

# A Verified Imperative Implementation of B-Trees

Niels Mündler

## Abstract

In this work, we use the interactive theorem prover Isabelle/HOL to verify an imperative implementation of the classical B-tree data structure [1]. The implementation supports set membership, insertion, deletion, iteration and range queries with efficient binary search for intra-node navigation. This is accomplished by first specifying the structure abstractly in the functional modeling language HOL and proving functional correctness. Using manual refinement, we derive an imperative implementation in Imperative/HOL. We show the validity of this refinement using the separation logic utilities from the Isabelle Refinement Framework [2]. The code can be exported to the programming languages SML, Scala and OCaml. This entry contains two developments:

- *B-Trees* This formalisation is discussed in greater detail in the corresponding Bachelor's Thesis[3].
- *B<sup>+</sup>-Trees* This formalisation also supports range queries and is discussed in a paper published at ICTAC 2022.

## Contents

<b>1</b>	<b>Definition of the B-Tree</b>	<b>2</b>
1.1	Datatype definition . . . . .	2
1.2	Inorder and Set . . . . .	3
1.3	Height and Balancedness . . . . .	3
1.4	Order . . . . .	4
1.5	Auxiliary Lemmas . . . . .	4
<b>2</b>	<b>Maximum and minimum height</b>	<b>7</b>
2.1	Definition of node/size . . . . .	7
2.2	Maximum number of nodes for a given height . . . . .	8
2.3	Maximum height for a given number of nodes . . . . .	8
<b>3</b>	<b>Set interpretation</b>	<b>10</b>
3.1	Auxiliary functions . . . . .	10
3.2	The split function locale . . . . .	11
3.3	Membership . . . . .	11

3.4	Insertion . . . . .	11
3.5	Deletion . . . . .	13
3.6	Proofs of functional correctness . . . . .	15
3.7	Set specification by inorder . . . . .	24
<b>4</b>	<b>Abstract split functions</b>	<b>24</b>
4.1	Linear split . . . . .	24
4.2	Binary split . . . . .	26
<b>5</b>	<b>Definition of the B-Plus-Tree</b>	<b>27</b>
5.1	Datatype definition . . . . .	27
5.2	Inorder and Set . . . . .	28
5.3	Height and Balancedness . . . . .	29
5.4	Order . . . . .	29
5.5	Auxiliary Lemmas . . . . .	30
5.6	Auxiliary functions . . . . .	36
5.7	The split function locale . . . . .	36
<b>6</b>	<b>Abstract split functions</b>	<b>37</b>
6.1	Linear split . . . . .	37
<b>7</b>	<b>Set interpretation</b>	<b>38</b>
7.1	Membership . . . . .	40
7.2	Insertion . . . . .	41
7.3	Proofs of functional correctness . . . . .	43
7.4	Deletion . . . . .	49
7.5	Set specification by inorder . . . . .	57

**theory** *BTree*

**imports** *Main HOL-Data-Structures.Sorted-Less HOL-Data-Structures.Cmp*  
**begin**

**hide-const** (**open**) *Sorted-Less.sorted*

**abbreviation** *sorted-less*  $\equiv$  *Sorted-Less.sorted*

## 1 Definition of the B-Tree

### 1.1 Datatype definition

B-trees can be considered to have all data stored interleaved as child nodes and separating elements (also keys or indices). We define them to either be a `Node` that holds a list of pairs of children and indices or be a completely empty `Leaf`.

**datatype** *'a btree* = *Leaf* | *Node ('a btree \* 'a) list 'a btree*

```

type-synonym 'a btree-list = ('a btree * 'a) list
type-synonym 'a btree-pair = ('a btree * 'a)

```

```

abbreviation subtrees where subtrees xs ≡ (map fst xs)
abbreviation separators where separators xs ≡ (map snd xs)

```

## 1.2 Inorder and Set

The set of B-tree elements is defined automatically.

```

thm btree.set
value set-btree (Node [(Leaf, (0::nat)), (Node [(Leaf, 1), (Leaf, 10)] Leaf, 12),
(Leaf, 30), (Leaf, 100)] Leaf)

```

The inorder view is defined with the help of the concat function.

```

fun inorder :: 'a btree ⇒ 'a list where
  inorder Leaf = [] |
  inorder (Node ts t) = concat (map (λ (sub, sep). inorder sub @ [sep]) ts) @
inorder t

