,rq_1,gt_1,rq_2,gt_2);
    usr1    : User(rq_1,gt_1);
    usr2    : User(rq_2,gt_2);
```

Figure 10: SMV code for a prioritized lock

However, if the second user requests access then access is only guaranteed to be eventually granted to the second user if the first user never requests access.

```
LTLSPEC G ( ( G !rq_1 & rq_2 ) -> F gt_2 )
```

a (*10 points*) Write an implementation of the Arbiter module.

b (*5 points*) The following formula does not hold in the model. Explain why.

```
LTLSPEC G ( rq_1 -> F gt_1 )
```

Answer

a The Arbiter module:

```
MODULE Arbiter(rq,gt,rq1,gt1,rq2,gt2)
ASSIGN
    init(rq):=FALSE;
    init(gt1):=FALSE;
    init(gt2):=FALSE;
    next(rq) := case
        !gt & (rq1 | rq2) : TRUE;
        gt & !rq1 & !rq2 : FALSE;
        TRUE : rq;
    esac;
    next(gt1) := case
        rq & gt & rq1 & !gt2 : TRUE;
        !rq1 : FALSE;
        TRUE : gt1 ;
    esac;
    next(gt2) := case
        rq & gt & !rq1 & rq2 : TRUE;
        !rq2 : FALSE;
        TRUE : gt2;
    esac;
```

b The formula expresses that if user 1 requests access then that access is eventually granted. This fails because user 2 can request and be given access. Subsequently user 2 can keep this access forever. If user 1 requests access after the access to user 2 has been granted then user 1 will never be granted access:

```
1 -- specification G (rq_1 -> F gt_1) is false
2 -- as demonstrated by the following execution sequence
3 Trace Description: LTL Counterexample
4 Trace Type: Counterexample
5 -> State: 1.1 <-
6   rq = FALSE
7   gt = FALSE
8   rq_1 = FALSE
9   gt_1 = FALSE
10  rq_2 = FALSE
11  gt_2 = FALSE
12  usr1.state = safe
```

```
13   usr2.state = safe
14 -> State: 1.2 <-
15   usr2.state = enter
16 -> State: 1.3 <-
17   rq_2 = TRUE
18 -> State: 1.4 <-
19   rq = TRUE
20 -> State: 1.5 <-
21   gt = TRUE
22   usr1.state = enter
23 -> State: 1.6 <-
24   rq_1 = TRUE
25   gt_2 = TRUE
26 -- Loop starts here
27 -> State: 1.7 <-
28   usr2.state = critical
29 -> State: 1.8 <-
```

Exercise 7: Traces (15 points)

Consider the following small Java program, that executes two threads.

```
1  class Point {
2      public double x,y;
3      Point(double x,double y){
4          this.x=x;
5          this.y=y;
6      }
7      public void shift(double dx,double dy){
8          x=x+dx;
9          y=y+dy;
10     }
11     public double dist(Point p){
12         double x1=x;
13         double x2=p.x;
14         double y1=y;
15         double y2=p.y;
16         return Math.sqrt(Math.pow(x1-x2,2)+Math.pow(y1-y2,2));
17     }
18 }
19
20 class MovingObject implements Runnable {
21     private Point source,target,current;
22     private int steps;
23     private MovingObject peer;
24     private double dx,dy;
25
26     MovingObject(Point source,Point target,int steps){
27         this.source=source;
28         this.target=target;
29         this.steps=steps;
30         current=new Point(source.x,source.y);
31         dx=(target.x-source.x)/steps;
32         dy=(target.y-source.y)/steps;
33     }
34     public void keepAwayFrom(MovingObject peer){
35         this.peer=peer;
36     }
37     public double dist(Point p){
38         return current.dist(p);
39     }
40     public void run(){
41         for(int i=0;i<steps;i++){
42             current.shift(dx,dy);
43             assert peer.current.dist(current) > 1.1 ;
44         }
45         assert target.dist(current) < 0.001;
46     }
47 }
48
49 public class TwinObjects {
50     public static void main(String args[]){
51         MovingObject obj1=new MovingObject(new Point(1,0),new Point(4,3),3);
52         MovingObject obj2=new MovingObject(new Point(0,1),new Point(3,4),3);
```

