Formal Methods for Java

Lecture 27: Model Checking Concurrent Java Programs

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

Feb 08, 2012

Introduction to Concurrent Java Programs

Example: Concurrent Java Code

```
public void MyStack {
  int size;
  Object[] elem;
  public void moveTopToStack(Stack other) {


      if (other.size == other.elem.length)
        other.grow();
      other.elem[other.size++] = this.elem[--this.size];


  }
  ...
  MyStack a,b;
  thread1() {
    a.moveTopToStack(b);
  }
  thread2() {
    b.moveTopToStack(a);
  }
}
```

Error Sources in Concurrent Java Code

- Races: Other threads may interfere at any time.
  Even instructions like `elem++` are not atomic.

  Solution: Add synchronized blocks.
- Deadlocks: Threads may block each other.
  Solution: Define a total order on synchronized and obey it everywhere.
- Non-Determinism: Whether or not problems occur depend on machine (multi-core/single-core) and exact timing.

  Problems occur randomly, usually only under heavy load.

Example: Concurrent Java Code

```
public void MyStack {
  int size;
  Object[] elem;

  public synchronized void moveTopToStack(Stack other) {

    synchronized (other) {

      if (other.size == other.elem.length)
        other.grow();
      other.elem[other.size++] = this.elem[--this.size];

    }

  }
  ...
  MyStack a,b;
  thread1() {
    a.moveTopToStack(b);
  }
  thread2() {
    b.moveTopToStack(a);
  }
}
```

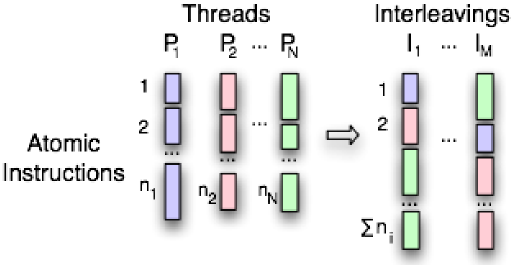
Race Example

Demo

Challenge in Model Checking Concurrent Programs

State Space Explosion

- Model checking has to consider all possible interleavings
- Assume N threads where thread i contains n_i instructions. How many possible interleavings?



$$M = \frac{(\sum_{i=1}^{N} n_i)!}{\prod_{i=1}^{N}(n_i!)}$$

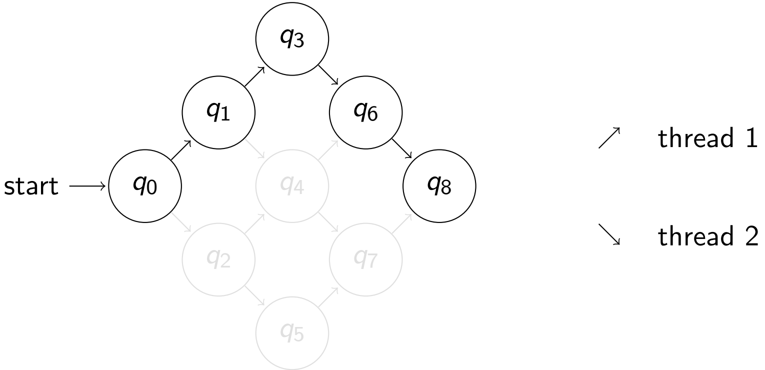
http://babelfish.arc.nasa.gov/trac/jpf/wiki

- For $N = 3$, $n_i = 20$: $M \approx 5.8 \cdot 10^{26}$
- Scalability problem for model checking

Solution: Partial Order Reduction (POR)

Observation

If a context switch does not influence the currently running thread, this interleaving is not interesting.



Partial Order Reduction in JPF

JVM and Concurrency

Observations

- JVM is a stack machine.
- Stacks are local to a thread.
- Most instructions only manipulate the stack.

➥ Only a few instructions can influence other threads.

