Model Checking Concurrent Red-Black Trees

Franck van Breugel

York University, Toronto

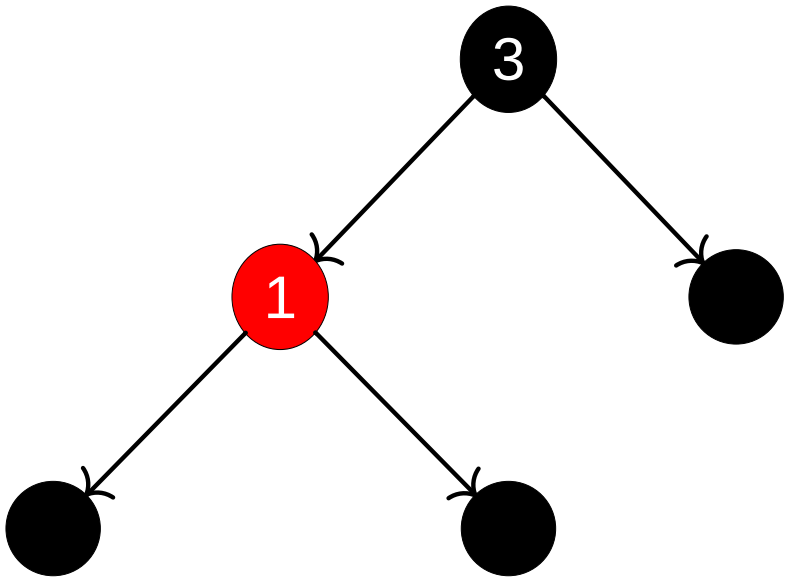
March 22, 2010

Red-Black Tree

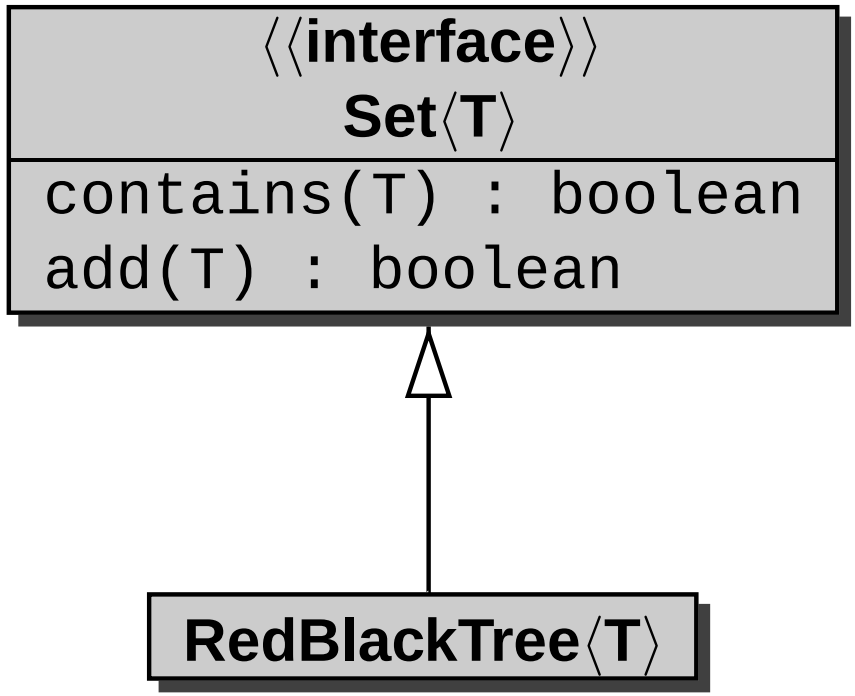
A red-black tree is a binary search tree the nodes of which are coloured either red or black and

- the root is black,
- every leaf is black,
- if a node is red, then both its children are black,
- for every node, every path from that node to a leaf contains the same number of black nodes.

[Bayer, 1972] and [Guibas and Sedgewick, 1978]



Three Implementations



The Monitor Solution

```
1  package monitor;
2
3  public class RedBlackTree<T extends Comparable<T>>
4     implements Set<T>
5  {
6     public synchronized boolean contains(T element)
7     {
8        ...
9     }
10
11    public synchronized boolean add(T element)
12    {
13       ...
14    }
15 }
```

