COMS 6998 - Formal Verification of HW/SW Systems (Final Report)

Model Checking in Software & GUI Applications using Java Path Finder

BY: Raghavendra Sirigeri - rs3603
Ishaan Sayal - is2439

Overview:
This project aims to apply model checking algorithms to generic suite of programs in Java and in GUI based applications to detect for inconsistencies like deadlocks, races and other system specific errors. The tools used to perform this checking are Java Path Finder and Findbugs. GUI applications in Java are checked using Java Path Finder and its AWT extension.

Description:
JPF-CORE:
The JPF core is a Virtual Machine (VM) for Java™ bytecode, which means it is a program which you give Java programs to execute. It is used to find defects in these programs, so you also need to give it the properties to check for as input. JPF gets back to you with a report that says if the properties hold and/or which verification artifacts have been created by JPF for further analysis (like test cases).

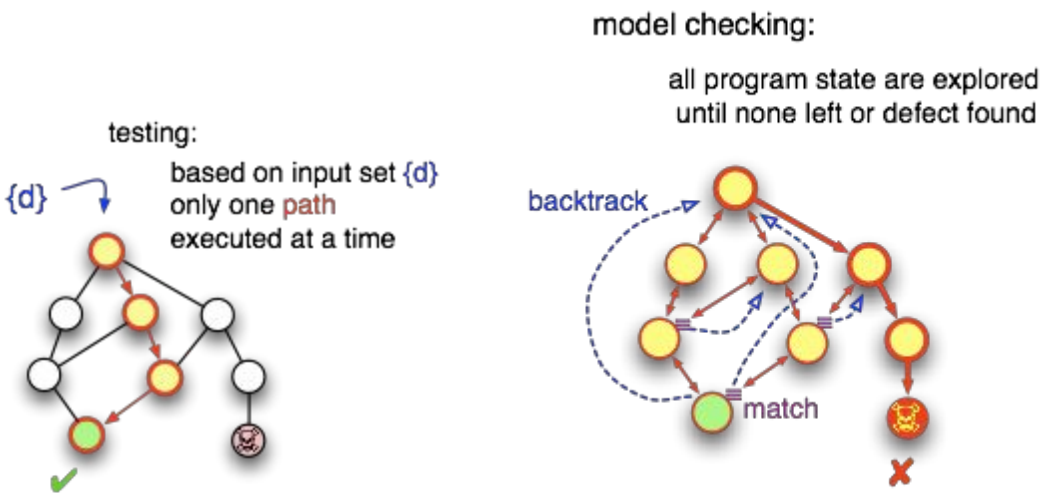
JPF is a VM with several twists. It is a VM running on top of a VM.
The default instruction set makes use of the next JPF feature: *execution choices*. JPF can identify points in your program from where execution could proceed differently, then systematically explore all of them. This means JPF (theoretically) executes *all* paths through your program, not just one like a normal VM. Typical choices are different scheduling sequences or random values, but again JPF allows you to introduce your own type of choices like user input or state machine events.

The number of paths can grow out of hand, and it usually will. This is what software model checking calls the *state explosion problem*. The first line of defense employed by JPF is *state matching*: each time JPF reaches a choice point, it checks if it has already seen a similar program state, in which case it can safely abandon this path, *backtrack* to a previous choice point that has still unexplored choices, and proceed from there. That's right, JPF can restore program states, which is like telling a debugger "go back 100 instructions".

The core checks for defects that can be identified by the VM without you having to specify any property: deadlocks and unhandled exceptions (which also covers Java assert expressions). We call these *non-functional* properties, and no application should violate them. But JPF doesn't stop there - you can define your own properties, which is mostly done with so called ***listeners***, little "plugins" that let you closely monitor all actions taken by JPF, like executing single instructions, creating objects, reaching a new program state and many more. A typical example of such a listener-implemented property is a race detector, which identifies unsynchronized access to shared variables in concurrent programs (the JPF core comes with two of them).

Testing techniques differ on how we choose the input (random, "interesting" problem domain values like corner cases etc.), and on how much knowledge about the system under test (SUT) and its execution environment we assume (black/grey/white box), which especially affects how we can define and check correct behavior. This involves a lot of educated guesses, or as Edsger Dijkstra has put it: "program testing can at best show the presence of errors but never their absence". We usually compensate this by performing "enough" tests - which would be the next guess. Testing complex systems can easily turn into finding a needle in a haystack. If you are a good tester, you make the right guesses and hit the defect that is inevitably there. If not, don't worry - your users will find it later.

Model Checking as Formal Method does not depend on guesses. At least as the theory goes, if there is a violation of a given specification, model checking will find it. Model checking is supposed to be a rigorous method that exhaustively explores all possible SUT behaviors.



The following codes were analysed using JPF.

1. Rand.java : A program to divide two random numbers.
2. CheckRace.java : A program that runs two thrreads and accesses the same variales.
3. CheckNPE.java : A program that does not initialise objects properly.
4. Deadlock.java : A program that concatenates two strings

After running the above codes we are able to detect the possibility of division by zero which is quite a common error. The **CheckRace** code tries to increment the same variable using two threads and an improper final value is output. The **CheckNPE** code returns a null pointer exception as the individual objects have not been initialised, these can be usually eliminated through try catch blocks but need to be corrected if found. Deadlock code tries to concatenate two strings in the two threads, the locks are held by the 2 threads in such a way that the execution cant progress, hence there is a deadlock
The results of running Java Path Finder on these codes and the statistics related to them are published in Appendix A of the report.

FindBugs:
FindBugs is a program which uses static analysis to look for bugs in Java code.

Static program analysis is the analysis of computer software that is performed without actually executing programs (analysis performed on executing programs is known as dynamic analysis).In most cases the analysis is performed on some version of the source code, and in the other cases, some form of the object code.

The term is usually applied to the analysis performed by an automated tool, with human analysis being called program understanding, program comprehension, or code review. Software inspections and Software walkthroughs are also used in the latter case.

As we can see below the Findbugs tool will only help us in detecting a deadlock in the deadlock code. Findbugs does not support exceptions, arithmetic exceptions, etc.
Below is a list of tools and a comparative analysis of the functions they perform.

FindBugs is a very easy to use tool and quite fast for statistical analysis. It is an indispensable to find bugs in code on the go.

