An Introduction to NASA's Java Pathfinder

Andrew V. Jones
⟨andrewj@doc.ic.ac.uk⟩

Department of Computing
Imperial College London

February, 2010

Deadlock 2: The Return of Deadlock

Thread 1

```
synchronized ( A ) {
    synchronized ( B ) { ... }
}
```

Thread 2

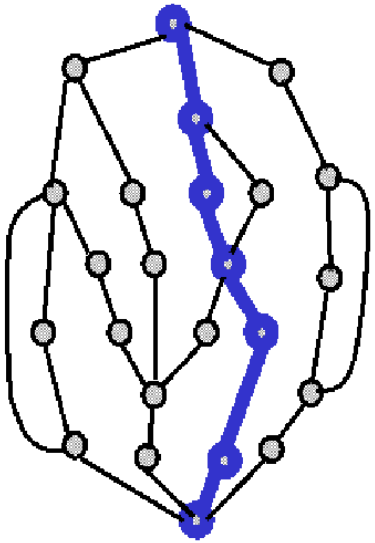
```
synchronized ( B ) {
    synchronized ( A ) {  ...  }
}
```

Introducing Formal Methods

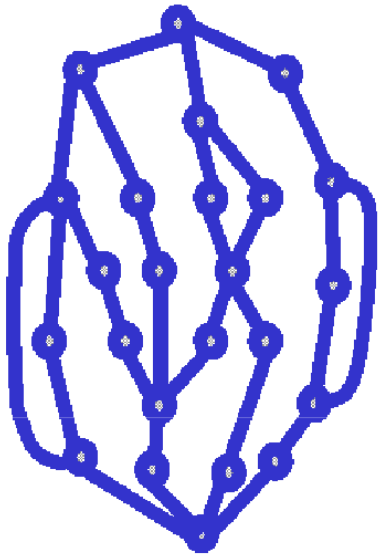
Generally, an approach to reasoning about a system (possibly code) to determine if it's correct.

What you've covered in your "Reasoning about Programs" course can be seen as "formal methods".

Model Checking vs. Testing

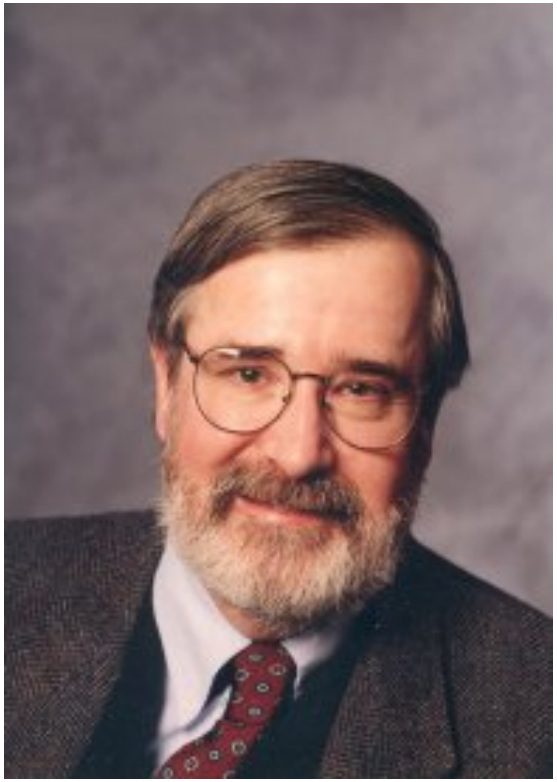


Testing explores a *single* path.



Model checking explores *all* paths.

Model Checking Is Big Business



Edmund M. Clarke

Awarded the 2008 Turing Prize for development of Model Checking!

Java Pathfinder

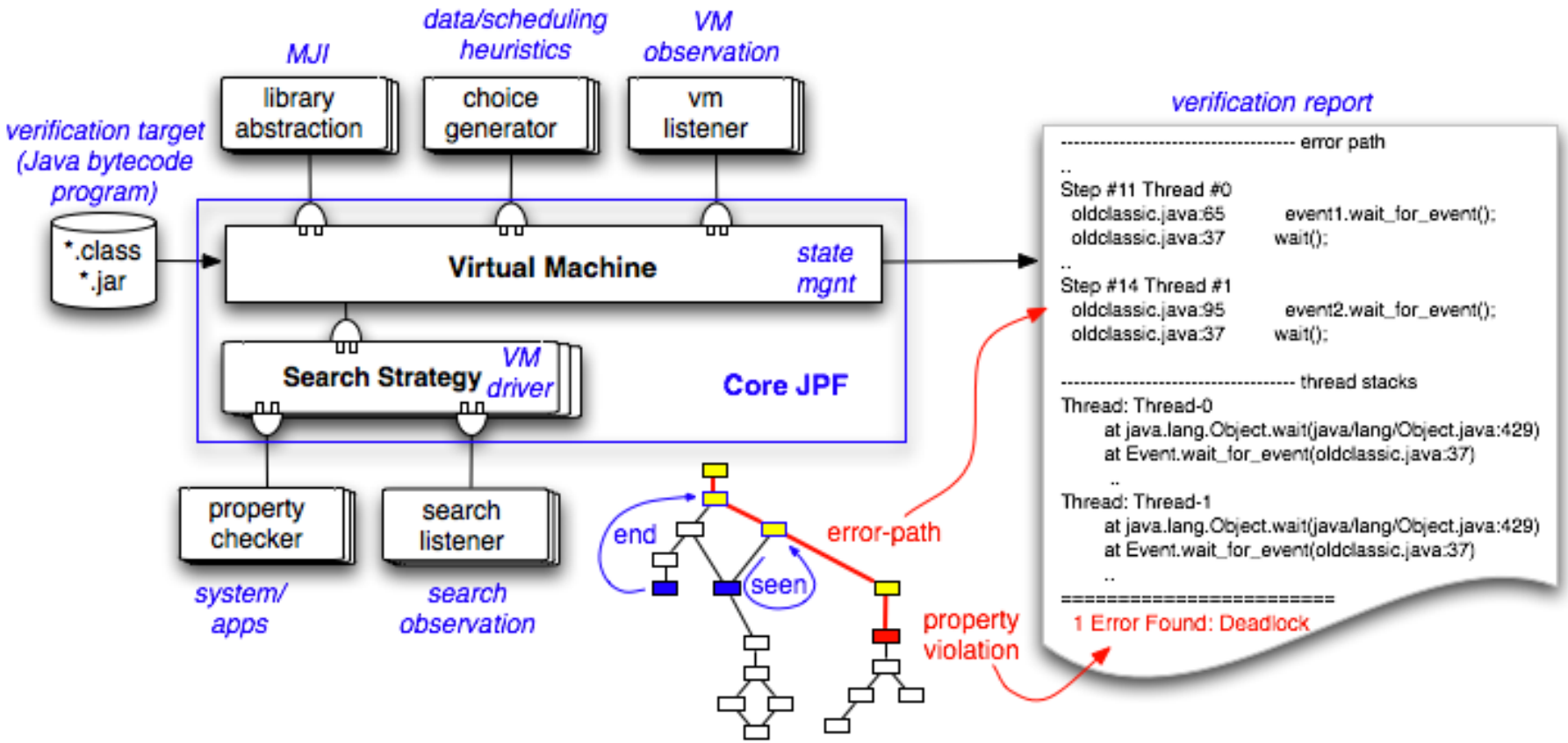
Java Pathfinder (JPF) is an *explicit state* "model checker" for Java.