```

```

abbreviation inorder-pair ≡ λ(sub,sep). inorder sub @ [sep]
abbreviation inorder-list ts ≡ concat (map inorder-pair ts)

```

```

thm inorder.simps

```

```

value inorder (Node [(Leaf, (0::nat)), (Node [(Leaf, 1), (Leaf, 10)] Leaf, 12),
(Leaf, 30), (Leaf, 100)] Leaf)

```

## 1.3 Height and Balancedness

```

class height =
  fixes height :: 'a ⇒ nat

```

```

instantiation btree :: (type) height
begin

```

```

fun height-btree :: 'a btree ⇒ nat where
  height Leaf = 0 |
  height (Node ts t) = Suc (Max (height ‘ (set (subtrees ts@[t])))

```

```

instance ⟨proof⟩

```

```

end

```

Balancedness is defined in close accordance to the definition by Ernst

```

fun bal:: 'a btree ⇒ bool where
  bal Leaf = True |
  bal (Node ts t) = (

```

```

    (∀ sub ∈ set (subtrees ts). height sub = height t) ∧
    (∀ sub ∈ set (subtrees ts). bal sub) ∧ bal t
  )

```

```

value height (Node [(Leaf, (0::nat)), (Node [(Leaf, 1), (Leaf, 10)] Leaf, 12), (Leaf,
30), (Leaf, 100)] Leaf)

```

## 1.4 Order

The order of a B-tree is defined just as in the original paper by Bayer.

```

fun order:: nat ⇒ 'a btree ⇒ bool where
  order k Leaf = True |
  order k (Node ts t) = (
    (length ts ≥ k) ∧
    (length ts ≤ 2*k) ∧
    (∀ sub ∈ set (subtrees ts). order k sub) ∧ order k t
  )

```

The special condition for the root is called *root\_order*

```

fun root-order:: nat ⇒ 'a btree ⇒ bool where
  root-order k Leaf = True |
  root-order k (Node ts t) = (
    (length ts > 0) ∧
    (length ts ≤ 2*k) ∧
    (∀ s ∈ set (subtrees ts). order k s) ∧ order k t
  )

```

## 1.5 Auxiliary Lemmas

**lemma** *separators-split*:

```

set (separators (l@(a,b)#r)) = set (separators l) ∪ set (separators r) ∪ {b}
⟨proof⟩

```

**lemma** *subtrees-split*:

```

set (subtrees (l@(a,b)#r)) = set (subtrees l) ∪ set (subtrees r) ∪ {a}
⟨proof⟩

```

**lemma** *finite-set-ins-swap*:

```

assumes finite A
shows max a (Max (Set.insert b A)) = max b (Max (Set.insert a A))
⟨proof⟩

```

**lemma** *finite-set-in-idem*:

```

assumes finite A
shows max a (Max (Set.insert a A)) = Max (Set.insert a A)

```

*<proof>*

**lemma** *height-Leaf*:  $height\ t = 0 \longleftrightarrow t = Leaf$   
*<proof>*

**lemma** *height-btree-order*:  
 $height\ (Node\ (ls@[a])\ t) = height\ (Node\ (a\#ls)\ t)$   
*<proof>*

**lemma** *height-btree-sub*:  
 $height\ (Node\ ((sub,x)\#ls)\ t) = max\ (height\ (Node\ ls\ t))\ (Suc\ (height\ sub))$   
*<proof>*

**lemma** *height-btree-last*:  
 $height\ (Node\ ((sub,x)\#ts)\ t) = max\ (height\ (Node\ ts\ sub))\ (Suc\ (height\ t))$   
*<proof>*

**lemma** *set-btree-inorder*:  $set\ (inorder\ t) = set-btree\ t$   
*<proof>*

**lemma** *child-subset*:  $p \in set\ t \implies set-btree\ (fst\ p) \subseteq set-btree\ (Node\ t\ n)$   
*<proof>*

**lemma** *some-child-sub*:  
**assumes**  $(sub,sep) \in set\ t$   
**shows**  $sub \in set\ (subtrees\ t)$   
**and**  $sep \in set\ (separators\ t)$   
*<proof>*

**lemma** *bal-all-subtrees-equal*:  $bal\ (Node\ ts\ t) \implies (\forall\ s1 \in set\ (subtrees\ ts). \forall\ s2 \in set\ (subtrees\ ts). height\ s1 = height\ s2)$   
*<proof>*