The Readers-Writers Solution

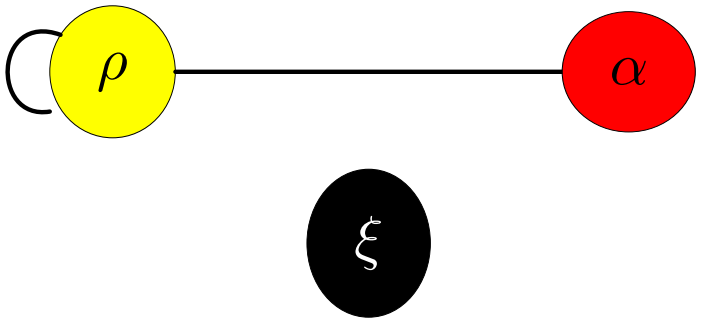
```
1  private ReadWriteLock lock;
2
3  public RedBlackTree()
4  {
5     this.lock = new ReentrantReadWriteLock();
6     ...
7  }
8
9  public boolean contains(T element)
10 {
11    this.lock.getReadLock().lock();
12    ...
13    this.lock.getReadLock().unlock();
14 }
15 ...
```

The Fine-Grained Locking Solution

Processes lock the nodes of the red-black tree in three different ways:

- ρ-lock: lock to read
- α-lock: lock to exclude writers
- ξ-lock: exclusive lock

Although a node can be locked by multiple processes, there are some restrictions.



Properties

- some synchronization is needed
- deadlock freedom
- no uncaught exceptions
- no data races
- post-conditions

Some Synchronization is Needed

```
1 add(3);
2 add(1);
3 (add(2) || assert(contains(1)))
```

Some Synchronization is Needed

JPF found an interleaving leading to an uncaught exception

```
% jpf +classpath=. sequential.Test
JavaPathfinder v5.x - (C) RIACS/NASA Ames Research
==========================================
application: sequential/Test.java
==========================================
gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty
java.lang.AssertionError
        at sequential.Checker.run(Checker.java:35)
==========================================
error #1: gov.nasa.jpf.jvm.NoUncaughtExceptionsProp
==========================================
elapsed time:       0:00:01
states:             new=131, visited=50, backtracke
search:             maxDepth=69, constraints=0
choice generators:  thread=130, data=0
heap:               gc=204, new=593, free=312
```

Deadlocks, Exceptions and Data Races

Numerous small tests were verified by JPF for the three implementations:

- no deadlocks,
- no uncaught exceptions,
- no data races.

Post-Conditions

Added to the implementations:

- isOk(): tests whether the tree is a red-black tree
- elements(): returns the collection of elements of the tree

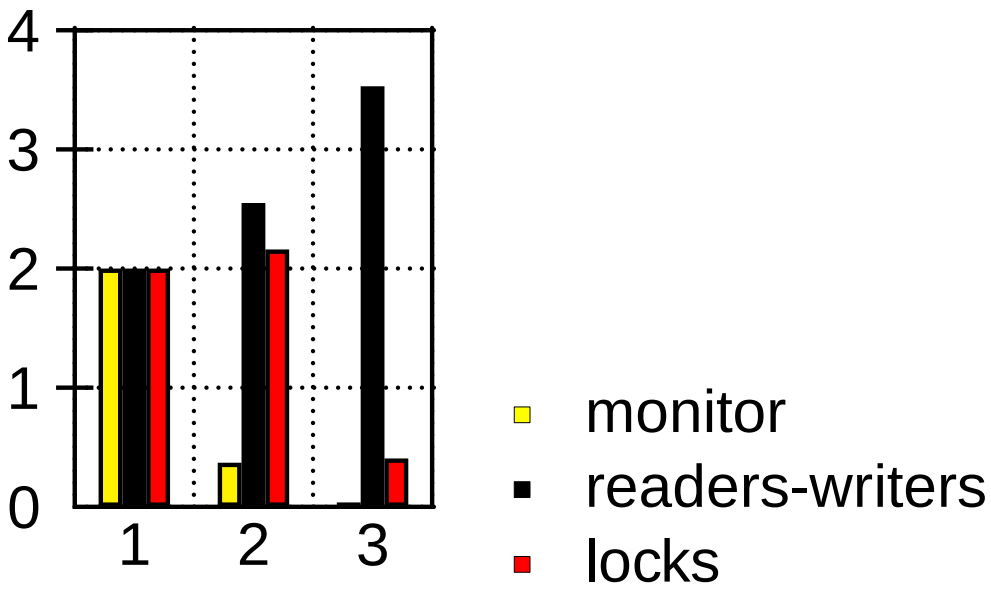
Post-Conditions

```
1 (tree.add(1) || tree.add(2));
2 assert tree.isOk();
3 assert tree.elements().contains(1);
4 assert tree.elements().contains(2);
```

State Spaces

```
1  add(1) ‖ ··· ‖ add(n)
```

State Space



Conclusion

Three algorithms

- the monitor solution
  - simplest implementation
  - smallest state space
- the readers-writers solution
  - most efficient implementation
  - largest state space
- the locks solution
  - most complicated implementation
  - most inefficient implementation

And the winner is ...

???