Report on JPF

# Introduction

Java path finder is a model checking framework for java. This can be used as a standalone software, eclips or netbeans plugins or can be run with command line. I have used the command lines to run the model checks. Before that, I used the jpf-core source to build the necessary .jar files for the command. I used Windows Powershell to run al lthe commands.

# Folder Structure

This section explains the folder structures.

1. Apps: This folder contains the apps that were used to run the model checking with jpf.
2. jpf-core: This folder contains the jpf core cloned from the jpf-core github repository.  
   i. bin: Contains the executables

ii. build: Contains the generated .jar files for jpf.

1. output: Some outputs were too large for the powershell so, for a better understanding I created text files containing the outputs of the apps while running model checking with jpf.
2. Screenshots: This folder contains all the screenshots of the project.
3. Root:  
   i. Report.docx: This is the Microsoft Word file that contains the report.

ii. README.md: This is the mark down version of the report to be read by github repository (For a simplified view).

iii. Report.pdf: pdf version of this report is also available here in this file.

# Steps for model checking using JPF

The following steps were executed to run the simple java application.

1. Clone JPS-core repository: I cloned the jpf-core repository inside my project folder from the jpf-core github repository by running the following command:

git clone https://github.com/javapathfinder/jpf-core.git

1. Build Jpf-core: Then, I navigated into the folder “jpf-core”. Now, I opened powershell and ran the following command to use gradle for building the .jar files:

./gradlew   
This build the necessary .jar files for the jpf-core. Fig. 1 shows the output of the command.