```
53          obj1.keepAwayFrom(obj2);
54          obj2.keepAwayFrom(obj1);
55          Thread t1 = new Thread(obj1);
56          Thread t2 = new Thread(obj2);
57          t1.start();
58          t2.start();
59          try {
60              t1.join();
61              t2.join();
62          } catch (InterruptedException e) {};
63      }
64  }
```

To analyse this program, we have used JPF. The input for JPF was

```
# class to verify
target = TwinObjects

# where to find byte code and source
classpath=.
sourcepath=.

# how to report errors
report.console.property_violation=error,snapshot,trace

# set heuristic
search.class = .search.heuristic.BFSHeuristic
```

The result was the following trace:

```
JavaPathfinder v6.0 - (C) RIACS/NASA Ames Research Center


====================================================== system under test
application: TwinObjects.java

====================================================== search started: 6/14/11 8:47 PM

====================================================== error #1
gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty
java.lang.AssertionError
at MovingObject.run(TwinObjects.java:43)


====================================================== snapshot #1
thread index=0,name=main,status=WAITING,this=java.lang.Thread@0,target=null,priority=5,lockCount=0,suspendCount=0
  waiting on: java.lang.Thread@14b
  call stack:
at java.lang.Thread.join(Thread.java:-1)
at TwinObjects.main(TwinObjects.java:60)

thread index=1,name=Thread-0,status=RUNNING,this=java.lang.Thread@14b,target=MovingObject@140,priority=5,lockCount=0,suspend
  call stack:
at MovingObject.run(TwinObjects.java:43)

thread index=2,name=Thread-1,status=RUNNING,this=java.lang.Thread@15d,target=MovingObject@147,priority=5,lockCount=0,suspend
  call stack:
at Point.shift(TwinObjects.java:9)
at MovingObject.run(TwinObjects.java:42)


====================================================== trace #1
------------------------------------------------------ transition #0 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet[id="root",isCascaded:false,{>main}]
```