**lemma** *fold-max-set*:  $\forall x \in set\ t. x = f \implies fold\ max\ t\ f = f$   
*<proof>*

**lemma** *height-bal-tree*:  $bal\ (Node\ ts\ t) \implies height\ (Node\ ts\ t) = Suc\ (height\ t)$   
*<proof>*

**lemma** *bal-split-last*:  
**assumes**  $bal\ (Node\ (ls@(sub,sep)\#rs)\ t)$

**shows**  $bal (Node (ls@rs) t)$   
**and**  $height (Node (ls@(sub,sep)#rs) t) = height (Node (ls@rs) t)$   
 $\langle proof \rangle$

**lemma** *bal-split-right*:  
**assumes**  $bal (Node (ls@rs) t)$   
**shows**  $bal (Node rs t)$   
**and**  $height (Node rs t) = height (Node (ls@rs) t)$   
 $\langle proof \rangle$

**lemma** *bal-split-left*:  
**assumes**  $bal (Node (ls@(a,b)#rs) t)$   
**shows**  $bal (Node ls a)$   
**and**  $height (Node ls a) = height (Node (ls@(a,b)#rs) t)$   
 $\langle proof \rangle$

**lemma** *bal-substitute*:  $\llbracket bal (Node (ls@(a,b)#rs) t); height t = height c; bal c \rrbracket \implies$   
 $bal (Node (ls@(c,b)#rs) t)$   
 $\langle proof \rangle$

**lemma** *bal-substitute-subtree*:  $\llbracket bal (Node (ls@(a,b)#rs) t); height a = height c; bal$   
 $c \rrbracket \implies bal (Node (ls@(c,b)#rs) t)$   
 $\langle proof \rangle$

**lemma** *bal-substitute-separator*:  $bal (Node (ls@(a,b)#rs) t) \implies bal (Node (ls@(a,c)#rs)$   
 $t)$   
 $\langle proof \rangle$

**lemma** *order-impl-root-order*:  $\llbracket k > 0; order k t \rrbracket \implies root-order k t$   
 $\langle proof \rangle$

**lemma** *sorted-inorder-list-separators*:  $sorted-less (inorder-list ts) \implies sorted-less$   
 $(separators ts)$   
 $\langle proof \rangle$

**corollary** *sorted-inorder-separators*:  $sorted-less (inorder (Node ts t)) \implies sorted-less$   
 $(separators ts)$   
 $\langle proof \rangle$

**lemma** *sorted-inorder-list-subtrees*:  
 $sorted-less (inorder-list ts) \implies \forall sub \in set (subtrees ts). sorted-less (inorder sub)$

*<proof>*

**corollary** *sorted-inorder-subtrees*:  $\text{sorted-less (inorder (Node ts t))} \implies \forall \text{ sub} \in \text{set (subtrees ts)}. \text{sorted-less (inorder sub)}$   
*<proof>*

**lemma** *sorted-inorder-list-induct-subtree*:  
 $\text{sorted-less (inorder-list (ls@(sub,sep)#rs))} \implies \text{sorted-less (inorder sub)}$   
*<proof>*

**corollary** *sorted-inorder-induct-subtree*:  
 $\text{sorted-less (inorder (Node (ls@(sub,sep)#rs) t))} \implies \text{sorted-less (inorder sub)}$   
*<proof>*

**lemma** *sorted-inorder-induct-last*:  $\text{sorted-less (inorder (Node ts t))} \implies \text{sorted-less (inorder t)}$   
*<proof>*

**end**

**theory** *BTree-Height*

**imports** *BTree*

**begin**

## 2 Maximum and minimum height

Textbooks usually provide some proofs relating the maximum and minimum height of the BTree for a given number of nodes. We therefore introduce this counting and show the respective proofs.

### 2.1 Definition of node/size

**thm** *BTree.btree.size*

**value** *size* (Node [(Leaf, (0::nat)), (Node [(Leaf, 1), (Leaf, 10)] Leaf, 12), (Leaf, 30), (Leaf, 100)] Leaf)

The default size function does not suit our needs as it regards the length of the list in each node. We would like to count the number of nodes in the tree only, not regarding the number of keys.