It's developed by NASA at their Ames Research Center (one of the active developers did her Ph.D. in DSE, with the current HoD).

It can be used to *systemically* verify if a program satisfies a given property.

Historically, JPF was originally a translator of Java to PROMELA – the input language for the model checker SPIN.

Current Architecture



Taken from `http://javapathfinder.sourceforge.net`

A JPF "State"

Information about each thread (i.e., the current stack frame).

The current static, and dynamic, fields of classes (including locks).

For the curious: this is stored in an `int` array.

Using JPF

Given that JPF is itself a JVM, it can be used as a "drop-in" replacement for the `$ java <classname>`.

The easiest way to "run" JPF is specify a `.jpf` file corresponding to your class.

- This file contains the properties you wish to check.

```
$ java -jar path/to/jpf/RunJPF.jar\
    +classpath=. class.jpf
```

How JPF Searches

1. Start the program from the start (i.e., Line 1 of `main(...)`)
2. Execute series of bytecode until a new state is reached
3. Check the properties
   - On an error – stop.
4. See if we've already visited that state
   - If *yes* – backtrack
   - If *no* – visit successors

Built-in Properties

- Deadlock
- Unhandled Exceptions
- Java assertions (`assert(x == y)`)
- . . . and a lot more!

Implementing Properties

Listeners are the preferred way to interact with JPF.

```
public class PropertyListenerAdapter
    extends GenericProperty
    implements SearchListener, VMListener
    {
        public boolean check(Search search, JVM vm)
        {
            return true;
        }
    }
```

Search Listeners

Basic idea: Observer pattern

- Listeners are notified about events in the VM
  - E.g., when the Java class performs a specific operation
- Listeners can interact with the executing code
  - E.g., to get additional information such as field values

Example: `VMListener` class

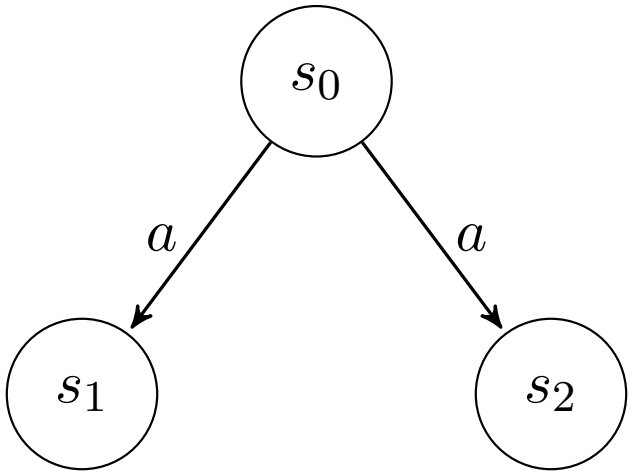
Can monitor the execution of Java bytecode instructions

- `void executeInstruction (JVM vm);`
- Can be used to inspect method calls and field assignments

Example – Specifying our own property

Detecting possible race conditions

Modelling Non-Determinism



Non-deterministic Choice

Using JPF's Verify Package

```
import gov.nasa.jpf.jvm.Verify;

public class VerifyExample
{
    public static void main(String[] args)
    {
        int i = Verify.getInt(-1, 1);
        System.out.println("i = " + i);
    }
}
```

Using JPF `Verify` Package

```
int i = 0;
while (i < 2)
   i++;
assert(i == 2);
```

Using JPF Verify Package

```
int i = 0; int j = 0;
boolean cond = Verify.getBoolean();
if (cond)
   j = 1;
assert(i == j);
```

Advanced JPF Topics

Partial Order Reduction.

Compositional Reasoning (Assume-Guarantee Reasoning).

If you found any of this interesting, I *thoroughly* recommend Alessio Lomuscio's (`alessio@doc`) 3rd year "Software Engineering – Systems Verification" course.

Thank you for coming – any questions?