Bug Category	Example	ESC/Java	FindBugs	JLint	PMD
General	Null dereference	√*	√*	√*	√
Concurrency	Possible deadlock	√*	√	√*	√
Exceptions	Possible unexpected exception	√*			
Array	Length may be less than zero	√		√*	
Mathematics	Division by zero	√*		√	
Conditional, loop	Unreachable code due to constant guard		√		√*
String	Checking equality using == or !=		√	√*	√
Object overriding	Equal objects must have equal hashcodes		√*	√*	√*
I/O stream	Stream not closed on all paths		√*		
Unused or duplicate statement	Unused local variable		√		√*
Design	Should be a static inner class		√*		
Unnecessary statement	Unnecessary return statement				√*

√ - tool checks for bugs in this category * - tool checks for this specific example

Findbugs is applied to Deadlock.java code and the tool identifies where exactly the problem is in the code. The results obtained by running Findbugs on the Deadlock Code is shown in Appendix B.

Model Checking in GUI Applications:

JPF_AWT:
Verification of GUIs is challenged by a number of GUI-specific aspects:

(1) GUI applications are open event-driven systems that engage in long running interactions with users and other software or hardware components. The number of possible interaction sequences may be too large for manual or even automated testing.

(2) GUI applications are typically implemented on top of large frameworks such as Java's Abstract Window Toolkit (AWT) and Swing libraries. The application code mostly provides callbacks for the framework, which is responsible for obtaining user input and dispatching it to the respective user interface components such as buttons and lists. A large part of the verification relevant behavior of the system under test is defined by the framework, not the application code itself.

(3) GUI applications are inherently concurrent. Even if the application itself is not multi-threaded, the framework always uses multiple processes or threads (e.g., event dispatcher thread, X server process). On top of the framework concurrency, GUI applications have to employ explicit concurrency for lengthy computations in order to guarantee responsiveness of the user interface . In addition, large parts of the GUI framework are system global, i.e., shared between different applications that should not affect each other.
Model checking techniques are specifically designed to exercise all possible thread interleavings of the system. However, in the context of large and complex GUI frameworks, software model checking has a huge barrier to cross in terms of scalability.

JPF-AWT model checking GUI applications addresses the challenges of model checking GUIs by (1) allowing the user to specify sets of interaction sequences using a simple yet powerful formal notation, (2) incorporating support for the actual AWT/Swing libraries, and
(3) employing the model checking engine of Java PathFinder (JPF) to verify all possible thread interleavings and input sequences. The tool is called JPF-AWT and is implemented as a JPF extension.

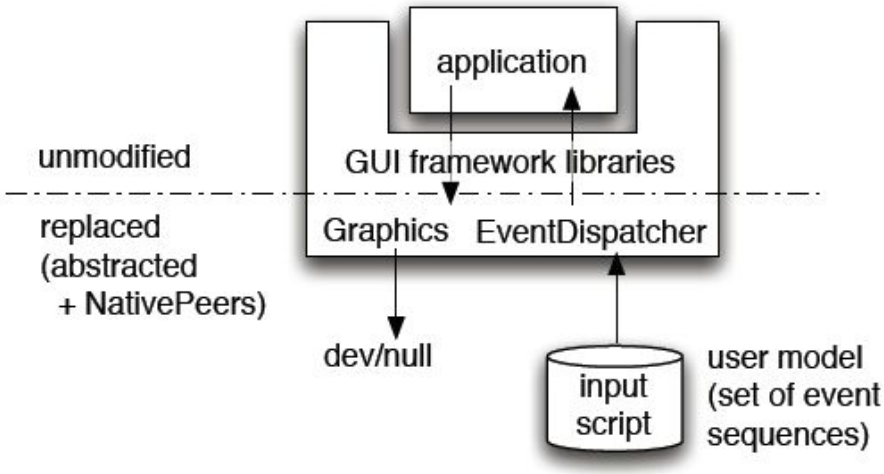
JPF-AWT is implemented as an extension of JPF, which is an open source, explicit state software model checker for Java bytecode. JPF features its own Java Virtual Machine (JVM) that is capable of storing, matching and restoring the state of the executed bytecode program. Off the shelf, JPF can check for property violations such as deadlocks, data races and unhandled exceptions, using on-the-fly partial order reduction and other techniques to reduce the number of program states that have to be explored. However, JPF's main quality is to provide a number of well defined extension mechanisms that allow

• introduction of new state space branches other than scheduling points (extensible ChoiceGenerators)

• observation of state space exploration events such as backtracking (Listeners)

• abstraction of standard libraries (Model and NativePeer classes) JPF-AWT tool makes use of all three extension mechanisms.

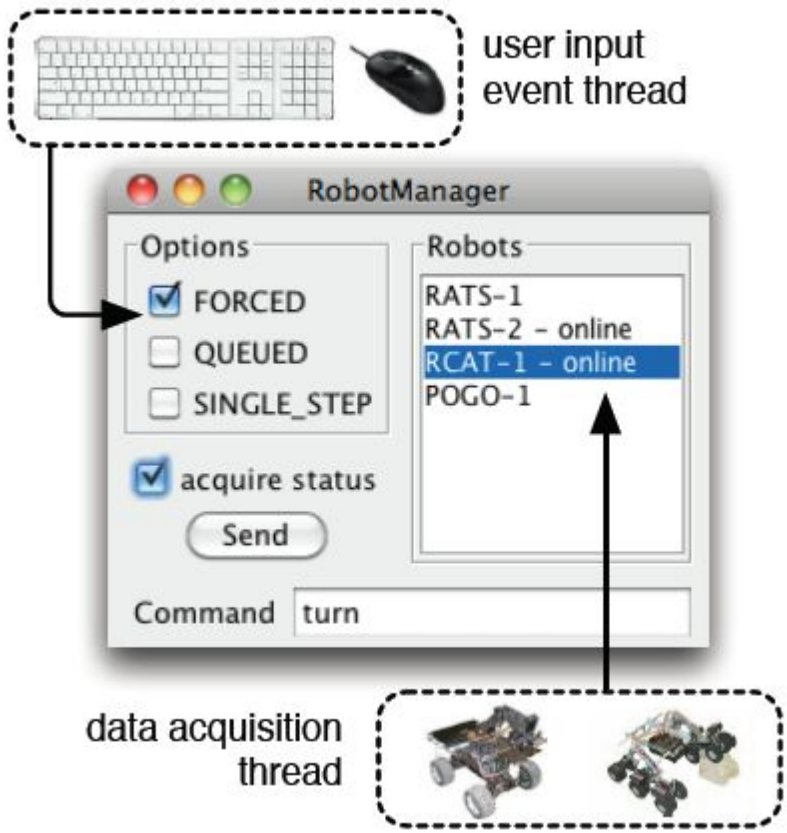
Since JPF verifies Java bytecode, JPF-AWT targets GUI applications that use the standard Java AWT and Swing framework libraries. The main objective for the JPF-AWT design is to check the application behavior against a potentially large set of different input sequences, while preserving as much of the GUI framework as possible, which includes threading structure, window composition and callback notification. This is achieved by replacing only the low-level, platform-specific parts of the framework libraries that handle rendering and input acquisition, leaving the application itself completely unmodified. A block diagram of JPF-AWT is shown below.