```
    [2894 insn w/o sources]
TwinObjects.java:51           : MovingObject obj1=new MovingObject(new Point(1,0),new Point(4,3),3);
    [1 insn w/o sources]
TwinObjects.java:20           : class MovingObject implements Runnable {
    [2 insn w/o sources]
TwinObjects.java:20           : class MovingObject implements Runnable {
    [1 insn w/o sources]
TwinObjects.java:51           : MovingObject obj1=new MovingObject(new Point(1,0),new Point(4,3),3);
TwinObjects.java:3            : Point(double x,double y){
    [1 insn w/o sources]
TwinObjects.java:4            : this.x=x;
TwinObjects.java:5            : this.y=y;
TwinObjects.java:6            : }
TwinObjects.java:51           : MovingObject obj1=new MovingObject(new Point(1,0),new Point(4,3),3);
TwinObjects.java:3            : Point(double x,double y){
    [1 insn w/o sources]
TwinObjects.java:4            : this.x=x;
TwinObjects.java:5            : this.y=y;
TwinObjects.java:6            : }
TwinObjects.java:51           : MovingObject obj1=new MovingObject(new Point(1,0),new Point(4,3),3);
TwinObjects.java:26           : MovingObject(Point source,Point target,int steps){
    [1 insn w/o sources]
TwinObjects.java:27           : this.source=source;
TwinObjects.java:28           : this.target=target;
TwinObjects.java:29           : this.steps=steps;
TwinObjects.java:30           : current=new Point(source.x,source.y);
TwinObjects.java:3            : Point(double x,double y){
    [1 insn w/o sources]
TwinObjects.java:4            : this.x=x;
TwinObjects.java:5            : this.y=y;
TwinObjects.java:6            : }
TwinObjects.java:30           : current=new Point(source.x,source.y);
TwinObjects.java:31           : dx=(target.x-source.x)/steps;
TwinObjects.java:32           : dy=(target.y-source.y)/steps;
TwinObjects.java:33           : }
TwinObjects.java:51           : MovingObject obj1=new MovingObject(new Point(1,0),new Point(4,3),3);
TwinObjects.java:52           : MovingObject obj2=new MovingObject(new Point(0,1),new Point(3,4),3);
TwinObjects.java:3            : Point(double x,double y){
    [1 insn w/o sources]
TwinObjects.java:4            : this.x=x;
TwinObjects.java:5            : this.y=y;
TwinObjects.java:6            : }
TwinObjects.java:52           : MovingObject obj2=new MovingObject(new Point(0,1),new Point(3,4),3);
TwinObjects.java:3            : Point(double x,double y){
    [1 insn w/o sources]
TwinObjects.java:4            : this.x=x;
TwinObjects.java:5            : this.y=y;
TwinObjects.java:6            : }
TwinObjects.java:52           : MovingObject obj2=new MovingObject(new Point(0,1),new Point(3,4),3);
TwinObjects.java:26           : MovingObject(Point source,Point target,int steps){
    [1 insn w/o sources]
TwinObjects.java:27           : this.source=source;
TwinObjects.java:28           : this.target=target;
TwinObjects.java:29           : this.steps=steps;
TwinObjects.java:30           : current=new Point(source.x,source.y);
TwinObjects.java:3            : Point(double x,double y){
    [1 insn w/o sources]
TwinObjects.java:4            : this.x=x;
TwinObjects.java:5            : this.y=y;
TwinObjects.java:6            : }
TwinObjects.java:30           : current=new Point(source.x,source.y);
TwinObjects.java:31           : dx=(target.x-source.x)/steps;
TwinObjects.java:32           : dy=(target.y-source.y)/steps;
TwinObjects.java:33           : }
TwinObjects.java:52           : MovingObject obj2=new MovingObject(new Point(0,1),new Point(3,4),3);
TwinObjects.java:53           : obj1.keepAwayFrom(obj2);
TwinObjects.java:35           : this.peer=peer;
TwinObjects.java:36           : }
TwinObjects.java:54           : obj2.keepAwayFrom(obj1);
TwinObjects.java:35           : this.peer=peer;
```