**fun** *nodes::'a btree  $\Rightarrow$  nat* **where**  
*nodes* Leaf = 0 |  
*nodes* (Node ts t) = 1 + ( $\sum t \leftarrow \text{subtrees ts}. \text{nodes } t$ ) + (*nodes* t)

**value** *nodes* (Node [(Leaf, (0::nat)), (Node [(Leaf, 1), (Leaf, 10)] Leaf, 12), (Leaf, 30), (Leaf, 100)] Leaf)

## 2.2 Maximum number of nodes for a given height

**lemma** *sum-list-replicate*:  $\text{sum-list } (\text{replicate } n \ c) = n * c$   
 ⟨proof⟩

**abbreviation**  $\text{bound } k \ h \equiv ((k+1) \wedge^h - 1)$

**lemma** *nodes-height-upper-bound*:  
 $\llbracket \text{order } k \ t; \text{bal } t \rrbracket \implies \text{nodes } t * (2*k) \leq \text{bound } (2*k) \ (\text{height } t)$   
 ⟨proof⟩

To verify our lower bound is sharp, we compare it to the height of artificially constructed full trees.

**fun** *full-node*:: $\text{nat} \Rightarrow 'a \Rightarrow \text{nat} \Rightarrow 'a \ \text{btree}$  **where**  
 $\text{full-node } k \ c \ 0 = \text{Leaf}$   
 $\text{full-node } k \ c \ (\text{Suc } n) = (\text{Node } (\text{replicate } (2*k) \ ((\text{full-node } k \ c \ n), c)) \ (\text{full-node } k \ c \ n))$

**value**  $\text{let } k = (2::\text{nat}) \ \text{in } \text{map } (\lambda x. \text{nodes } x * 2*k) \ (\text{map } (\text{full-node } k \ (1::\text{nat})) \ [0,1,2,3,4])$

**value**  $\text{let } k = (2::\text{nat}) \ \text{in } \text{map } (\lambda x. ((2*k+(1::\text{nat})) \wedge^x - 1)) \ [0,1,2,3,4]$

**lemma** *compow-comp-id*:  $c > 0 \implies f \circ f = f \implies (f \wedge\wedge c) = f$   
 ⟨proof⟩

**lemma** *compow-id-point*:  $f \ x = x \implies (f \wedge\wedge c) \ x = x$   
 ⟨proof⟩

**lemma** *height-full-node*:  $\text{height } (\text{full-node } k \ a \ h) = h$   
 ⟨proof⟩

**lemma** *bal-full-node*:  $\text{bal } (\text{full-node } k \ a \ h)$   
 ⟨proof⟩

**lemma** *order-full-node*:  $\text{order } k \ (\text{full-node } k \ a \ h)$   
 ⟨proof⟩

**lemma** *full-btrees-sharp*:  $\text{nodes } (\text{full-node } k \ a \ h) * (2*k) = \text{bound } (2*k) \ h$   
 ⟨proof⟩

**lemma** *upper-bound-sharp-node*:  
 $t = \text{full-node } k \ a \ h \implies \text{height } t = h \wedge \text{order } k \ t \wedge \text{bal } t \wedge \text{bound } (2*k) \ h = \text{nodes } t * (2*k)$   
 ⟨proof⟩

## 2.3 Maximum height for a given number of nodes

**lemma** *nodes-height-lower-bound*:  
 $\llbracket \text{order } k \ t; \text{bal } t \rrbracket \implies \text{bound } k \ (\text{height } t) \leq \text{nodes } t * k$



*<proof>*

To verify our upper bound is sharp, we compare it to the height of artificially constructed minimally filled (=slim) trees.

```
fun slim-node::nat  $\Rightarrow$  'a  $\Rightarrow$  nat  $\Rightarrow$  'a btree where
  slim-node k c 0 = Leaf|
  slim-node k c (Suc n) = (Node (replicate k ((slim-node k c n),c)) (slim-node k c n))
```

```
value let k = (2::nat) in map ( $\lambda x$ . nodes x * k) (map (slim-node k (1::nat)) [0,1,2,3,4])
```

```
value let k = (2::nat) in map ( $\lambda x$ . ((k+1::nat)(x-1)) [0,1,2,3,4]
```