Conceptually, the main component of JPF-AWT is a replaced **java.awt.EventDispatchThread** which models non-deterministic user input, using JPF's NativePeer mechanism to interface to a scripting engine.
Input sequences are specified by means of a simple scripting language, which uses events as its basic building block, consisting of a target component identifier, the name of a method within the related component class, and optional string or numeric arguments.

Consider the following Model GUI Application:

Robot Manager:



The program is used to control four robots, as shown in the Robots list: RATS-1, RATS-2, RCAT-1 and POGO-1. Each robot can be online or offline, as indicated within the respective list entry. The user can enter commands in a text field; commands can be further attributed by selecting any of the three FORCED, QUEUED and SINGLE_STEP checkboxes. The nominal control procedure consists of the following steps:
1) entering a command
2) selecting command options (if any)
3) selecting a robot from the list
4) sending the command to the robot by means of clicking the Send button.
By checking the acquire status checkbox, the user can start a background thread that probes robots for their online status, and dynamically updates the related entries of the Robots list as each robot becomes online or offline.

We observe that once acquire status checkbox is checked, the status of the robots randomly keep changing every three seconds.

The code runs

acquireRobotStatus(val)

```
JCheckBox cb = new JCheckBox("acquire status");
cb.addItemListener( new ItemListener() {
  public void itemStateChanged (ItemEvent e) {
    boolean val = (e.getStateChange() == ItemEvent.SELECTED);
    Logger.log("set acquire status=", Boolean.valueOf(val));
    acquireRobotStatus(val);
  }
});
contentPane.add(cb);
```

which in turn calls

startAcquisitionThread();

```
void acquireRobotStatus (boolean b) {
  if (b) {
    model.startAcquisitionThread();
  } else {
    model.stopAcquisitionThread();
  }
}
```

which starts thread ***t***

```
public void startAcquisitionThread() {
  if (acquisitionThread == null) {
    RobotStatusAcquisitionThread t = new RobotStatusAcquisitionThread(this);
    acquisitionThread = t;
    t.start();
  }
```

The thread runs the following

```
public void run() {
  ListModel robotList = model.getListModel();

  int n = robotList.getSize();
  Random random = new Random(0);

  while (!done) {
    int idx = random.nextInt(n);
    Robot robot = (Robot)robotList.getElementAt(idx);
    model.setRobotOnline(robot, !model.isRobotOnline(robot.getId()));

    try {
      Thread.sleep(3000);
    } catch (InterruptedException ix) {}
  }
}
```

The ***setRobotOnline*** function randomly sets the robot to online or offline. One important observation we need to make is that the thread can set **ALL THE ROBOTS TO OFFLINE status**

```
public void setRobotOnline (Robot robot, boolean isOnline) {
  if (isOnline) {
    onlineRobots.put(robot.getId(), robot);
  } else {
    onlineRobots.remove(robot.getId());
  }
  listModel.changed(robot);
}
```

onlineRobots is a hashmap which matches the robot string name to the robot object **of only the online robots**. So now consider the case when the hashmap is empty and all robots are offline.

Now consider the spawning of another thread of type **EventDispatchThread** when the Send button in the application is clicked.

```
// the Send button
JButton b1 = new JButton("Send");
b1.addActionListener( new ActionListener() {
  public void actionPerformed (ActionEvent e) {
    Logger.log("Send pressed");
    sendSequence();
  }
});
contentPane.add(b1);
```

The ***sendSequence*** function is called.

```
void sendSequence () {
  // collect all the data (this could also be kept in fields
  String sequence = sequenceField.getText();

  if (selectedRobotName != null) {
    if (model.isRobotOnline(selectedRobotName)) {
      Robot robot = model.getOnlineRobot(selectedRobotName);
      String result = model.sendSequence(robot, sequence);
      resultLabel.setText(result);
    } else {
      resultLabel.setText("REJECT: " + selectedRobotName + " not online");
    }
  } else {
    resultLabel.setText("REJECT: no robot selected");
  }
 }
}
```

Now consider that once the if condition is entered ,i.e, there was atleast one robot online, the Data Acquisition Thread now sets all the robots to offline and *model.getOnlineRobot(selectedRobotName)* returns a null object.

The method ***model.sendSequence*** is shown below which returns ***processSequence(sequence)***

```
// this is the main purpose of the model
public String sendSequence(Robot robot, String sequence) {
        return robot.processSequence(sequence);
    }
```

here robot is a null object and **null pointer exception** is thrown.(Appendix C)

Hence a bug has been detected successfully. The following trace can be done by seeing the Console output in eclipse when JPF is run. This bug is very hard to detect. JPF comes to the rescue and gives a trace as to how this bug may be generated. It is evident that using any other testing tool to detect such bugs is impractical. We can now understand the importance of model checking and Formal Verification.

Now suppose we are given a specification that robot POGO cannot force queued sequences. This is an example of an application specific property. We can do the following

```
class POGO extends Robot {
  public POGO (String id) {
    super(id);
  }
  public String processSequence(String sequence) {
    //... do some processing
    assert !(Option.FORCED.isSet() && Option.QUEUED.isSet()) : "POGOs cannot force queued sequences";
    //... do a lot more processing
    return getResult(sequence);
  }
}
```

Therefore when the undesirable sequence is entered an error is asserted and JPF finds this error.(Appendix D)
Therefore an application specific bug has been detected. JPF provides a framework where we can set the system error specifications and check for them. This is an indispensable feature of JPF.

Now consider an extension of this application where we always have to keep the robot 'RATS-2' online. A thread ***SetRoboOneOnline*** is created to set this robot online always.

```
public void run() {
  ListModel robotList = model.getListModel();

  int n = robotList.getSize();
  Random random = new Random(0);

  while (!done) {
    int idx = random.nextInt(n);
    Robot robot = (Robot)robotList.getElementAt(1);
    model.setRobotOnline(robot, true);

    try {
      Thread.sleep(5000);
    } catch (InterruptedException ix) {}
  }
}
```

Now when we run the program we see that the ***RobotStartAcquisitionThread*** can actually change this to offline when it sets robots online and offline randomly. This is a **potential race condition**.