```
  TwinObjects.java:36            : }
  TwinObjects.java:55            : Thread t1 = new Thread(obj1);
      [190 insn w/o sources]
  TwinObjects.java:55            : Thread t1 = new Thread(obj1);
  TwinObjects.java:56            : Thread t2 = new Thread(obj2);
      [143 insn w/o sources]
  TwinObjects.java:56            : Thread t2 = new Thread(obj2);
  TwinObjects.java:57            : t1.start();
      [1 insn w/o sources]
------------------------------------------------------ transition #1 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet[id="start",isCascaded:false,{main,>Thread-0}]
      [2 insn w/o sources]
  TwinObjects.java:58            : t2.start();
      [1 insn w/o sources]
------------------------------------------------------ transition #2 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet[id="start",isCascaded:false,{main,Thread-0,>Thread-1}]
  TwinObjects.java:41            : for(int i=0;i<steps;i++){
------------------------------------------------------ transition #3 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet[id="sharedField",isCascaded:false,{main,Thread-0,>Thread-1}]
  TwinObjects.java:41            : for(int i=0;i<steps;i++){
  TwinObjects.java:42            : current.shift(dx,dy);
------------------------------------------------------ transition #4 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet[id="sharedField",isCascaded:false,{main,Thread-0,>Thread-1}]
  TwinObjects.java:42            : current.shift(dx,dy);
------------------------------------------------------ transition #5 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet[id="sharedField",isCascaded:false,{main,Thread-0,>Thread-1}]
  TwinObjects.java:42            : current.shift(dx,dy);
------------------------------------------------------ transition #6 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet[id="sharedField",isCascaded:false,{main,Thread-0,>Thread-1}]
  TwinObjects.java:42            : current.shift(dx,dy);
  TwinObjects.java:8             : x=x+dx;
------------------------------------------------------ transition #7 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet[id="sharedField",isCascaded:false,{main,Thread-0,>Thread-1}]
      [2 insn w/o sources]
  TwinObjects.java:60            : t1.join();
      [1 insn w/o sources]
------------------------------------------------------ transition #8 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet[id="wait",isCascaded:false,{Thread-0,>Thread-1}]
  TwinObjects.java:41            : for(int i=0;i<steps;i++){
  TwinObjects.java:42            : current.shift(dx,dy);
  TwinObjects.java:8             : x=x+dx;
  TwinObjects.java:9             : y=y+dy;
  TwinObjects.java:10            : }
  TwinObjects.java:43            : assert peer.current.dist(current) > 1.1 ;
------------------------------------------------------ transition #9 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet[id="sharedField",isCascaded:false,{Thread-0,>Thread-1}]
  TwinObjects.java:43            : assert peer.current.dist(current) > 1.1 ;
  TwinObjects.java:12            : double x1=x;
------------------------------------------------------ transition #10 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet[id="sharedField",isCascaded:false,{Thread-0,>Thread-1}]
  TwinObjects.java:8             : x=x+dx;
------------------------------------------------------ transition #11 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet[id="sharedField",isCascaded:false,{Thread-0,>Thread-1}]
  TwinObjects.java:8             : x=x+dx;
  TwinObjects.java:9             : y=y+dy;
------------------------------------------------------ transition #12 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet[id="sharedField",isCascaded:false,{Thread-0,>Thread-1}]
  TwinObjects.java:12            : double x1=x;
  TwinObjects.java:13            : double x2=p.x;
  TwinObjects.java:14            : double y1=y;
------------------------------------------------------ transition #13 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet[id="sharedField",isCascaded:false,{Thread-0,>Thread-1}]
  TwinObjects.java:14            : double y1=y;
  TwinObjects.java:15            : double y2=p.y;
  TwinObjects.java:16            : return Math.sqrt(Math.pow(x1-x2,2)+Math.pow(y1-y2,2));
      [1 insn w/o sources]
------------------------------------------------------ transition #14 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet[id="monitorEnter",isCascaded:false,{>Thread-0,Thread-1}]
      [14 insn w/o sources]
  TwinObjects.java:16            : return Math.sqrt(Math.pow(x1-x2,2)+Math.pow(y1-y2,2));
```

```
      [2 insn w/o sources]
  TwinObjects.java:16            : return Math.sqrt(Math.pow(x1-x2,2)+Math.pow(y1-y2,2));
      [2 insn w/o sources]
  TwinObjects.java:16            : return Math.sqrt(Math.pow(x1-x2,2)+Math.pow(y1-y2,2));
      [2 insn w/o sources]
  TwinObjects.java:16            : return Math.sqrt(Math.pow(x1-x2,2)+Math.pow(y1-y2,2));
  TwinObjects.java:43            : assert peer.current.dist(current) > 1.1 ;
      [21 insn w/o sources]

====================================================== results
error #1: gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty "java.lang.AssertionError  at MovingObject.run(Twin..."

====================================================== statistics
elapsed time:       0:00:01
states:             new=427, visited=364, backtracked=790, end=2
search:             maxDepth=15, constraints hit=0
choice generators:  thread=360 (signal=0, lock=9, shared ref=256), data=0
heap:               new=513, released=56, max live=355, gc-cycles=758
instructions:       17404
max memory:         81MB
loaded code:        classes=83, methods=1167

====================================================== search finished: 6/14/11 8:47 PM
```

Analyse the counter example and explain why the assertion was triggered.

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

Answer

0-1,7 The main thread starts threads 1 and 2 and blocks waiting for thread 1.

2-6 Thread 2 enters the first iteration and stops right before updating x.

8,9 Thread 1 runs through the first iteration and then calls distance; it stops just before getting the value of x from thread 2.

10,11 Thread 2 writes x and stops before updating y.

12-14 Thread 1 gets the values of x and y from thread 2 and computes the distance between the points, giving rise to an exception.
The exception happens because thread 1 is at $(2, 1)$, while thread 2 is moving from $(0, 1)$ to $(1, 2)$, but due to the update order, Thread 1 thinks it is at $(1, 1)$, which is too close.