Mini Models

EECS 4315

www.cse.yorku.ca/course/4315/

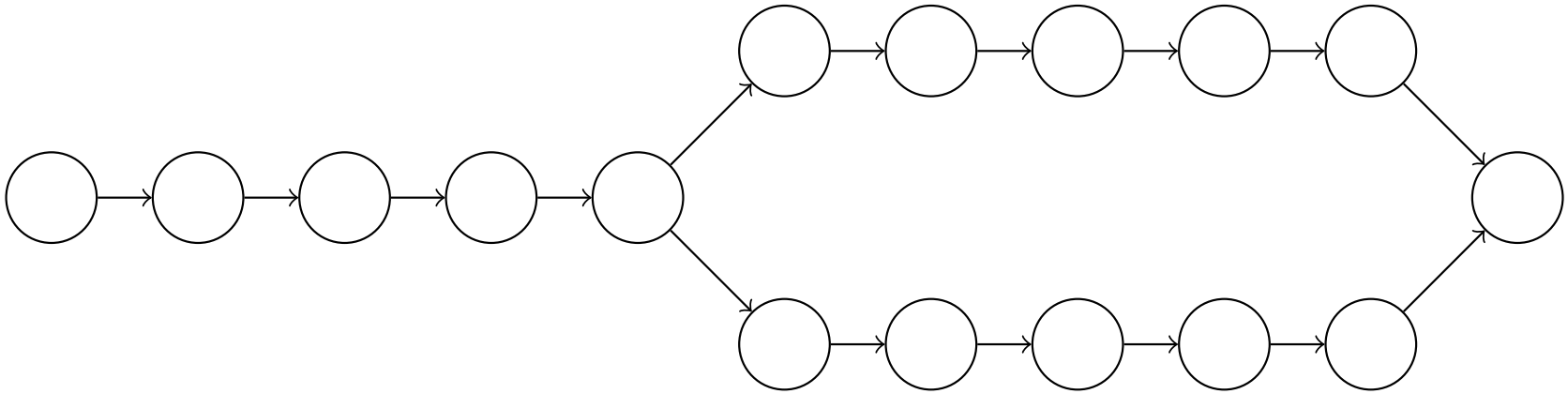
Mini

source: Keld Gydum

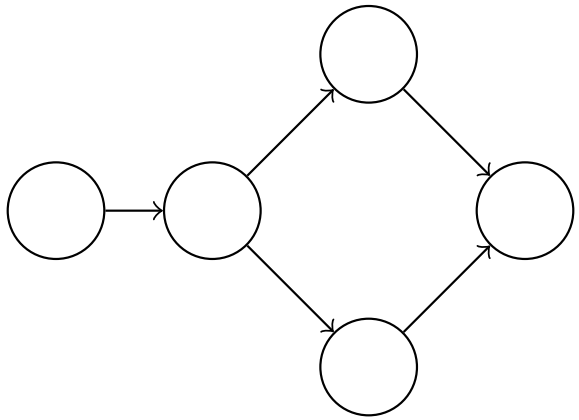
Mini Model

source: Mike Bird

Model



Mini Model

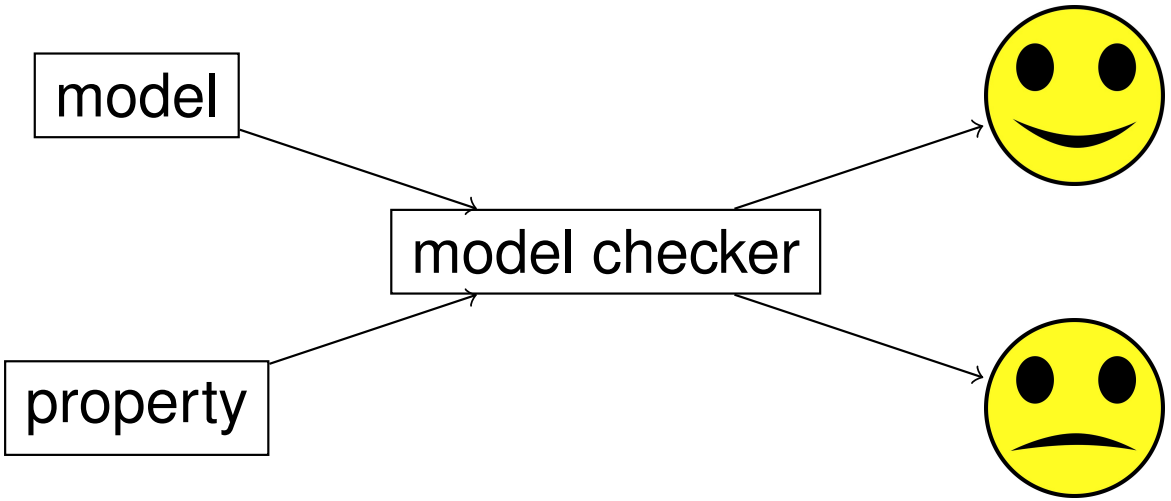


Introduction to Java PathFinder

EECS 4315

www.cse.yorku.ca/course/4315/

Model Checker



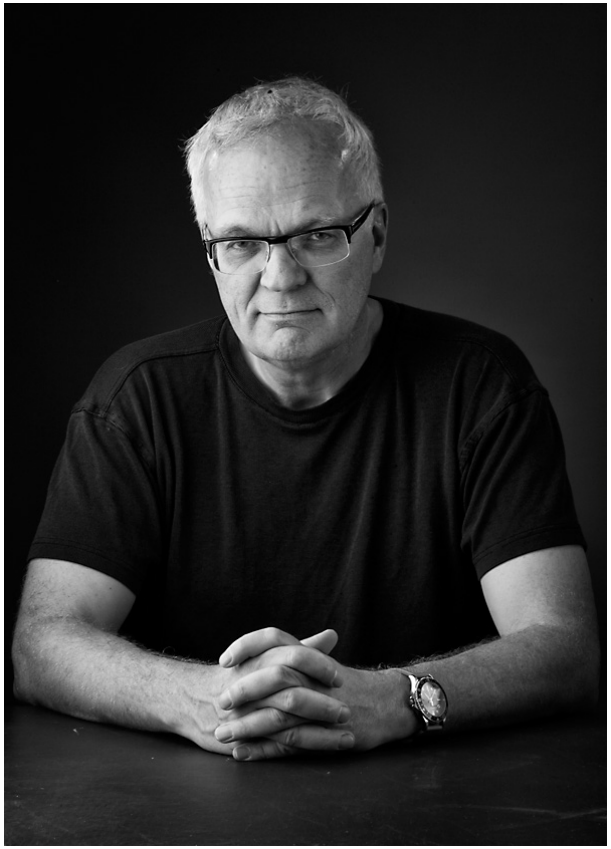
Some History

In 1999, Klaus Havelund introduced Java PathFinder (JPF).

Klaus Havelund. Java PathFinder – A Translator from Java to Promela. In, Dennis Dams, Rob Gerth, Stefan Leue and Mieke Massink, editors, *Proceedings of the 5th and 6th International SPIN Workshops*, volume 1680 of *Lecture Notes in Computer Science*, page 152. Springer-Verlag.

Klaus Havelund

- PhD in Computer Science from the University of Copenhagen.
- Senior Research Scientist at NASA's Jet Propulsion Laboratory.
- ASE 2014 most influential paper award.