Although an unexpected situation occurs when JPF is run on this code.

```
JavaPathfinder core system v8.0 (rev 29+) - (C) 2005-2014 United States Government. All rights reserved.


====================================================== system under test
RobotManager.main()

====================================================== search started: 21/12/15 10:10 PM
[SEVERE] JPF out of memory

====================================================== search constraint
JPF out of memory

====================================================== snapshot
thread java.awt.EventDispatchThread:{id:1,name:AWT-EventQueue-0,status:RUNNING,priority:6,isDaemon:false,lockCount:0,suspendCount:0}
  call stack:
        at Robot.getResult(RobotManager.java:47)
        at RATS.processSequence(RobotManager.java:76)
        at RobotManager.sendSequence(RobotManager.java:312)
        at RobotManagerView.sendSequence(RobotManager.java:612)
        at RobotManagerView$3.actionPerformed(RobotManager.java:384)
        at javax.swing.AbstractButton.fireActionPerformed(AbstractButton.java:120)
        at javax.swing.AbstractButton$Handler.actionPerformed(AbstractButton.java:41)
        at javax.swing.DefaultButtonModel.fireActionPerformed(DefaultButtonModel.java:402)
        at javax.swing.DefaultButtonModel.setPressed(DefaultButtonModel.java:259)
        at javax.swing.AbstractButton.doClick(AbstractButton.java:195)
        at java.awt.EventDispatchThread.processNextEvent(EventDispatchThread.java)
        at java.awt.EventDispatchThread.run(EventDispatchThread.java:63)
```

We see that JPF is not able to detect the race condition or it has not yet found it before the memory is filled up due to state explosion when another thread is added. There is also a possibility of null pointer exception again when ***RobotStartAcquisitionThread*** sets 'RATS-2' to offline. But this also is not detected.

The results obtained is shown in Appendix(E).

Analysis and Problems faced:

First of all, starting out to install Java Path Finder is a tedious task. Once it is imported from the mercurial repository into Eclipse, lot of configurational changes have to be made. An Eclipse plugin for jpf also has to be installed. But FindBugs was easier to install and could instantly see it working in action.

Second, as the first phase of testing out non GUI codes for undesirable conditions like deadlocks, races, etc. was completed, exploration of JPF debugging for GUI applications was initiated. There is very little documentation online to read about JPF-AWT. It was a very difficult task to figure out how JPF worked for GUI applications.(nonetheless it was figured out)

A distanceCalculator.java GUI application was being built parallely to be evaluted using JPF-AWT. However, once a considerable amount was completed, we realized that it does not give proper results when JPF-AWT is applied as we needed to model the application differently including user input sequences, etc. This was not the initial perception as we assumed that JPF-AWT works for any kind of java code and has support for awt and swing libraries.

Finally, we relied on the example RobotManager.java GUI application and extended this application itself to check for different bugs and errors. We were successful in detecting null pointer exceptions and the application specific exceptions although the analysis was going out of bounds when checking for race conditions.

Contribution:

Raghavendra Sirigeri:

1. Installed JPF-CORE and JPF-AWT
2. Analysed the codes using JPF-Core
3. Findbugs tool on the suite of 4 codes
4. Worked on RobotManager Application and its analysis
5. Worked on modified RobotManager Application and its analysis

Ishaan Sayal:

1. distanceCalculator Java Application
2. Findbugs tool on the above code

Conclusion:

The suite of test codes have been analysed and tested successfully using JPF-Core as well as FindBugs tools and the RobotManager GUI Application also has been evaluated.
It was a great experience working on this project, which nonetheless was filled with many roadblocks. It took a week to install JPF and get it working successfully. Along the course of the project, we found that there is a dearth of model checking softwares to check GUI based applications. There is very less documentation online about usage of JPF(other than the JPF site) which is the only source. So, we actually plan to make a tutorial of its usage and probably host it online so that in the future, others can refer to it. There is no doubt that this project and the course has enabled us to quantify the advantages of Model Checking and Formal Verification.

References:

1. Java PathFinder. Website. http://babelfish.arc.nasa.gov/trac/jpf.
2. JPF-AWT: Model Checking GUI Applications. Peter Mehlitz, NASA, Ames Research Center,Moffett Field, CA, USA & Mateusz Ujma,Department of Computer Science,University of Oxford, UK
3. Model Checking Graphical User Interfaces using Abstractions. Matthew B. Dwyer, Vicki Carr, Laura Hines, Kansas State University
4. http://www.cs.umd.edu/~jfoster/papers/issre04.pdf
5. http://www.java2novice.com/java-interview-programs/thread-deadlock/

Appendix A

1. Rand.java

```
JavaPathfinder core system v8.0 (rev 29+) - (C) 2005-2014 United States Government.
All rights reserved.


====================================================== system under test
Rand.main()

====================================================== search started: 21/12/15
11:56 PM
computing c = a/(b+a - 2)..
a=0
  b=0       ,a=0
=>  c=0     , b=0, a=0
  b=1       ,a=0
=>  c=0     , b=1, a=0
  b=2       ,a=0

====================================================== error 1
gov.nasa.jpf.vm.NoUncaughtExceptionsProperty
java.lang.ArithmeticException: division by zero
       at Rand.main(Rand.java:34)


====================================================== trace #1
------------------------------------------------------ transition #0 thread: 0
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"ROOT" ,1/1,isCascaded:false}
      [3157 insn w/o sources]
  Rand.java:23                  : System.out.println("computing c = a/(b+a -
2)..");
      [2 insn w/o sources]
  Rand.java:24                  : Random random = new Random(42);      // (1)
      [2 insn w/o sources]
  Rand.java:24                  : Random random = new Random(42);      // (1)
  Rand.java:26                  : int a = random.nextInt(2);           // (2)
      [1 insn w/o sources]
------------------------------------------------------ transition #1 thread: 0
gov.nasa.jpf.vm.choice.IntIntervalGenerator[id="verifyGetInt(II)",isCascaded:false,
0..1,delta=+1,cur=0]
      [2 insn w/o sources]
  Rand.java:26                  : int a = random.nextInt(2);           // (2)
  Rand.java:27                  : System.out.printf("a=%d\n", a);
      [2 insn w/o sources]
  Rand.java:27                  : System.out.printf("a=%d\n", a);
      [2 insn w/o sources]
  Rand.java:27                  : System.out.printf("a=%d\n", a);
  Rand.java:31                  : int b = random.nextInt(3);           // (3)
      [1 insn w/o sources]
```

```
------------------------------------------------------ transition #2 thread: 0
gov.nasa.jpf.vm.choice.IntIntervalGenerator[id="verifyGetInt(II)",isCascaded:false,
0..2,delta=+1,cur=2]
      [2 insn w/o sources]
  Rand.java:31                    : int b = random.nextInt(3);           // (3)
  Rand.java:32                    : System.out.printf("  b=%d       ,a=%d\n", b, a);
      [2 insn w/o sources]
  Rand.java:32                    : System.out.printf("  b=%d       ,a=%d\n", b, a);
      [2 insn w/o sources]
  Rand.java:32                    : System.out.printf("  b=%d       ,a=%d\n", b, a);
      [2 insn w/o sources]
  Rand.java:32                    : System.out.printf("  b=%d       ,a=%d\n", b, a);
  Rand.java:34                    : int c = a/(b+a -2);                  // (4)

====================================================== results
error #1: gov.nasa.jpf.vm.NoUncaughtExceptionsProperty
"java.lang.ArithmeticException: division by zero  a..."

====================================================== statistics
elapsed time:       00:00:01
states:             new=4,visited=1,backtracked=2,end=2
search:             maxDepth=3,constraints=0
choice generators:  thread=1
(signal=0,lock=1,sharedRef=0,threadApi=0,reschedule=0), data=2
heap:               new=649,released=27,maxLive=615,gcCycles=4
instructions:       3354
max memory:         61MB
loaded code:        classes=65,methods=1362

====================================================== search finished: 21/12/15
11:56 PM
```