**lemma** height-slim-node: height (slim-node k a h) = h

*<proof>*

**lemma** bal-slim-node: bal (slim-node k a h)

*<proof>*

**lemma** order-slim-node: order k (slim-node k a h)

*<proof>*

**lemma** slim-nodes-sharp: nodes (slim-node k a h) \* k = bound k h

*<proof>*

**lemma** lower-bound-sharp-node:

$t = \text{slim-node } k \ a \ h \implies \text{height } t = h \wedge \text{order } k \ t \wedge \text{bal } t \wedge \text{bound } k \ h = \text{nodes } t * k$

*<proof>*

Since BTrees have special roots, we need to show the overall nodes separately

**lemma** nodes-root-height-lower-bound:

**assumes** root-order k t

**and** bal t

**shows**  $2 * ((k+1)^{\text{height } t - 1} - 1) + (\text{of-bool } (t \neq \text{Leaf})) * k \leq \text{nodes } t * k$

*<proof>*

**lemma** nodes-root-height-upper-bound:

**assumes** root-order k t

**and** bal t

**shows**  $\text{nodes } t * (2 * k) \leq (2 * k + 1)^{\text{height } t} - 1$

*<proof>*

**lemma** root-order-imp-divmuleq: root-order k t  $\implies$  (nodes t \* k) div k = nodes t

*<proof>*

**lemma** nodes-root-height-lower-bound-simp:

**assumes** root-order k t

**and** *bal t*  
**and**  $k > 0$   
**shows**  $(2*((k+1)^\wedge(\text{height } t - 1) - 1)) \text{ div } k + (\text{of-bool } (t \neq \text{Leaf})) \leq \text{nodes } t$   
 <proof>

**lemma** *nodes-root-height-upper-bound-simp*:  
**assumes** *root-order k t*  
**and** *bal t*  
**shows**  $\text{nodes } t \leq ((2*k+1)^\wedge(\text{height } t) - 1) \text{ div } (2*k)$   
 <proof>

**definition** *full-tree = full-node*

**fun** *slim-tree* **where**  
*slim-tree k c 0 = Leaf* |  
*slim-tree k c (Suc h) = Node [(slim-node k c h, c)] (slim-node k c h)*

**lemma** *lower-bound-sharp*:  
 $k > 0 \implies t = \text{slim-tree } k a h \implies \text{height } t = h \wedge \text{root-order } k t \wedge \text{bal } t \wedge \text{nodes } t * k = 2*((k+1)^\wedge(\text{height } t - 1) - 1) + (\text{of-bool } (t \neq \text{Leaf})) * k$   
 <proof>

**lemma** *upper-bound-sharp*:  
 $k > 0 \implies t = \text{full-tree } k a h \implies \text{height } t = h \wedge \text{root-order } k t \wedge \text{bal } t \wedge ((2*k+1)^\wedge(\text{height } t) - 1) = \text{nodes } t * (2*k)$   
 <proof>

**end**  
**theory** *BTree-Set*  
**imports** *BTree*  
*HOL-Data-Structures.Set-Specs*  
**begin**

## 3 Set interpretation

### 3.1 Auxiliary functions

**fun** *split-half*::  $('a \text{ btree} \times 'a) \text{ list} \Rightarrow (('a \text{ btree} \times 'a) \text{ list} \times ('a \text{ btree} \times 'a) \text{ list})$  **where**  
*split-half xs = (take (length xs div 2) xs, drop (length xs div 2) xs)*

**lemma** *drop-not-empty*:  $xs \neq [] \implies \text{drop } (\text{length } xs \text{ div } 2) \text{ xs} \neq []$   
 <proof>

**lemma** *split-half-not-empty*:  $\text{length } xs \geq 1 \implies \exists ls \text{ sub sep rs. } \text{split-half } xs = (ls, (\text{sub}, \text{sep}) \# rs)$   
 <proof>

### 3.2 The split function locale

Here, we abstract away the inner workings of the split function for B-tree operations.

```

locale split =
  fixes split :: ('a btree × 'a::linorder) list ⇒ 'a ⇒ (('a btree × 'a) list × ('a btree × 'a) list)
  assumes split-req:
    [[split xs p = (ls,rs)]] ⇒ xs = ls @ rs
    [[split xs p = (ls@[ (sub,sep)],rs); sorted-less (separators xs)]] ⇒ sep < p
    [[split xs p = (ls,(sub,sep)#rs); sorted-less (separators xs)]] ⇒ p ≤ sep
begin

lemmas split-conc = split-req(1)
lemmas split-sorted = split-req(2,3)

