**Efficient Functional Maps**

**Functional Balanced Search Trees**

Idea:

* update creates a new path (copying only log(n) nodes)
* lookup and update in log(n)

We present a simplified example that stores integers

* a real implementation stores *comparable* objects
* instead of ‘<’ below, we would use the comparison method

**Case Class Definition**

[sealed](http://scala-lang.org) [abstract](http://scala-lang.org) [class](http://scala-lang.org) BST

[case](http://scala-lang.org) [class](http://scala-lang.org) Empty() [extends](http://scala-lang.org) BST

[case](http://scala-lang.org) [class](http://scala-lang.org) Node(left: BST, value: Int, right: BST) [extends](http://scala-lang.org) BST

**Checking Membership in Tree**

[def](http://scala-lang.org) contains(key: Int, t : BST): Boolean = t [match](http://scala-lang.org) {

[case](http://scala-lang.org) Empty() => [false](http://scala-lang.org)

[case](http://scala-lang.org) Node(left,v,right) => {

[if](http://scala-lang.org) (key == v) [true](http://scala-lang.org)

[else](http://scala-lang.org) [if](http://scala-lang.org) (key < v) contains(key, left)

[else](http://scala-lang.org) contains(key, right)

}

}

**Inserting into Tree**

[def](http://scala-lang.org) add(x : Int, t : BST) : Node = t [match](http://scala-lang.org) {

[case](http://scala-lang.org) Empty() => Node(Empty(),x,Empty())

[case](http://scala-lang.org) t @ Node(left,v,right) => {

[if](http://scala-lang.org) (x < v) Node(add(x, left), v, right)

[else](http://scala-lang.org) [if](http://scala-lang.org) (x==v) t

[else](http://scala-lang.org) Node(left, v, add(x, right))

}

}

**Ensuring Balancing: Red-Black Trees**

Idea:

* prevent tree from degenerating into a list with bad sequence of insertions
* rebalance tree after each change

Invariants that imply balancing formulated using nodes of two **colors**: red and black.

* red node must have black children
* each path in tree from root to children must have the same number of black nodes

**Data Type with Colors**

[sealed](http://scala-lang.org) [abstract](http://scala-lang.org) [class](http://scala-lang.org) Color

[case](http://scala-lang.org) [class](http://scala-lang.org) Red() [extends](http://scala-lang.org) Color

[case](http://scala-lang.org) [class](http://scala-lang.org) Black() [extends](http://scala-lang.org) Color

[sealed](http://scala-lang.org) [abstract](http://scala-lang.org) [class](http://scala-lang.org) Tree

[case](http://scala-lang.org) [class](http://scala-lang.org) Empty() [extends](http://scala-lang.org) Tree

[case](http://scala-lang.org) [class](http://scala-lang.org) Node(color: Color, left: Tree, value: Int, right: Tree) [extends](http://scala-lang.org) Tree

**Inserting with Balancing**

[def](http://scala-lang.org) add(x: Int, t: Tree): Tree = {

[def](http://scala-lang.org) ins(t: Tree): Tree = t [match](http://scala-lang.org) {

[case](http://scala-lang.org) Empty() => Node(Red(),Empty(),x,Empty())

[case](http://scala-lang.org) Node(c,a,y,b) =>

[if](http://scala-lang.org) (x < y) balance(c, ins(a), y, b)

[else](http://scala-lang.org) [if](http://scala-lang.org) (x == y) Node(c,a,y,b)

[else](http://scala-lang.org) balance(c,a,y,ins(b))

}

makeBlack(ins(t))

}

[def](http://scala-lang.org) balance(c: Color, a: Tree, x: Int, b: Tree): Tree = (c,a,x,b) [match](http://scala-lang.org) {

[case](http://scala-lang.org) (Black(),Node(Red(),Node(Red(),a,xV,b),yV,c),zV,d) =>

Node(Red(),Node(Black(),a,xV,b),yV,Node(Black(),c,zV,d))

[case](http://scala-lang.org) (Black(),Node(Red(),a,xV,Node(Red(),b,yV,c)),zV,d) =>

Node(Red(),Node(Black(),a,xV,b),yV,Node(Black(),c,zV,d))

[case](http://scala-lang.org) (Black(),a,xV,Node(Red(),Node(Red(),b,yV,c),zV,d)) =>

Node(Red(),Node(Black(),a,xV,b),yV,Node(Black(),c,zV,d))

[case](http://scala-lang.org) (Black(),a,xV,Node(Red(),b,yV,Node(Red(),c,zV,d))) =>

Node(Red(),Node(Black(),a,xV,b),yV,Node(Black(),c,zV,d))

[case](http://scala-lang.org) (c,a,xV,b) => Node(c,a,xV,b)

}

[def](http://scala-lang.org) makeBlack(n: Tree): Tree = n [match](http://scala-lang.org) {

[case](http://scala-lang.org) Node(Red(),l,v,r) => Node(Black(),l,v,r)

[case](http://scala-lang.org) \_ => n

}

**Benefit Summary**

* keep old versions (e.g. when exiting scope, the old scope is around)
* easier to prove correct

**References**

* Book by Chris Okasaki: Purely Functional Data Structures
* [Red-Black Trees in Functional Settings](http://lara.epfl.ch/web2010/_media/compilation:okasakiredblack99.ps?id=cc09%3Afunctional_maps&cache=cache) (a pearl)
* [Binary Search Trees Proved Correct](http://afp.sourceforge.net/entries/BinarySearchTree.shtml)
* [AVL Trees Proved Correct](http://afp.sourceforge.net/entries/AVL-Trees.shtml) (also balanced, but not red-black)

**Imperative Table with Update Log**

Emulate functional behavior using an imperative hash table

* internally mutable, observationally immutable

Insertion:

* updates the hashtable
* creates a log entry that specifies how the old version differs from new one

Old version:

* checks if the value has been changed, if so uses log to look it up
* otherwise looks up the log in the table

Benefits:

* constant-time access for last version produced in computation

**References**

* [Immutable TreeMap in Scala](http://www.scala-lang.org/docu/files/api/scala/collection/immutable/TreeMap.html)