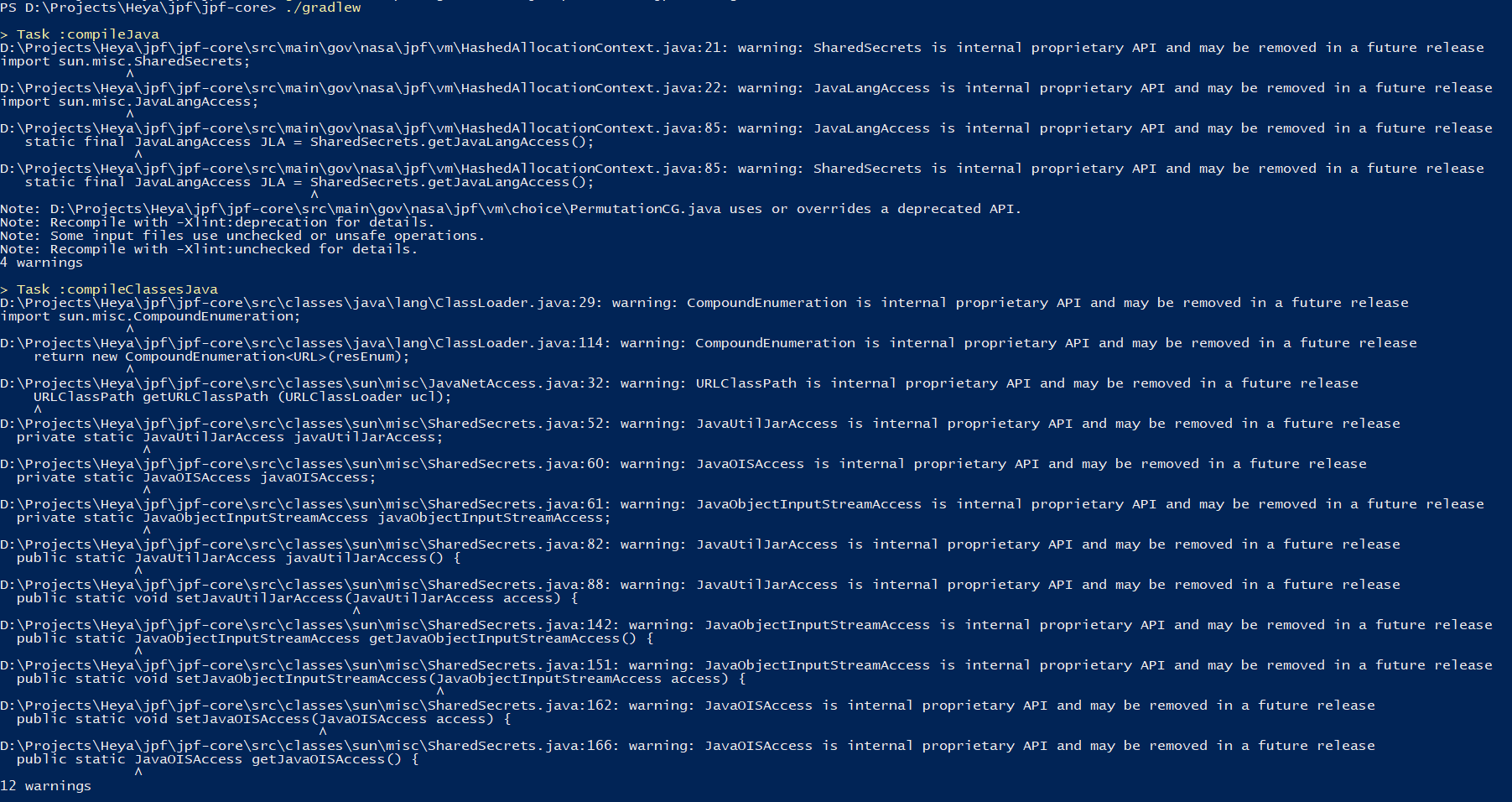
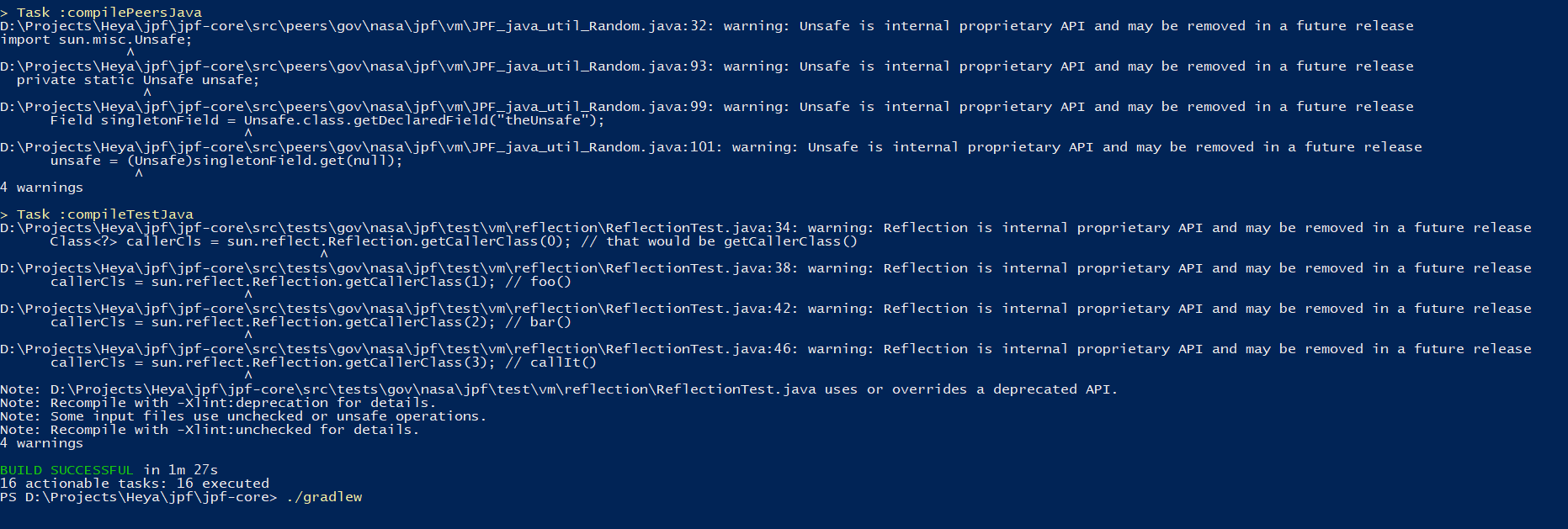
 

Fig 1: Building necessary .jar files

I also ran the following command to build again and this time I stored the outputs in a text file in the output folder mentioned in the folder structure.

./gradlew > ../output/jpfbuild.txt

1. Creating example java application “Rand.java”: I created the following files in the project folder.

Rand.java**:**

import java.util.Random;

public class Rand {

public static void main (String[] args) {

System.out.println("computing c = a/(b+a - 2)..");

Random random = new Random(42); // (1)

int a = random.nextInt(2); // (2)

System.out.printf("a=%d\n", a);

//... lots of code here

int b = random.nextInt(3); // (3)

System.out.printf(" b=%d ,a=%d\n", b, a);

int c = a/(b+a -2); // (4)

System.out.printf("=> c=%d , b=%d, a=%d\n", c, b, a);

}

}

Rand.jpf**:**  
  
target = Rand

cg.enumerate\_random = true

report.console.property\_violation=error,trace

## Running the model check:

I ran the following command to run the model check.