2. CheckRace.java

```
JavaPathfinder core system v8.0 (rev 29+) - (C) 2005-2014 United States Government.
All rights reserved.


====================================================== system under test
CheckRace.main()

====================================================== search started: 21/12/15
11:58 PM

====================================================== error 1
gov.nasa.jpf.listener.PreciseRaceDetector
race for field CheckRace.cnt
  Thread-1 at CheckRace.run(CheckRace.java:5)
             "cnt = y + 1;"  WRITE: putstatic CheckRace.cnt
  Thread-2 at CheckRace.run(CheckRace.java:4)
             "int y = cnt;"  READ:  getstatic CheckRace.cnt


====================================================== trace #1
```

```
------------------------------------------------------ transition #0 thread: 0
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"ROOT" ,1/1,isCascaded:false}
      [3157 insn w/o sources]
  CheckRace.java:2              : private static int cnt = 0; // shared state
  CheckRace.java:1              : public class CheckRace extends Thread {
      [1 insn w/o sources]
  CheckRace.java:8              : Thread t1 = new CheckRace();
  CheckRace.java:1              : public class CheckRace extends Thread {
      [145 insn w/o sources]
  CheckRace.java:1              : public class CheckRace extends Thread {
  CheckRace.java:8              : Thread t1 = new CheckRace();
  CheckRace.java:9              : Thread t2 = new CheckRace();
  CheckRace.java:1              : public class CheckRace extends Thread {
      [145 insn w/o sources]
  CheckRace.java:1              : public class CheckRace extends Thread {
  CheckRace.java:9              : Thread t2 = new CheckRace();
  CheckRace.java:10             : t1.start();
      [1 insn w/o sources]
------------------------------------------------------ transition #1 thread: 0
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"START" ,1/2,isCascaded:false}
      [2 insn w/o sources]
  CheckRace.java:11             : t2.start();
      [1 insn w/o sources]
------------------------------------------------------ transition #2 thread: 1
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"START" ,2/3,isCascaded:false}
      [1 insn w/o sources]
  CheckRace.java:1              : public class CheckRace extends Thread {
  CheckRace.java:4              : int y = cnt;
------------------------------------------------------ transition #3 thread: 0
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"SHARED_CLASS"
,1/3,isCascaded:false}
      [2 insn w/o sources]
  CheckRace.java:12             : }
  CheckRace.java:3              : public void run() {
------------------------------------------------------ transition #4 thread: 1
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"TERMINATE" ,1/2,isCascaded:false}
  CheckRace.java:4              : int y = cnt;
  CheckRace.java:5              : cnt = y + 1;
------------------------------------------------------ transition #5 thread: 2
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"SHARED_CLASS"
,2/2,isCascaded:false}
      [1 insn w/o sources]
  CheckRace.java:1              : public class CheckRace extends Thread {
  CheckRace.java:4              : int y = cnt;
------------------------------------------------------ transition #6 thread: 1
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"SHARED_CLASS"
,1/2,isCascaded:false}
  CheckRace.java:5              : cnt = y + 1;

====================================================== results
error #1: gov.nasa.jpf.listener.PreciseRaceDetector "race for field CheckRace.cnt
Thread-1 at CheckR..."

====================================================== statistics
```

```
elapsed time:       00:00:01
states:             new=11,visited=2,backtracked=6,end=2
search:             maxDepth=7,constraints=0
choice generators:  thread=10
(signal=0,lock=1,sharedRef=3,threadApi=2,reschedule=4), data=0
heap:               new=368,released=46,maxLive=360,gcCycles=9
instructions:       3538
max memory:         61MB
loaded code:        classes=62,methods=1472

============================================================ search finished: 21/12/15
11:58 PM
```

3.CheckNPE.java

```
JavaPathfinder core system v8.0 (rev 29+) - (C) 2005-2014 United States Government.
All rights reserved.


============================================================ system under test
CheckNPE.main()

============================================================ search started: 22/12/15
12:01 AM

============================================================ error 1
gov.nasa.jpf.vm.NoUncaughtExceptionsProperty
java.lang.NullPointerException: referencing field 'name' on null object
       at CheckNPE.main(CheckNPE.java:6)


============================================================ trace #1
------------------------------------------------------------ transition #0 thread: 0
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"ROOT" ,1/1,isCascaded:false}
      [3157 insn w/o sources]
  CheckNPE.java:5                : ResultList[] boll = new ResultList[5];
  CheckNPE.java:6                : boll[0].name = "iiii";

============================================================ results
error #1: gov.nasa.jpf.vm.NoUncaughtExceptionsProperty
"java.lang.NullPointerException: referencing field ..."

============================================================ statistics
elapsed time:       00:00:01
states:             new=1,visited=0,backtracked=0,end=0
search:             maxDepth=1,constraints=0
choice generators:  thread=1
(signal=0,lock=1,sharedRef=0,threadApi=0,reschedule=0), data=0
heap:               new=368,released=0,maxLive=0,gcCycles=0
instructions:       3164
max memory:         61MB
loaded code:        classes=65,methods=1351
```

```
====================================================== search finished: 22/12/15
12:01 AM
```

