Lecture 17 — Model Checking

David J. Pearce

School of Engineering and Computer Science
Victoria University of Wellington

Model Checking

In computer science, **model checking** *or property checking refers to the following problem: Given a model of a system, exhaustively and automatically check whether this model meets a given specification.*

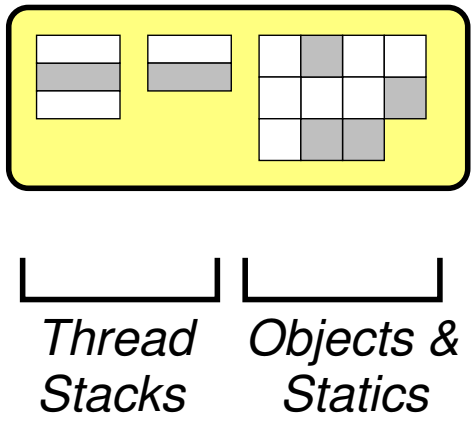
–Wikipedia

Java PathFinder: Overview

> *"JPF ... the swiss army knife of Java verification"*
>
> *–JPFWiki*

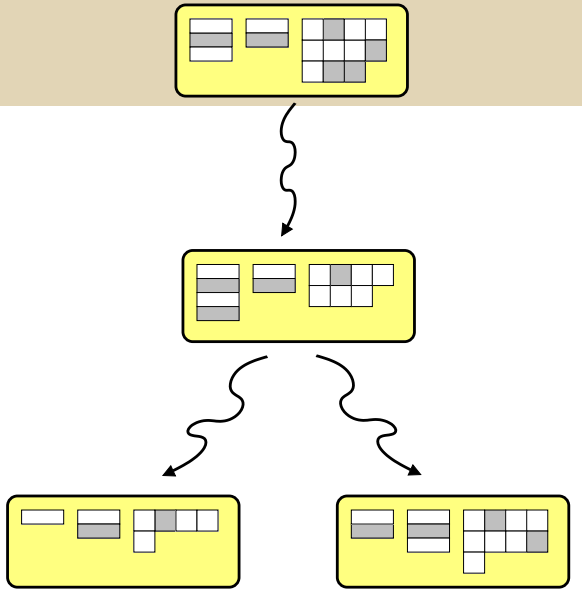
- Supports **explicit-state** and **symbolic model** checking
- Developed by NASA and used to find bugs in their software
- Provides custom **Java Virtual Machine** for executing bytecodes
- JPF VM explores **different execution paths** through a program
- **See:** `https://github.com/javapathfinder`

Java PathFinder: States



- JPF State consists of **three components**:
  1. Call stack for each thread
  2. Static (i.e. global) variables
  3. Dynamic variables (i.e. objects)
- Various mechanisms used for **state compression**

Java PathFinder: Execution



- Execution proceeds as for normal JVM with **concrete values**
- When **choice point** reached, **state is forked** ...

  ... and **both** are executed!

- If same state reached again, execution **terminated**.

Java PathFinder: Observations

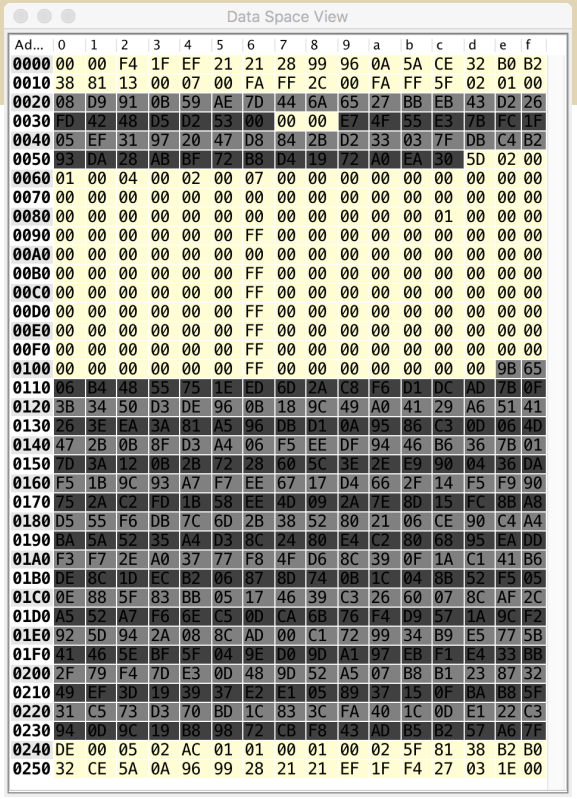
"We believe that applying model checking by itself to programs will not scale to programs of much more than **10k lines**. *The avenue we are pursuing is to augment model checking with information gathered from other techniques in order to handle large programs. Specifically, we are investigating the use of abstract interpretation,* **static analysis** *and runtime analysis to allow more efficient model checking of Java programs."*