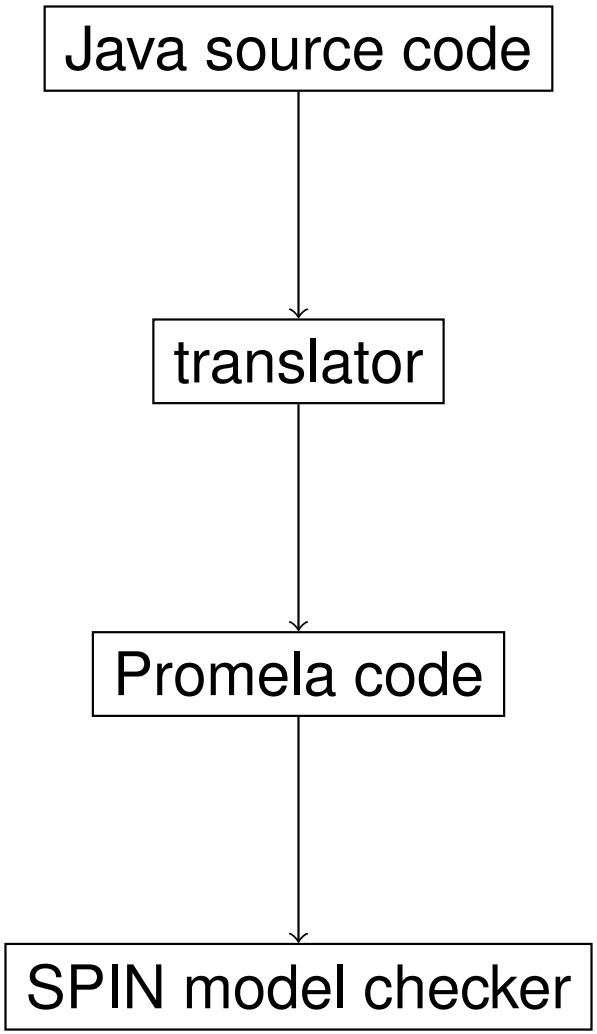
Source: Klaus Havelund

Some History

Others who initially worked on JPF:

- Michael Lowry (NASA)
- John Penix (NASA, now Google)
- Thomas Pressburger (NASA)
- Jens Ulrik Skakkebaek (Stanford, now Google)
- Willem Visser (NASA, now Stellenbosch University)

First Version of JPF

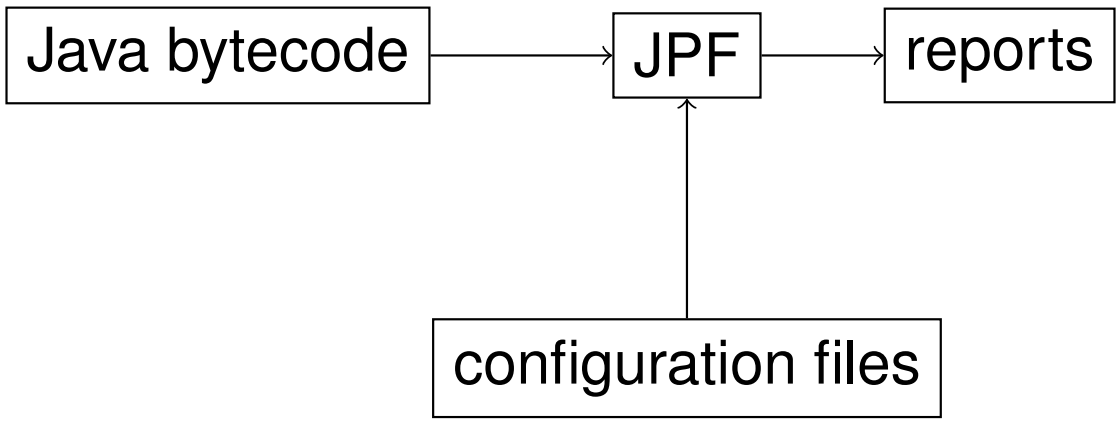


First Version of JPF

Major limitations:

- Representing all features of Java in Promela is impossible;
- Mapping bugs found by SPIN in the Promela code back to the Java code is challenging.

Second Version of JPF



The second version of JPF is a Java virtual machine (JVM).

Second Version of JPF

Willem Visser, Klaus Havelund, Guillaume Brat, Seungjoon Park. Model Checking Programs. In *Proceedings of the 15th IEEE International Conference on Automated Software Engineering*, pages 3–12, Grenoble, France, September 2000. IEEE

The Automated Software Engineering conference series has a rich history of good contributions to the area of research and development. The ASE most influential paper award is an effort to identity the most influential ASE paper 14 years after being published. In 2014, the above paper won this award.