```

```

lemma [termination-simp]:(ls, (sub, sep) # rs) = split ts y ⇒
  size sub < Suc (size-list (λx. Suc (size (fst x))) ts + size l)
  <proof>

```

```

fun invar-inorder where invar-inorder k t = (bal t ∧ root-order k t)

```

```

definition empty-btree = Leaf

```

### 3.3 Membership

```

fun isin:: 'a btree ⇒ 'a ⇒ bool where
  isin (Leaf) y = False |
  isin (Node ts t) y = (
    case split ts y of (_,(sub,sep)#rs) ⇒ (
      if y = sep then
        True
      else
        isin sub y
    )
  | (_,[]) ⇒ isin t y
)

```

### 3.4 Insertion

The insert function requires an auxiliary data structure and auxiliary invariant functions.

```

datatype 'b upi = Ti 'b btree | Upi 'b btree 'b 'b btree

```

```

fun order-upi where
  order-upi k (Ti sub) = order k sub |

```

$order\text{-}up_i\ k\ (Up_i\ l\ a\ r) = (order\ k\ l \wedge order\ k\ r)$

**fun** *root-order-up<sub>i</sub>* **where**

*root-order-up<sub>i</sub>*  $k\ (T_i\ sub) = root\text{-}order\ k\ sub\ |\$   
*root-order-up<sub>i</sub>*  $k\ (Up_i\ l\ a\ r) = (order\ k\ l \wedge order\ k\ r)$

**fun** *height-up<sub>i</sub>* **where**

*height-up<sub>i</sub>*  $(T_i\ t) = height\ t\ |\$   
*height-up<sub>i</sub>*  $(Up_i\ l\ a\ r) = max\ (height\ l)\ (height\ r)$

**fun** *bal-up<sub>i</sub>* **where**

*bal-up<sub>i</sub>*  $(T_i\ t) = bal\ t\ |\$   
*bal-up<sub>i</sub>*  $(Up_i\ l\ a\ r) = (height\ l = height\ r \wedge bal\ l \wedge bal\ r)$

**fun** *inorder-up<sub>i</sub>* **where**

*inorder-up<sub>i</sub>*  $(T_i\ t) = inorder\ t\ |\$   
*inorder-up<sub>i</sub>*  $(Up_i\ l\ a\ r) = inorder\ l\ @\ [a]\ @\ inorder\ r$

The following function merges two nodes and returns separately split nodes if an overflow occurs

**fun** *node<sub>i</sub>*::  $nat \Rightarrow ('a\ btree \times 'a)\ list \Rightarrow 'a\ btree \Rightarrow 'a\ up_i$  **where**

*node<sub>i</sub>*  $k\ ts\ t = (\$   
*if*  $length\ ts \leq 2*k$  *then*  $T_i\ (Node\ ts\ t)$   
*else*  $($   
*case* *split-half* *ts* *of*  $(ls, (sub, sep)\#rs) \Rightarrow$   
 $Up_i\ (Node\ ls\ sub)\ sep\ (Node\ rs\ t)$   
 $)$   
 $)$

**lemma** *nodei-ti-simp*:  $node_i\ k\ ts\ t = T_i\ x \Longrightarrow x = Node\ ts\ t$   
*<proof>*

**fun** *ins*::  $nat \Rightarrow 'a \Rightarrow 'a\ btree \Rightarrow 'a\ up_i$  **where**

*ins*  $k\ x\ Leaf = (Up_i\ Leaf\ x\ Leaf)\ |\$   
*ins*  $k\ x\ (Node\ ts\ t) = (\$   
*case* *split* *ts* *x* *of*  
 $(ls, (sub, sep)\#rs) \Rightarrow$   
 $(if\ sep = x\ then$   
 $T_i\ (Node\ ts\ t)$   
*else*  
 $(case\ ins\ k\ x\ sub\ of$   
 $Up_i\ l\ a\ r \Rightarrow$   
 $node_i\ k\ (ls\ @\ (l, a)\ \#(r, sep)\ \#rs)\ t\ |\$   
 $T_i\ a \Rightarrow$   
 $T_i\ (Node\ (ls\ @\ (a, sep)\ \#rs)\ t)))\ |\$   
 $(ls, []) \Rightarrow$   
 $(case\ ins\ k\ x\ t\ of$

```

    Upi l a r ⇒
      nodei k (ls@[l,a]) r |
    Ti a ⇒
      Ti (Node ls a)
  )
)