4.Deadlock.java

```
JavaPathfinder core system v8.0 (rev 29+) - (C) 2005-2014 United States Government.
All rights reserved.


====================================================== system under test
Deadlock.main()

====================================================== search started: 22/12/15
12:02 AM
JavaUNIX
JavaUNIX
JavaUNIX
JavaUNIX
JavaUNIX

====================================================== error 1
gov.nasa.jpf.vm.NotDeadlockedProperty
deadlock encountered:
  thread Deadlock$1:{id:1,name:My Thread
1,status:BLOCKED,priority:5,isDaemon:false,lockCount:0,suspendCount:0}
  thread Deadlock$2:{id:2,name:My Thread
2,status:BLOCKED,priority:5,isDaemon:false,lockCount:0,suspendCount:0}


====================================================== trace #1
------------------------------------------------------ transition #0 thread: 0
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"ROOT" ,1/1,isCascaded:false}
      [3157 insn w/o sources]
  Deadlock.java:31               : Deadlock mdl = new Deadlock();
  Deadlock.java:1                : public class Deadlock {
      [1 insn w/o sources]
  Deadlock.java:3                : String str1 = "Java";
  Deadlock.java:4                : String str2 = "UNIX";
  Deadlock.java:6                : Thread trd1 = new Thread("My Thread 1"){
  Deadlock.java:1                : public class Deadlock {
  Deadlock.java:6                : Thread trd1 = new Thread("My Thread 1"){
      [125 insn w/o sources]
  Deadlock.java:6                : Thread trd1 = new Thread("My Thread 1"){
  Deadlock.java:18               : Thread trd2 = new Thread("My Thread 2"){
  Deadlock.java:1                : public class Deadlock {
  Deadlock.java:18               : Thread trd2 = new Thread("My Thread 2"){
      [125 insn w/o sources]
  Deadlock.java:18               : Thread trd2 = new Thread("My Thread 2"){
  Deadlock.java:1                : public class Deadlock {
  Deadlock.java:31               : Deadlock mdl = new Deadlock();
  Deadlock.java:32               : mdl.trd1.start();
      [1 insn w/o sources]
------------------------------------------------------ transition #1 thread: 0
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"START" ,1/2,isCascaded:false}
```

```
      [2 insn w/o sources]
  Deadlock.java:33              : mdl.trd2.start();
      [1 insn w/o sources]
------------------------------------------------------ transition #2 thread: 0
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"START" ,1/3,isCascaded:false}
      [2 insn w/o sources]
  Deadlock.java:34              : }
  Deadlock.java:3               : String str1 = "Java";
------------------------------------------------------ transition #3 thread: 1
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"TERMINATE" ,1/2,isCascaded:false}
      [1 insn w/o sources]
  Deadlock.java:1               : public class Deadlock {
  Deadlock.java:9               : synchronized(str1){
------------------------------------------------------ transition #4 thread: 1
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"LOCK" ,1/2,isCascaded:false}
  Deadlock.java:9               : synchronized(str1){
  Deadlock.java:10              : synchronized(str2){
------------------------------------------------------ transition #5 thread: 1
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"LOCK" ,1/2,isCascaded:false}
  Deadlock.java:10              : synchronized(str2){
  Deadlock.java:11              : System.out.println(str1 + str2);
      [7 insn w/o sources]
  Deadlock.java:11              : System.out.println(str1 + str2);
      [2 insn w/o sources]
  Deadlock.java:11              : System.out.println(str1 + str2);
      [2 insn w/o sources]
  Deadlock.java:11              : System.out.println(str1 + str2);
      [2 insn w/o sources]
  Deadlock.java:11              : System.out.println(str1 + str2);
      [2 insn w/o sources]
  Deadlock.java:10              : synchronized(str2){
  Deadlock.java:9               : synchronized(str1){
------------------------------------------------------ transition #6 thread: 1
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"LOCK" ,1/2,isCascaded:false}
  Deadlock.java:9               : synchronized(str1){
  Deadlock.java:10              : synchronized(str2){
------------------------------------------------------ transition #7 thread: 2
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"LOCK" ,2/2,isCascaded:false}
      [1 insn w/o sources]
  Deadlock.java:1               : public class Deadlock {
  Deadlock.java:21              : synchronized(str2){
------------------------------------------------------ transition #8 thread: 1
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"SHARED_OBJECT"
,1/2,isCascaded:false}
  Deadlock.java:10              : synchronized(str2){
  Deadlock.java:11              : System.out.println(str1 + str2);
------------------------------------------------------ transition #9 thread: 1
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"SHARED_OBJECT"
,1/2,isCascaded:false}
  Deadlock.java:11              : System.out.println(str1 + str2);
      [7 insn w/o sources]
  Deadlock.java:11              : System.out.println(str1 + str2);
      [2 insn w/o sources]
  Deadlock.java:11              : System.out.println(str1 + str2);
```

```
------------------------------------------------------ transition #10 thread: 1
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"SHARED_OBJECT"
,1/2,isCascaded:false}
  Deadlock.java:11              : System.out.println(str1 + str2);
      [2 insn w/o sources]
  Deadlock.java:11              : System.out.println(str1 + str2);
      [2 insn w/o sources]
  Deadlock.java:11              : System.out.println(str1 + str2);
      [2 insn w/o sources]
  Deadlock.java:10              : synchronized(str2){
  Deadlock.java:9               : synchronized(str1){
------------------------------------------------------ transition #11 thread: 1
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"SHARED_OBJECT"
,1/2,isCascaded:false}
  Deadlock.java:9               : synchronized(str1){
------------------------------------------------------ transition #12 thread: 1
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"LOCK" ,1/2,isCascaded:false}
  Deadlock.java:9               : synchronized(str1){
  Deadlock.java:10              : synchronized(str2){
------------------------------------------------------ transition #13 thread: 2
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"SHARED_OBJECT"
,2/2,isCascaded:false}
  Deadlock.java:21              : synchronized(str2){
------------------------------------------------------ transition #14 thread: 1
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"LOCK" ,1/2,isCascaded:false}
  Deadlock.java:10              : synchronized(str2){
------------------------------------------------------ transition #15 thread: 1
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"LOCK" ,1/2,isCascaded:false}
  Deadlock.java:10              : synchronized(str2){
  Deadlock.java:11              : System.out.println(str1 + str2);
      [7 insn w/o sources]
  Deadlock.java:11              : System.out.println(str1 + str2);
      [2 insn w/o sources]
  Deadlock.java:11              : System.out.println(str1 + str2);
      [2 insn w/o sources]
  Deadlock.java:11              : System.out.println(str1 + str2);
      [2 insn w/o sources]
  Deadlock.java:11              : System.out.println(str1 + str2);
      [2 insn w/o sources]
  Deadlock.java:10              : synchronized(str2){
------------------------------------------------------ transition #16 thread: 1
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"RELEASE" ,1/2,isCascaded:false}
  Deadlock.java:10              : synchronized(str2){
  Deadlock.java:9               : synchronized(str1){
------------------------------------------------------ transition #17 thread: 1
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"SHARED_OBJECT"
,1/2,isCascaded:false}
  Deadlock.java:9               : synchronized(str1){
------------------------------------------------------ transition #18 thread: 1
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"LOCK" ,1/2,isCascaded:false}
  Deadlock.java:9               : synchronized(str1){
  Deadlock.java:10              : synchronized(str2){
------------------------------------------------------ transition #19 thread: 1
```