../jpf-core/bin/jpf +cg.enumerate\_random=true Rand

The output of the model check can be seen the screenshot in figure 2. I also wanted to store the output of the command in a file for a better understanding so, I again ran the following command.

../jpf-core/bin/jpf +cg.enumerate\_random=true Rand > ..\output\randoutput.txt

This created a randoutput.txt file with the powershell outputs.

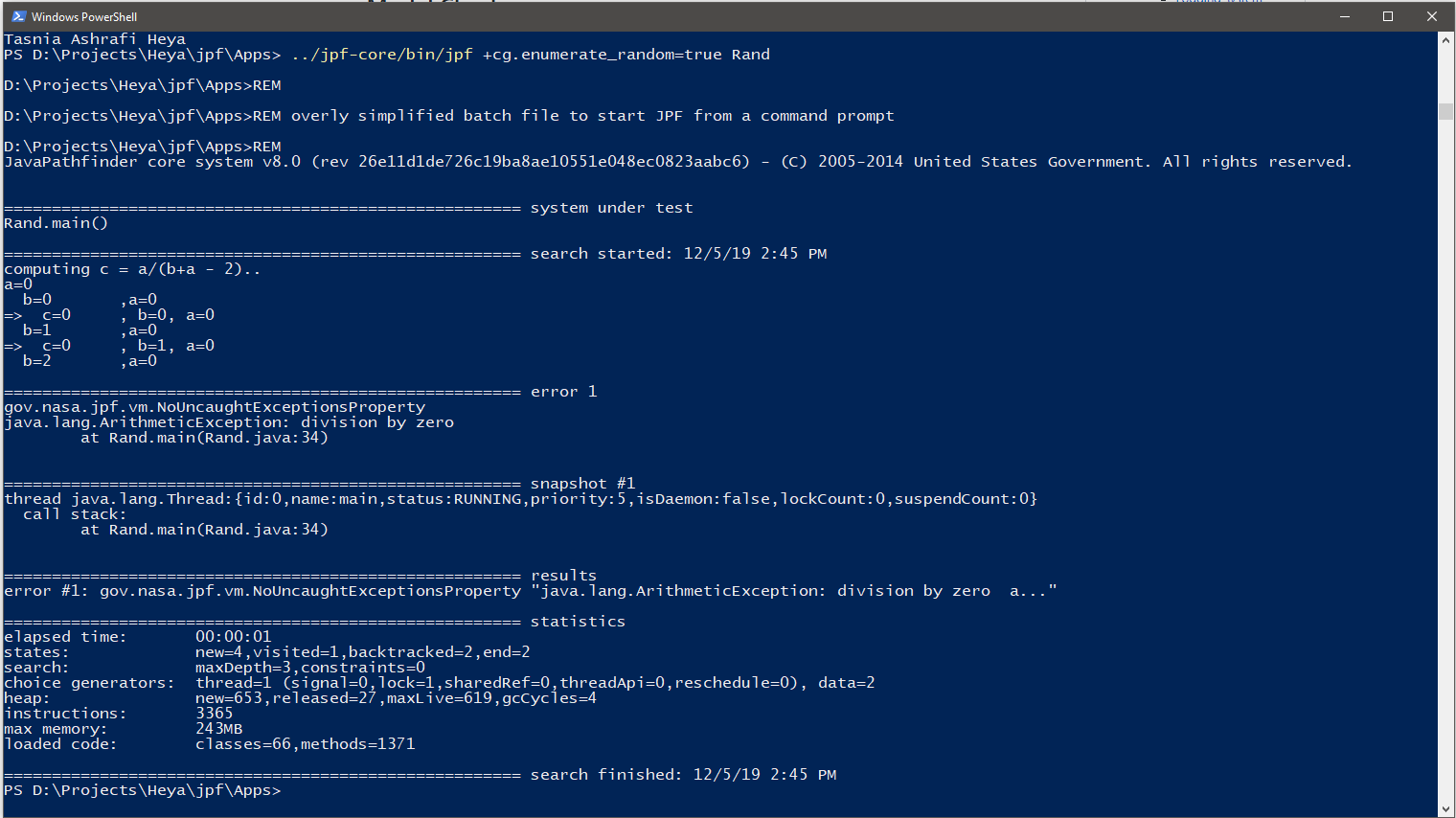


Figure 2: Running model check on Rand.java

## Example2 App:

I also ran another example. The source code of the file is large to put here. It is under “Apps” folder with the name “RobotManager.java”. A related jpf file was also created “RobotManager.jpf”.

