

# 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

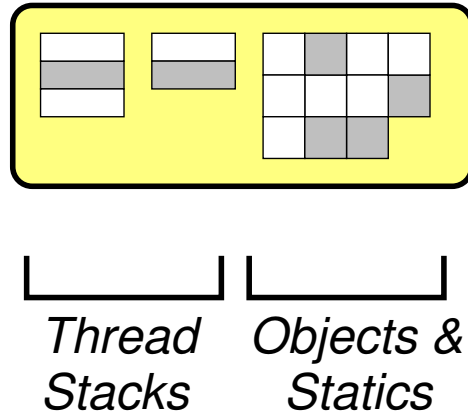


*“JPF ... the swiss army knife of Java verification”*

*–JPFWiki*

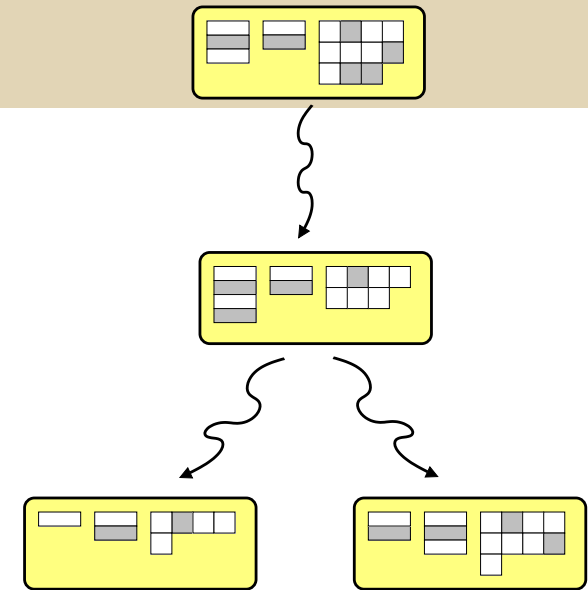
- Supports **explicit-state** and **symbolic model** checking
- Developed by NASA and used to find bugs in their software
- Provides custom **Java Virtual Machine** for executing bytecodes
- JPF VM explores **different execution paths** through a program
- **See:** <https://github.com/javapathfinder>

# 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

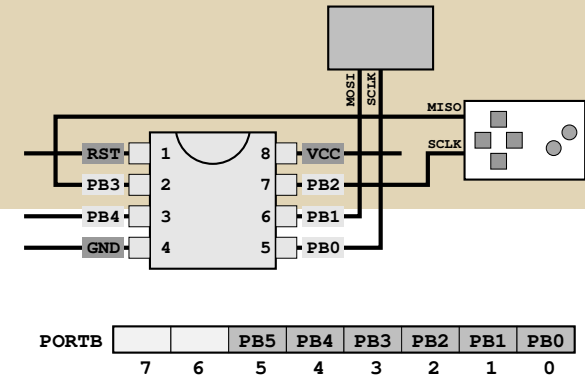
Ad...	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0000	00	00	F4	1F	EF	21	21	28	99	96	0A	5A	CE	32	B0	B2
0010	38	81	13	00	07	00	FA	FF	2C	00	FA	FF	5F	02	01	00
0020	08	D9	91	0B	59	AE	7D	44	6A	65	27	BB	EB	43	D2	26
0030	FD	42	48	D5	D2	53	00	00	00	00	00	00	00	00	00	00
0040	05	EF	31	97	20	47	D8	84	2B	D2	33	03	7F	DB	C4	B2
0050	03	DA	28	AB	BF	72	B8	D4	19	72	A0	EA	30	5D	02	00
0060	01	00	04	00	02	00	07	00	00	00	00	00	00	00	00	00
0070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0080	00	00	00	00	00	00	00	00	00	00	00	00	01	00	00	00
0090	00	00	00	00	00	00	FF	00	00	00	00	00	00	00	00	00
00A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00B0	00	00	00	00	00	00	FF	00	00	00	00	00	00	00	00	00
00C0	00	00	00	00	00	00	FF	00	00	00	00	00	00	00	00	00
00D0	00	00	00	00	00	00	FF	00	00	00	00	00	00	00	00	00
00E0	00	00	00	00	00	00	FF	00	00	00	00	00	00	00	00	00
00F0	00	00	00	00	00	00	FF	00	00	00	00	00	00	00	00	00
0100	00	00	00	00	00	00	FF	00	00	00	00	00	00	00	9B	65
0110	06	84	48	55	75	1E	ED	6D	2A	68	F6	D1	DC	AD	7B	0F
0120	3B	34	50	D3	DE	96	0B	18	9C	49	A0	41	29	A6	51	41
0130	26	3E	EA	3A	81	A5	96	DB	D1	0A	95	86	C3	0D	06	4D
0140	47	2B	0B	8F	D3	A4	06	F5	EE	DF	94	46	B6	36	7B	01
0150	7D	3A	12	0B	2B	72	28	60	5C	3E	2E	E9	90	04	36	DA
0160	F5	1B	9C	93	A7	F7	EE	67	17	D4	66	2F	14	F5	F9	00
0170	75	2A	C2	FD	1B	58	EE	4D	09	2A	7E	8D	15	FC	8B	A8
0180	D5	55	F6	DB	7C	6D	2B	38	52	80	21	06	CE	90	C4	A4
0190	BA	5A	52	35	A4	D3	8C	24	80	E4	C2	80	68	95	EA	DD
01A0	F3	F7	2E	A0	37	77	F8	4F	D6	8C	39	0F	1A	C1	41	B6
01B0	DE	8C	1D	EC	B2	06	87	8D	74	0B	1C	04	8B	52	F5	05
01C0	0E	88	5F	83	BB	05	17	46	39	C3	26	60	07	8C	AF	2C
01D0	A5	52	A7	F6	6E	C5	0D	CA	6B	76	F4	D9	57	1A	9C	F2
01E0	92	5D	94	2A	08	8C	AD	00	C1	72	99	34	B9	E5	77	5B
01F0	41	46	5E	8F	5F	04	9E	D0	9D	A1	97	EB	F1	E4	33	BB
0200	2F	79	F4	7D	E3	0D	48	9D	52	A5	07	B8	B1	23	87	32
0210	49	EF	3D	19	39	37	E2	E1	05	89	37	15	0F	5A	88	5F
0220	31	C5	73	D3	70	BD	1C	83	3C	FA	40	1C	0D	E1	22	C3
0230	04	0D	9C	19	B8	98	72	CB	F8	43	AD	B5	B2	57	A6	7F
0240	DE	00	05	02	AC	01	01	00	01	00	02	5F	81	38	B2	B0
0250	32	CE	5A	0A	96	99	28	21	21	EF	1F	F4	27	03	1E	00

