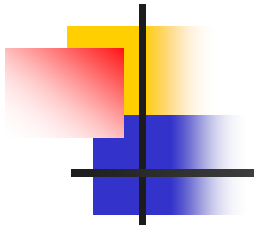
Verify the Implementation of the Concurrent Heap Algorithm by JPF

Shouzheng Yang

CSE6490A

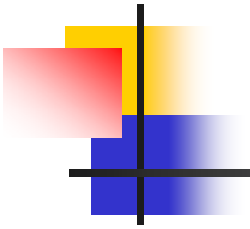
Class Presentation 3

Apr 4, 2011



Outline

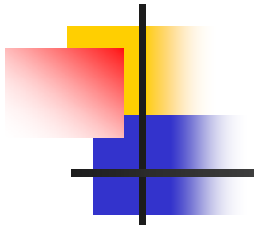
- Preparation For Verification
- Verification With JPF
  - Data Race Detection
    - Inappropriate Scheduling
    - Non-instant Visibility
  - Deadlock Detection
  - Testing Under The Help of JPF
  - Combine JPF and JUnit
  - Other Attempts



Preparation For Verification

- Remove unrelated variables for each specific experiment.
- Use annotation @FilterField
- Limit the number of nodes and threads.

- All above efforts aim at reducing the state space.



Data Race Detection

- A data race condition is an ambiguous condition occurring when one event makes a change to its next state before a second one has had sufficient time to latch.
- Two possibilities
  - Inappropriate Scheduling
  - Non-instant Visibility

Inappropriate Scheduling

- Remove the lock of the root node.

```
====================================================== error #1
gov.nasa.jpf.listener.PreciseRaceDetector
race for field shouzheng.cool6490a.HeapNode@17c.tag
  Thread-1 at shouzheng.cool6490a.ConcurrentHeap.swapItem(ConcurrentHeap.java:271)
                "(shouzheng\cool6490a\ConcurrentHeap.java:271)"  : putfield
  Thread-2 at shouzheng.cool6490a.ConcurrentHeap.delete(ConcurrentHeap.java:151)
                "(shouzheng\cool6490a\ConcurrentHeap.java:151)"  : getfield
```

Non-instant Visibility

```
public class VolatileDataRace {
    static int v; // without volatile keyword
     public static void main(String args[]) {
            v= 10;
           new MyThread().start();
     }
     public static class MyThread extends Thread {
            public void run() {
           System.out.println(v);
           }
     }
}
```

JPF detects nothing!

Another Data Race Detection Example

```
// JPF can report data race error
public class DataRace1 {
static int a;
public static void main(args) {
    new MyThread().start();
    a= 10;
}
public static class MyThread
            extends Thread {
    public void run() {
    System.out.println(a);
    }
}
}
```

```
// JPF cannot report data race error
public class DataRace2 {
    static int a[]=new int[1];
public static void main(args) {
     new MyThread().start();
     a[0]= 10;
}
public static class MyThread
                extends Thread {
       public void run() {
       System.out.println(a[0]);
       }
 }
 }
```

Deadlock Detection

- Insert operation release the parent node lock and child node lock at the end of each parent-child comparison iteration and re-acquire the parent node lock again which is the child node lock in the next iteration.

