Concurrent Red-Black Trees

Franck van Breugel

York University, Toronto

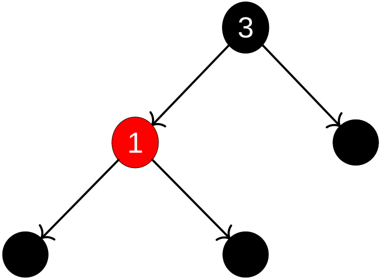
May 12, 2009

Red-Black Tree

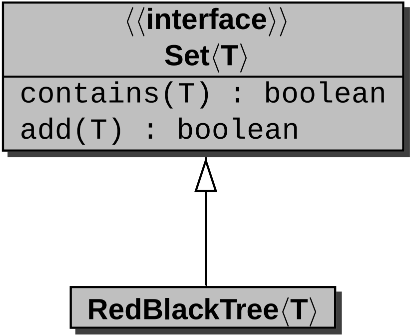
A red-black tree is a binary search tree the nodes of which are coloured either red or black and

- 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

```
package monitor;

public class RedBlackTree<T extends Comparable<T>>
  implements Set<T>
{
  public synchronized boolean contains(T element)
  {
    ...
  }

  public synchronized boolean add(T element)
  {
    ...
  }
}
```

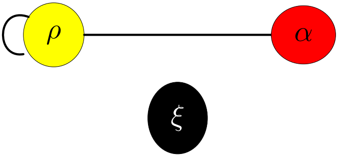
The Readers-Writers Solution

```
private ReadWriteLock lock;

public RedBlackTree()
{
    this.lock = new ReentrantReadWriteLock();
    ...
}

public boolean contains(T element)
{
    this.lock.getReadLock().lock();
    ...
    this.lock.getReadLock().unlock();
}
...
```

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

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

Some Synchronization is Needed

JPF found an interleaving leading to an uncaught exception

```
==================================================
gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty
java.lang.NoClassDefFoundError: RedBlackTree
        at Main.main(Starter.java:11)

==================================================
elapsed time:         0:00:00
states:               new=0, visited=1, backtracked=0
search:               maxDepth=0, constraints=0
choice generators:    thread=1, data=0
heap:                 gc=0, new=205, free=0
instructions:         2079
max memory:           16MB
loaded code:          classes=56, methods=763
```

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 collections of elements of the tree

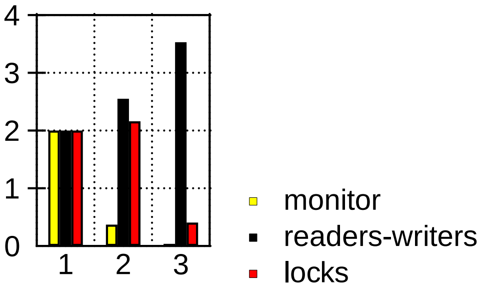
Post-Conditions

```
(tree.add(1) ‖ tree.add(2));
assert tree.isOk();
assert tree.elements().contains(1);
assert tree.elements().contains(2);
```

State Spaces

```
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 ...

???