–Visser, Havelund, Brat & Park

Exercise

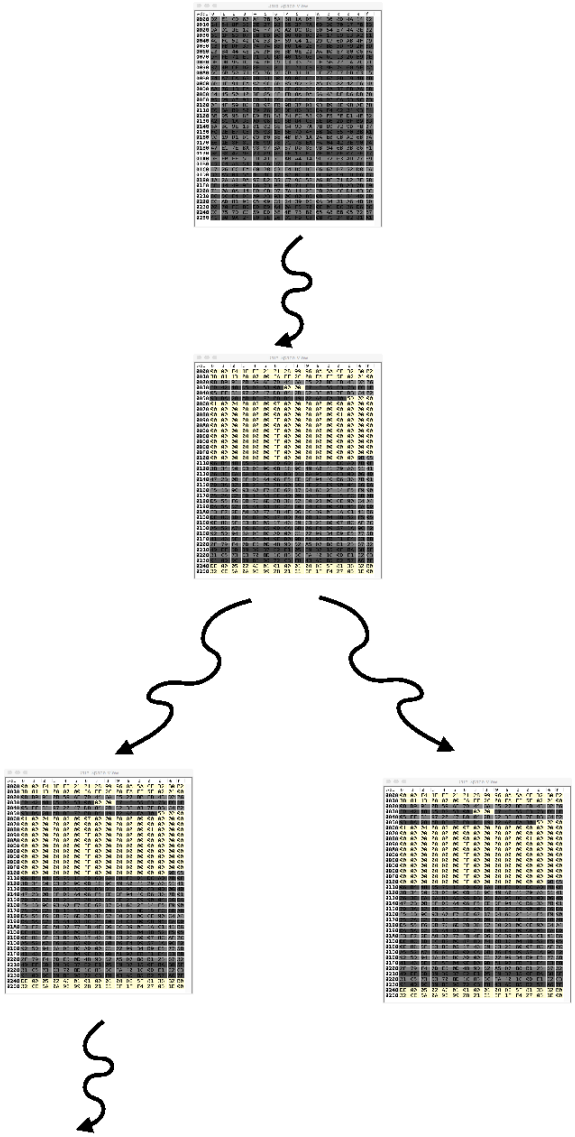
Let's build our own explicit-state model checker!

AVR Model Checker: Overview

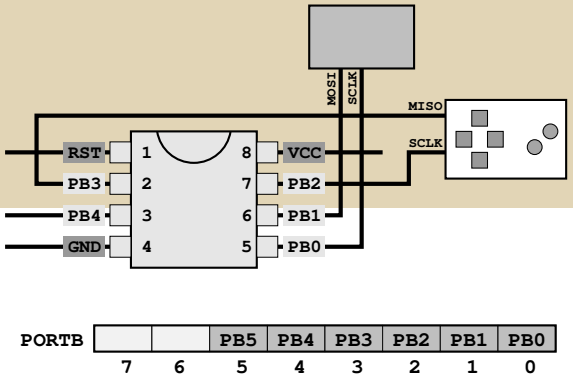


- Each **state** models 608 bytes of data
- **Execute** from initial `rjmp` instruction
- Memory locations initially given **don't care** values
- Current state **cloned** at choice points...
- ... and clone executed **after** current one finished.

AVR Model Checker: Execution



AVR Model Checker: Choice Points



Don't Care

A don't care value is simply a byte of data whose value is *unknown*. They can be used to model uninitialised memory and I/O ports.

- What causes **choice points** in our system?
- Choices occur whenever "don't care" value's are **used**.
- At choice point, must **concretize** don't care value (somehow).
- On ATtiny85, don't care values correspond to value of **input pins**.

AVR Model Checker: Choice Points (Cont'd)