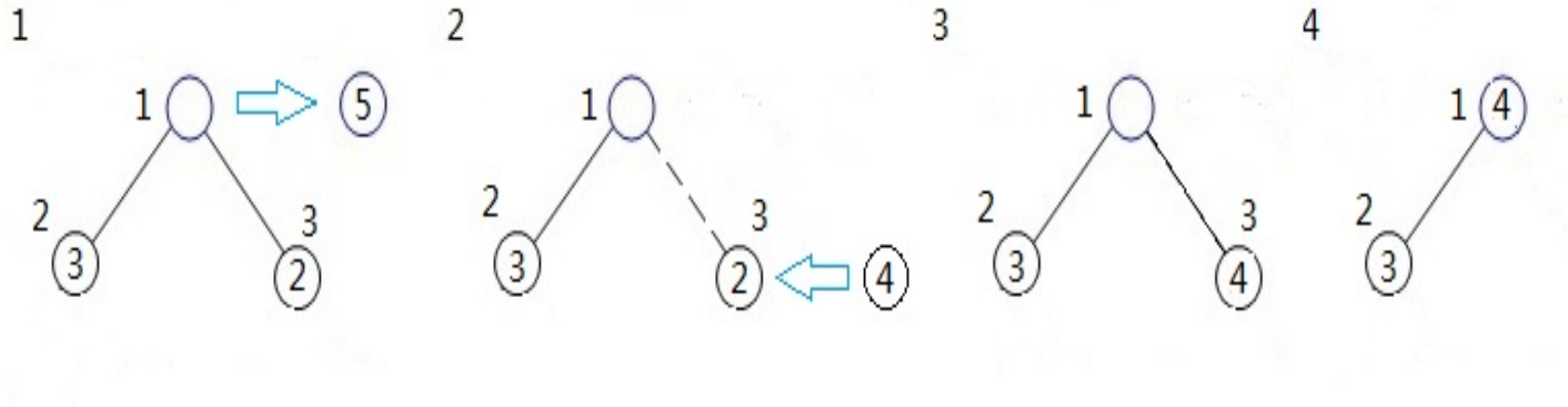
```
================================================== thread ops #1
   3       2       0     trans      insn          loc                : stmt
------- ------- ------- ----------------------------------------------
W:457      |       |         7      invokespecial java\util\concurrent\locks\ReentrantLock.java:49 : (java\util\concurrent\locks\ReentrantLock.java:49)
   |    W:508     |         6      invokespecial java\util\concurrent\locks\ReentrantLock.java:49 : (java\util\concurrent\locks\ReentrantLock.java:49)
   |       |   W:443         5      invokevirtual shouzheng\cool6490a\HeapOperator.java:133 : (shouzheng\cool6490a\HeapOperator.java:133)
   |       S       |         4
   S       |       |         3
```

Testing Under The Help of JPF

```
heapSizeLock.lock();
bottom=decrementSize(heapSize);
heapNodes[bottom].nodeLock.lock();
heapSizeLock.unlock();
.....
heapNodes[bottom].nodeLock.unlock();
```

Testing Under The Help of JPF

- Switch the two highlighted lines.



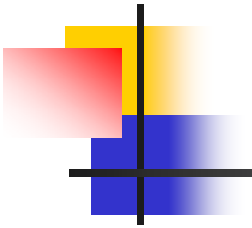
- JPF enumerates all executing paths.
- Printing out useful information for each executing path.

Combine JPF and JUnit

```
public class JPFInJUnit extends TestJPF{
        public static void main(String[] testMethods){
                runTestsOfThisClass(testMethods);
        }
    @Test public void Test1()...
    @Test public void Test1()...
    @Test
    public void TestDeadLock(){
        if (!isJPFRun()) { /* run something outside JPF */ }
        if (verifyDeadlock("+listener=.listener.DeadlockAnalyzer",
                "+deadlock.format=essential",
                "+report.console.property_violation=error,trace,snapshot"))
        {
            Invoke threads....
            System.out.println("There is no deadlock!");
        }
        else System.out.println("There exists deadlock!");
    }
}
```

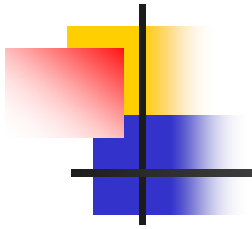
Other Verification Attempts

- Different search strategies
  - BFheuristic
  - DFHeuristic
  - Depth-first
  - Random
- Various listeners
  - CoverageAnalyzer
  - ExecTracker
- Jpf-concurrent project
  - ReentrantLock



Conclusion

- The capability of searching all space states gives JPF special power to verify the concreteness of concurrent programs.
- Bottleneck: search strategy.
- Documentation.



Reference

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

Thank you very much!