Simple Example

```
import java.util.Random;

public class PrintRandom
{
  public static void main(String[] args)
  {
    Random random = new Random();
    final int MAX = 9;
    System.out.println(random.nextInt(MAX + 1));
  }
}
```

Simple Example

```
target=PrintRandom
classpath=.
```

Simple Example

```
JavaPathfinder core system v8.0 (rev 2+) - (C) 2005

=============================================================
PrintRandom.main()

=============================================================
0

=============================================================
no errors detected

=============================================================
elapsed time:       00:00:00
states:             new=1,visited=0,backtracked=1,e
...

=============================================================
```

Simple Example

Question

To how many different executions may the Java code give rise?

Simple Example

Question

To how many different executions may the Java code give rise?

Answer

10.

Simple Example

Question

To how many different executions may the Java code give rise?

Answer

10.

Question

How many different executions does JPF check?

Simple Example

Question

To how many different executions may the Java code give rise?

Answer

10.

Question

How many different executions does JPF check?

Answer

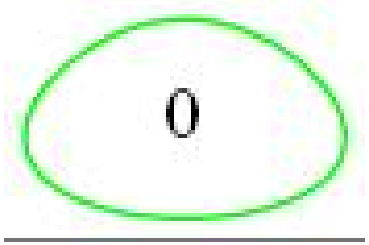
1.

Simple Example

Let's have a look at the state space diagram.

```
target=PrintRandom
classpath=.
listener=gov.nasa.jpf.listener.StateSpaceDot
```

Simple Example



Simple Example

Configure JPF so that it explores all random choices.

Simple Example

Configure JPF so that it explores all random choices.

```
target=PrintRandom
classpath=.
cg.enumerate_random=true
listener=gov.nasa.jpf.listener.StateSpaceDot
```

Simple Example

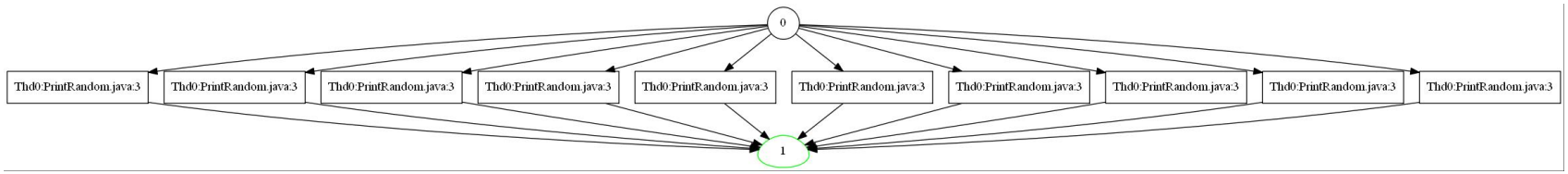
```
JavaPathfinder core system v8.0 (rev 2+) - (C) 2005

====================================================
PrintRandom.main()

====================================================
0
1
2
3
4
5
6
7
8
9
```

Simple Example

Let's have a look at the state space diagram.

```
target=PrintRandom
classpath=.
cg.enumerate_random=true
listener=gov.nasa.jpf.listener.StateSpaceDot
```

Simple Example



The ByteTest Revisited

In Lab 1, we wrote a JUnit test case to test the Byte class.

- JPF can only be run on apps, that is, classes that contain a main method.
- By default JPF checks for uncaught exceptions.

The ByteTest Revisited

```
package quiz;

import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class RunTest
{
  public static void main(String[] args)
    throws Throwable
  {
    Result result =
    JUnitCore.runClasses(ByteTest.class);
    for (Failure failure : result.getFailures())
    {
      throw failure.getException();
    }
```

The ByteTest Revisited

```
target=quiz.RunTest
classpath=.;/software/jars/junit-4.11.jar;\
/software/jars/hamcrest-core-1.3.jar
cg.enumerate_random=true
```

- `target` contains both the class name and the package name.
- The JUnit jars need to be added to the `classpath`.

The ByteTest Revisited

By default, JPF stops after detecting a bug.

The ByteTest Revisited

By default, JPF stops after detecting a bug.

To find multiple bugs . . .

```
target=quiz.RunTest
classpath=.;/software/jars/junit-4.11.jar;\
/software/jars/hamcrest-core-1.3.jar
cg.enumerate_random=true
search.multiple_errors=true
```