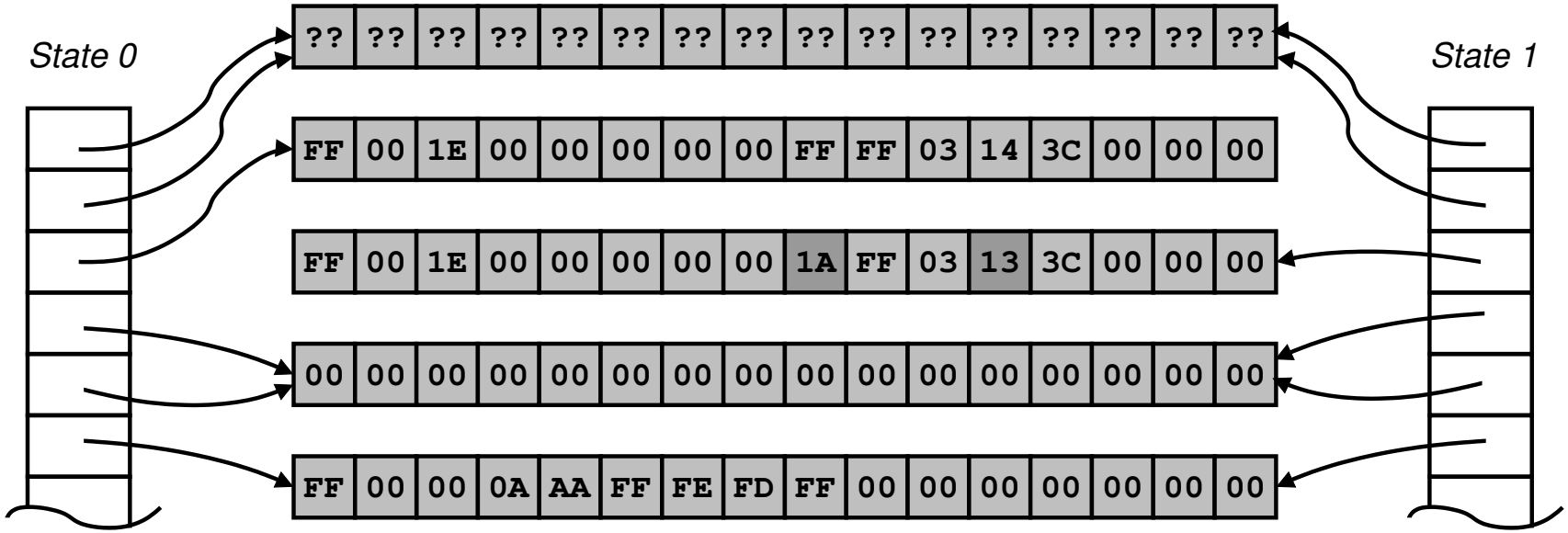
```
...
int buttons = read_buttons();
//
if (buttons & BUTTON_LEFT) { dir = WEST; }
else if (buttons & BUTTON_RIGHT) { dir = EAST; }
else if (buttons & BUTTON_UP) { dir = NORTH; }
else if (buttons & BUTTON_DOWN) { dir = SOUTH; }
else {
  return;
}
...
```

- Only **four choice** points in `snake.hex`!
- **Don't care** value propagates from `read_buttons()`...
- ... but it doesn't **live for long** though!

AVR Model Checker: State Compression

- 608 bytes per state is **not much**, but still compressable.
- Much of state might be **unused** for given program.
- Differences between states typically only **a few bytes changed**.
- **Simple strategy**:
  - Represent state as array of **16byte lines**.
  - Many lines **unchanged** between states, allowing **reuse**.
  - This gives good **compression**!

AVR Model Checker: State Compression



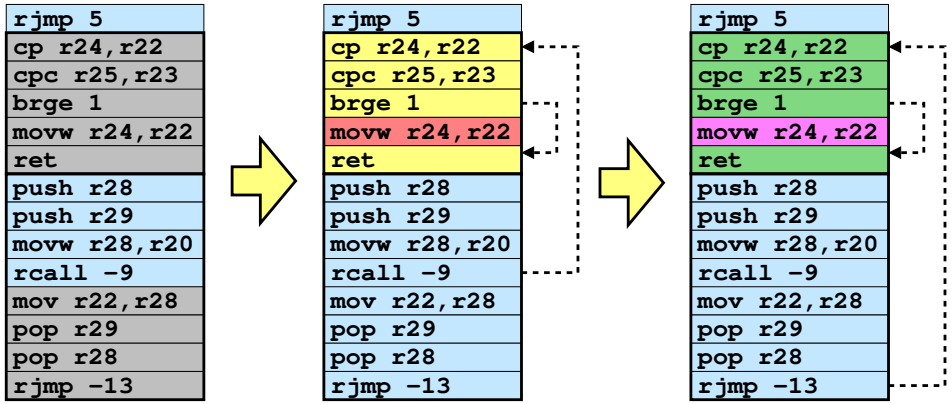
- Illustrates lines from two **related** states.
- Can see **slight difference** exists between third line. Assume all other lines **reused** between the two states.
- Memory required for *State 1* is **152 bytes** (for line array) plus **16 bytes** (for changed line).

AVR Model Checker: Bounded Execution

```
while(1) {
  // Time for the next frame!
  // Check for change of direction
  updateDirection();
  // Move snake in current direction
  moveSnake();
  ...
}
```

- What about programs that **loop forever?**
- For simple programs, may **exhaust** unique states anyway.
- Otherwise, **limit** how often state can visit same instruction ...
- ... and, when limit reached, **terminate** state.

AVR Model Checker: Comparison against CFA



- Explicit state model checking simliar to CFA, but ...
- ... choice points mean much **larger state space** to explore.
- ... and **more memory** required to store states.
- ... but, some aspects **easier to handle** (e.g. indirect jumps)

Further Reading

- **Model Checking Programs**, W. Visser, K. Havelund, G. Brat, S. Park. In *ASE*, 2003.
- **Model Checking Real Time Java Using Java PathFinder**, G. Lindstrom1, P. C. Mehlitz, and W. Visser. In *ATVA*, 2005.
- **Model checking of software for microcontrollers**, B. Schlich. In *TECS*, 2010.
- **Model checking C source code for embedded systems**, B Schlich and S. Kowalewski. In *STTT*, 2009.
- **CBMC — C Bounded Model Checker**, Daniel Kroening and Michael Tautschnig. In *TACAS*, 2014.