## Example2 App model check:

I ran the second example with the following command.

../jpf-core/bin/jpf +cg.enumerate\_random=true RobotManager

The output of the command can be observed in figure 3. I also ran the following command to store the output in a text file by running the following command.

../jpf-core/bin/jpf +cg.enumerate\_random=true RobotManager > ..\output\robotmanageroutput.txt

The output file is available in the “output” folder mentioned in the folder structure. The name of the output file is “robotmanageroutput.txt”.

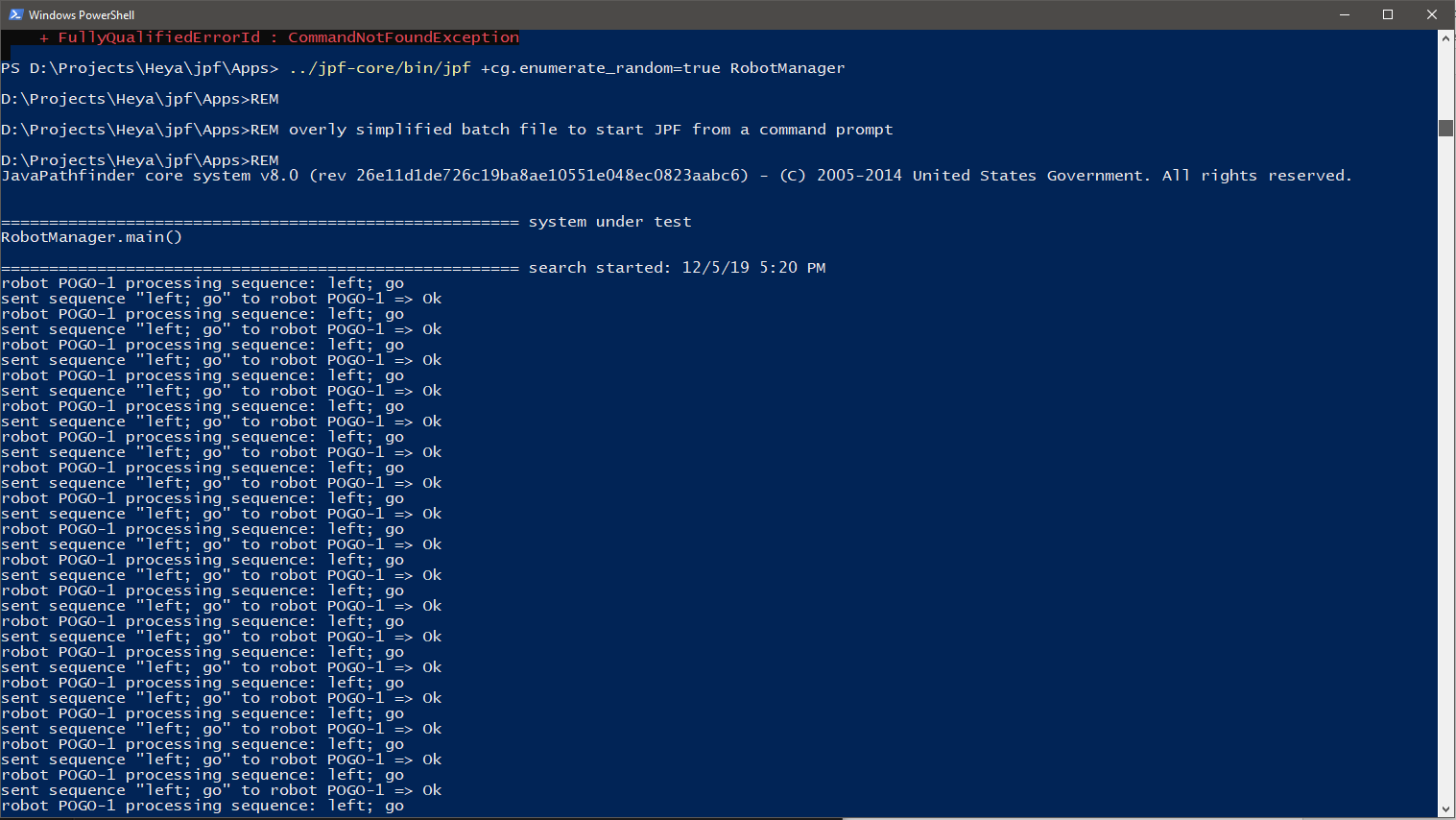
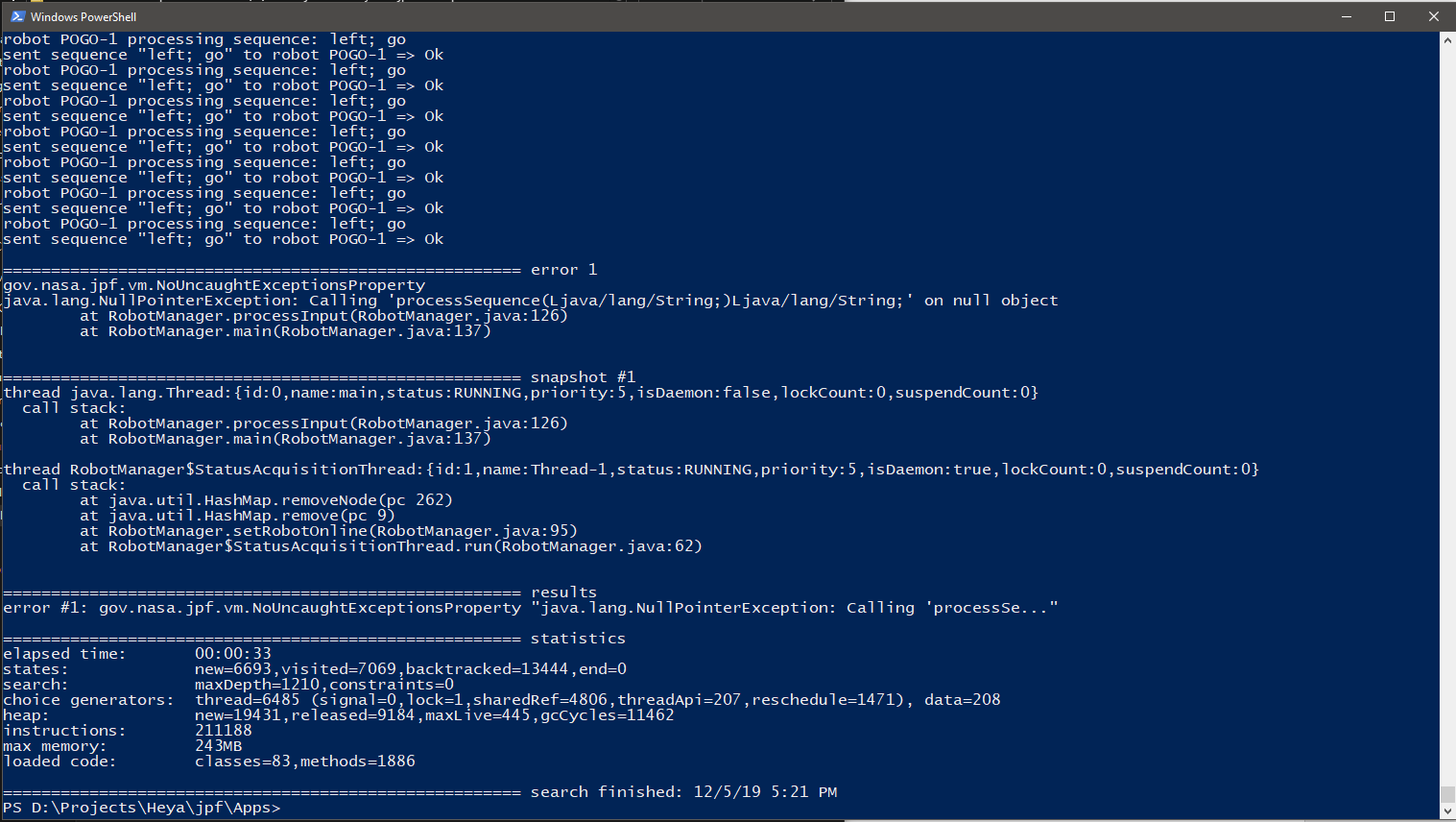


Figure 3: RobotManager model checking with jpf.