Instructions Influencing Other Threads

- Field instructions (GETFIELD, PUTFIELD, GETSTATIC, PUTSTATIC)
- Array instructions (xALOAD, xASTORE)
- Synchronization (MONITORENTER, MONITOREXIT)
- Function calls:
  - synchronized functions
  - thread management functions
  - object notification functions

Limiting the Number of Relevant Instructions

Observation

Field and Array instructions, and synchronization only interesting if object is shared.

However, detecting objects shared between multiple threads is expensive.

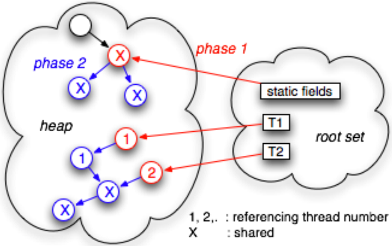
Idea

- Reuse jpf's garbage collector for this (piggybacking).
- Garbage collector marks objects that are reachable.
- We need to mark objects reachable from different threads.

Detecting Shared Objects

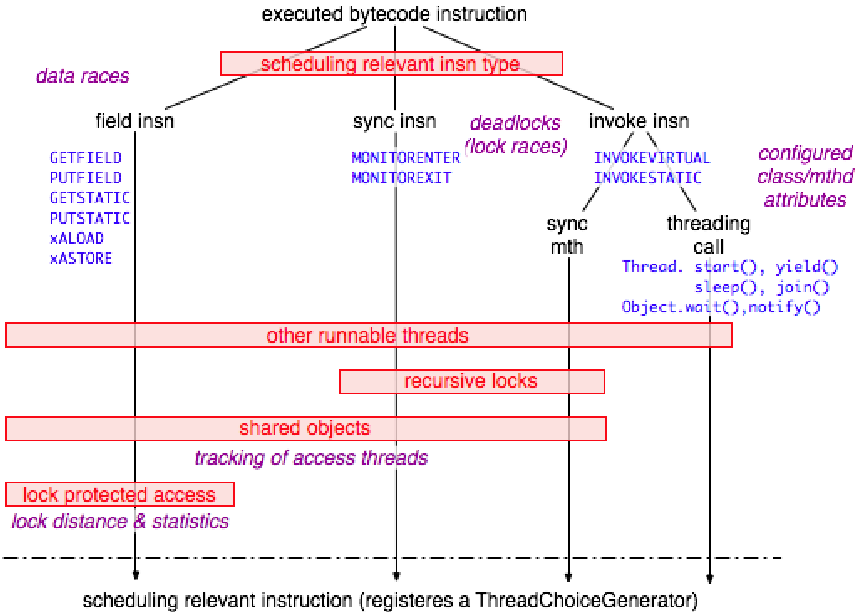
Extending the Mark Phase

- Mark either with thread id, or shared.
- Mark every static field shared.
- For every field f in the root set of Thread i:
  - If f already has a mark different from i, mark f shared.
  - Otherwise mark f with i.
- Propagate marks until a fixed point is reached.



http://babelfish.arc.nasa.gov/trac/jpf/wiki

POR: Relevant instructions

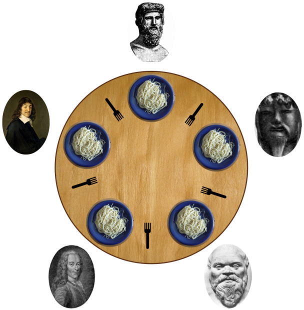


http://babelfish.arc.nasa.gov/trac/jpf/wiki

Concurrency Problems

Dining Philosophers

Five philosophers sit around a round table. A plate with spaghetti is in front of every philosopher. A fork lies to the right of every philosopher. A philosopher is not allowed to speak, but may think, grab or drop a fork, or eat if he has two forks. How can we ensure that no philosopher starves?



http://en.wikipedia.org/wiki/Dining_philosophers_problem

A Solution?

1. Think
2. Grab left fork
3. Grab right fork
4. Eat
5. Drop right fork
6. Drop left fork
7. Go to Step 1

➥ Check with JPF