- 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: 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)

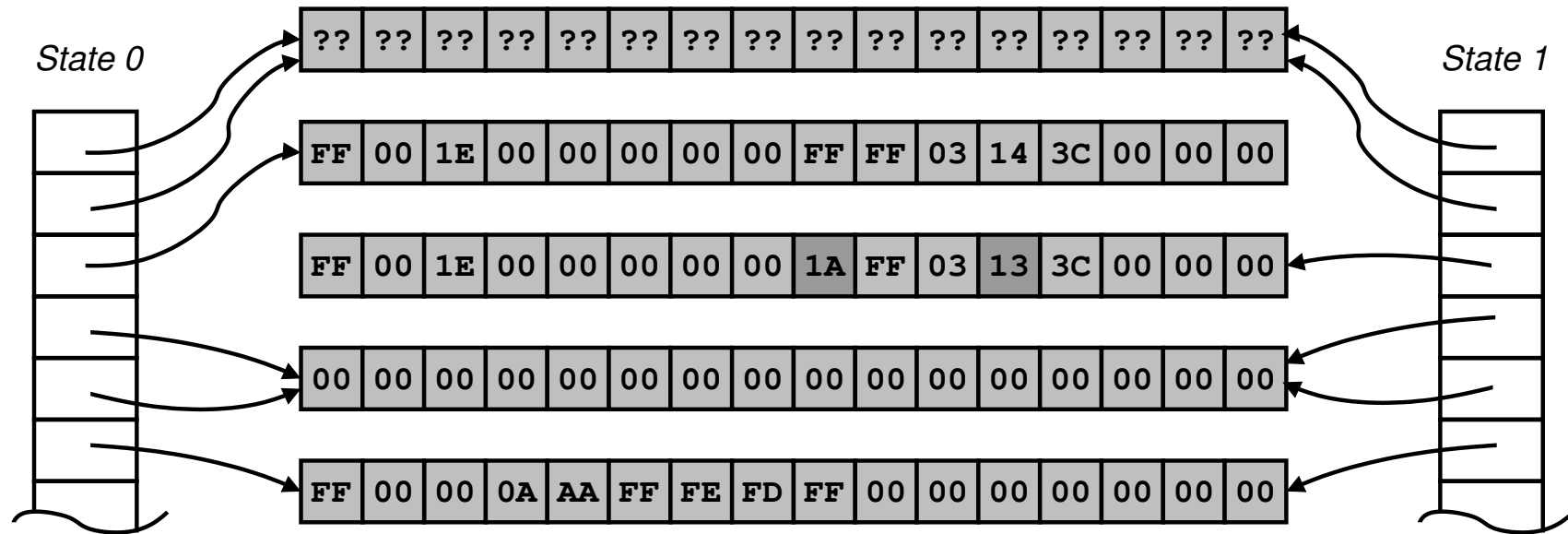
```
...
int buttons = read_buttons();
//
if (buttons & BUTTON_LEFT) { dir = WEST; }
else if (buttons & BUTTON_RIGHT) { dir = EAST; }
else if (buttons & BUTTON_UP) { dir = NORTH; }
else if (buttons & BUTTON_DOWN) { dir = SOUTH; }
else {
    return;
}
...
```

- 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



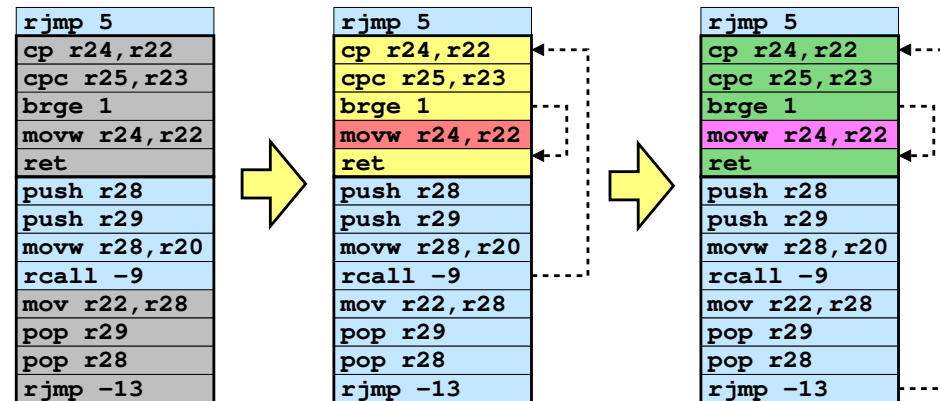
- Illustrates lines from two **related** states.
- Can see **slight difference** exists between third line. Assume all other lines **reused** between the two states.
- Memory required for *State 1* is **152 bytes** (for line array) plus **16 bytes** (for changed line).

# 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 similar 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. Lindstrom<sup>1</sup>, 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.