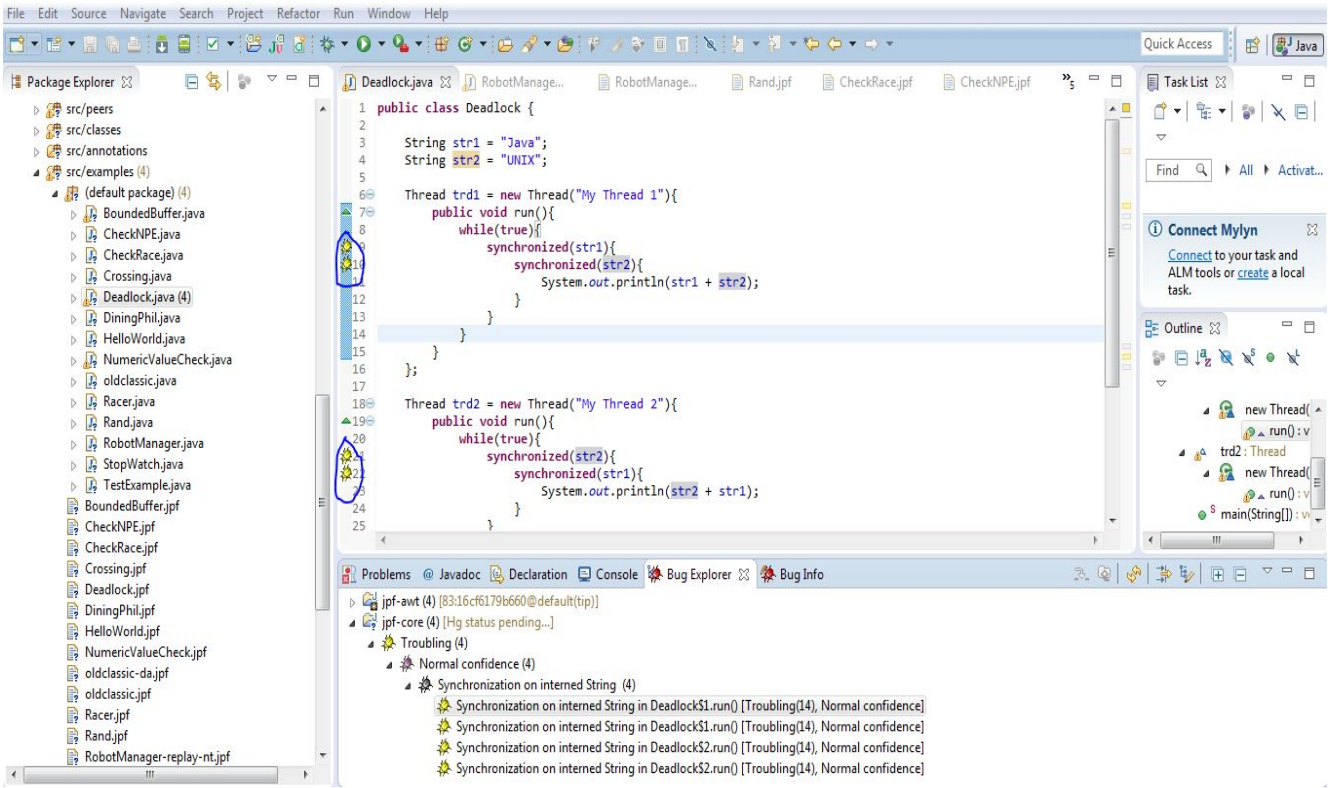
```
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"SHARED_OBJECT"
,1/2,isCascaded:false}
  Deadlock.java:10                : synchronized(str2){
------------------------------------------------------ transition #20 thread: 2
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"LOCK" ,2/2,isCascaded:false}
  Deadlock.java:21                : synchronized(str2){
  Deadlock.java:22                : synchronized(str1){

====================================================== results
error #1: gov.nasa.jpf.vm.NotDeadlockedProperty "deadlock encountered:    thread
Deadlock$1:{id:1,n..."

====================================================== statistics
elapsed time:       00:00:01
states:             new=21,visited=3,backtracked=3,end=1
search:             maxDepth=21,constraints=0
choice generators:  thread=21
(signal=0,lock=11,sharedRef=7,threadApi=2,reschedule=1), data=0
heap:               new=393,released=29,maxLive=366,gcCycles=24
instructions:       3726
max memory:         61MB
loaded code:        classes=64,methods=1474

====================================================== search finished: 22/12/15
12:02 AM
```

Appendix B



Appendix C

NULL POINTER EXCEPTION IN RobotManager:

```
JavaPathfinder core system v8.0 (rev 29+) - (C) 2005-2014 United States Government.
All rights reserved.


====================================================== system under test
RobotManager.main()

====================================================== search started: 22/12/15
12:12 AM

====================================================== snapshot #1
thread
java.awt.EventDispatchThread:{id:1,name:AWT-EventQueue-0,status:RUNNING,priority:6,
isDaemon:false,lockCount:0,suspendCount:0}
  call stack:
       at RobotManager.sendSequence(RobotManager.java:269)
       at RobotManagerView.sendSequence(RobotManager.java:569)
       at RobotManagerView$3.actionPerformed(RobotManager.java:341)
       at javax.swing.AbstractButton.fireActionPerformed(AbstractButton.java:120)
       at
javax.swing.AbstractButton$Handler.actionPerformed(AbstractButton.java:41)
       at
javax.swing.DefaultButtonModel.fireActionPerformed(DefaultButtonModel.java:402)
       at javax.swing.DefaultButtonModel.setPressed(DefaultButtonModel.java:259)
       at javax.swing.AbstractButton.doClick(AbstractButton.java:195)
       at java.awt.EventDispatchThread.processNextEvent(EventDispatchThread.java)
       at java.awt.EventDispatchThread.run(EventDispatchThread.java:63)

thread
RobotStatusAcquisitionThread:{id:2,name:Thread-2,status:RUNNING,priority:5,isDaemon
:true,lockCount:0,suspendCount:0}
  call stack:
       at java.util.HashMap.removeNode(HashMap.java:844)
       at java.util.HashMap.remove(HashMap.java:798)
       at RobotManager.setRobotOnline(RobotManager.java:246)
       at RobotStatusAcquisitionThread.run(RobotManager.java:149)


====================================================== error 1
gov.nasa.jpf.vm.NoUncaughtExceptionsProperty
java.lang.NullPointerException: Calling
'processSequence(Ljava/lang/String;)Ljava/lang/String;' on null object
       at RobotManager.sendSequence(RobotManager.java:269)
       at RobotManagerView.sendSequence(RobotManager.java:569)
       at RobotManagerView$3.actionPerformed(RobotManager.java:341)
       at
javax.swing.AbstractButton.fireActionPerformed(javax/swing/AbstractButton.java:120)
       at
javax.swing.AbstractButton$Handler.actionPerformed(javax/swing/AbstractButton.java:
41)
```

```
	at
javax.swing.DefaultButtonModel.fireActionPerformed(javax/swing/DefaultButtonModel.j
ava:402)
	at
javax.swing.DefaultButtonModel.setPressed(javax/swing/DefaultButtonModel.java:259)
	at javax.swing.AbstractButton.doClick(javax/swing/AbstractButton.java:195)
	at
java.awt.EventDispatchThread.processNextEvent(gov.nasa.jpf.awt.JPF_java_awt_EventDi
spatchThread)
	at java.awt.EventDispatchThread.run(java/awt/EventDispatchThread.java:63)


====================================================== statistics
elapsed time:       00:00:38
states:             new=8231,visited=3055,backtracked=11177,end=2671
search:             maxDepth=1364,constraints=0
choice generators:  thread=5424
(signal=0,lock=45,sharedRef=4026,threadApi=130,reschedule=1223), data=136
heap:               new=41095,released=40356,maxLive=3347,gcCycles=11156
instructions:       1095541
max memory:         393MB
loaded code:        classes=404,methods=5995

====================================================== search finished: 22/12/15
12:12 AM
```

Appendix D

Application Specific Error

```
JavaPathfinder core system v8.0 (rev 29+) - (C) 2005-2014 United States Government.
All rights reserved.


====================================================== system under test
RobotManager.main()

====================================================== search started: 22/12/15
12:16 AM

====================================================== error 1
gov.nasa.jpf.vm.NoUncaughtExceptionsProperty
java.lang.AssertionError: POGOs cannot force queued sequences
	at POGO.processSequence(RobotManager.java:93)
	at RobotManager.sendSequence(RobotManager.java:269)
	at RobotManagerView.sendSequence(RobotManager.java:569)
	at RobotManagerView$3.actionPerformed(RobotManager.java:341)
	at
javax.swing.AbstractButton.fireActionPerformed(javax/swing/AbstractButton.java:120)
```

```
	at
javax.swing.AbstractButton$Handler.actionPerformed(javax/swing/AbstractButton.java:
41)
	at
javax.swing.DefaultButtonModel.fireActionPerformed(javax/swing/DefaultButtonModel.j
ava:402)
	at
javax.swing.DefaultButtonModel.setPressed(javax/swing/DefaultButtonModel.java:259)
	at javax.swing.AbstractButton.doClick(javax/swing/AbstractButton.java:195)
	at
java.awt.EventDispatchThread.processNextEvent(gov.nasa.jpf.awt.JPF_java_awt_EventDi
spatchThread)
	at java.awt.EventDispatchThread.run(java/awt/EventDispatchThread.java:63)
```

Appendix E

RobotManager Extension with extra thread

```
JavaPathfinder core system v8.0 (rev 29+) - (C) 2005-2014 United States Government.
All rights reserved.


====================================================== system under test
RobotManager.main()

====================================================== search started: 21/12/15
10:10 PM
[SEVERE] JPF out of memory

====================================================== search constraint
JPF out of memory

====================================================== snapshot
thread
java.awt.EventDispatchThread:{id:1,name:AWT-EventQueue-0,status:RUNNING,priority:6,
isDaemon:false,lockCount:0,suspendCount:0}
  call stack:
	at Robot.getResult(RobotManager.java:47)
	at RATS.processSequence(RobotManager.java:76)
	at RobotManager.sendSequence(RobotManager.java:312)
	at RobotManagerView.sendSequence(RobotManager.java:612)
	at RobotManagerView$3.actionPerformed(RobotManager.java:384)
	at javax.swing.AbstractButton.fireActionPerformed(AbstractButton.java:120)
	at
javax.swing.AbstractButton$Handler.actionPerformed(AbstractButton.java:41)
	at
javax.swing.DefaultButtonModel.fireActionPerformed(DefaultButtonModel.java:402)
	at javax.swing.DefaultButtonModel.setPressed(DefaultButtonModel.java:259)
	at javax.swing.AbstractButton.doClick(AbstractButton.java:195)
	at java.awt.EventDispatchThread.processNextEvent(EventDispatchThread.java)
	at java.awt.EventDispatchThread.run(EventDispatchThread.java:63)
```

```
thread
RobotStatusAcquisitionThread:{id:2,name:Thread-2,status:RUNNING,priority:5,isDaemon
:true,lockCount:0,suspendCount:0}
  call stack:
       at java.util.ArrayList.indexOf(ArrayList.java:316)
       at RobotManager$ListModel.changed(RobotManager.java:215)
       at RobotManager.setRobotOnline(RobotManager.java:291)
       at RobotStatusAcquisitionThread.run(RobotManager.java:149)

thread
SetRoboOneOnline:{id:3,name:Thread-3,status:RUNNING,priority:5,isDaemon:true,lockCo
unt:0,suspendCount:0}
  call stack:
       at java.util.ArrayList.indexOf(ArrayList.java:317)
       at RobotManager$ListModel.changed(RobotManager.java:215)
       at RobotManager.setRobotOnline(RobotManager.java:291)
       at SetRoboOneOnline.run(RobotManager.java:180)


====================================================== statistics
elapsed time:       00:37:08
states:             new=91650,visited=2056,backtracked=48048,end=45017
search:             maxDepth=45658,constraints=1
choice generators:  thread=45100
(signal=0,lock=47,sharedRef=35447,threadApi=1535,reschedule=8071), data=1534
heap:               new=544593,released=631290,maxLive=3356,gcCycles=92167
instructions:       15619795
max memory:         910MB
loaded code:        classes=401,methods=5968

====================================================== search finished: 21/12/15
10:47 PM
```