```

```

fun treei::'a upi ⇒ 'a btree where
  treei (Ti sub) = sub |
  treei (Upi l a r) = (Node [l,a] r)

```

```

fun insert::nat ⇒ 'a ⇒ 'a btree ⇒ 'a btree where
  insert k x t = treei (ins k x t)

```

### 3.5 Deletion

The following deletion method is inspired by Bayer (70) and Fielding (80). Rather than stealing only a single node from the neighbour, the neighbour is fully merged with the potentially underflowing node. If the resulting node is still larger than allowed, the merged node is split again, using the rules known from insertion splits. If the resulting node has admissible size, it is simply kept in the tree.

```

fun rebalance-middle-tree where
  rebalance-middle-tree k ls Leaf sep rs Leaf = (
    Node (ls@(Leaf,sep)#rs) Leaf
  ) |
  rebalance-middle-tree k ls (Node mts mt) sep rs (Node tts tt) = (
    if length mts ≥ k ∧ length tts ≥ k then
      Node (ls@(Node mts mt,sep)#rs) (Node tts tt)
    else (
      case rs of [] ⇒ (
        case nodei k (mts@(mt,sep)#tts) tt of
          Ti u ⇒
            Node ls u |
          Upi l a r ⇒
            Node (ls@[l,a]) r |
        (Node rts rt,rsep)#rs ⇒ (
          case nodei k (mts@(mt,sep)#rts) rt of
            Ti u ⇒
              Node (ls@(u,rsep)#rs) (Node tts tt) |
          Upi l a r ⇒
            Node (ls@(l,a)#(r,rsep)#rs) (Node tts tt)
        ))
    ))

```

Deletion

All trees are merged with the right neighbour on underflow. Obviously for

the last tree this would not work since it has no right neighbour. Therefore this tree, as the only exception, is merged with the left neighbour. However since we it does not make a difference, we treat the situation as if the second to last tree underflowed.

```
fun rebalance-last-tree where
  rebalance-last-tree k ts t = (
    case last ts of (sub,sep) ⇒
      rebalance-middle-tree k (butlast ts) sub sep [] t
  )
```

Rather than deleting the minimal key from the right subtree, we remove the maximal key of the left subtree. This is due to the fact that the last tree can easily be accessed and the left neighbour is way easier to access than the right neighbour, it resides in the same pair as the separating element to be removed.

```
fun split-max where
  split-max k (Node ts t) = (case t of Leaf ⇒ (
    let (sub,sep) = last ts in
      (Node (butlast ts) sub, sep)
  )|
  - ⇒
case split-max k t of (sub, sep) ⇒
  (rebalance-last-tree k ts sub, sep)
)
```

```
fun del where
  del k x Leaf = Leaf |
  del k x (Node ts t) = (
    case split ts x of
      (ls,[]) ⇒
        rebalance-last-tree k ls (del k x t)
    | (ls,(sub,sep)#rs) ⇒ (
      if sep ≠ x then
        rebalance-middle-tree k ls (del k x sub) sep rs t
      else if sub = Leaf then
        Node (ls@rs) t
      else let (sub-s, max-s) = split-max k sub in
        rebalance-middle-tree k ls sub-s max-s rs t
    )
  )
```

```
fun reduce-root where
  reduce-root Leaf = Leaf |
  reduce-root (Node ts t) = (case ts of
    [] ⇒ t |
    - ⇒ (Node ts t)
  )
```

**fun** *delete* **where** *delete*  $k\ x\ t = \text{reduce-root } (\text{del } k\ x\ t)$

An invariant for intermediate states at deletion. In particular we allow for an underflow to 0 subtrees.

**fun** *almost-order* **where**  
*almost-order*  $k\ \text{Leaf} = \text{True} \mid$   
*almost-order*  $k\ (\text{Node } ts\ t) = (  
(\text{length } ts \leq 2*k) \wedge$   
 $(\forall s \in \text{set } (\text{subtrees } ts). \text{order } k\ s) \wedge$   
 $\text{order } k\ t$   
 $)$

A recursive property of the "spine" we want to walk along for splitting off the maximum of the left subtree.

**fun** *nonempty-lasttreebal* **where**  
*nonempty-lasttreebal*  $\text{Leaf} = \text{True} \mid$   
*nonempty-lasttreebal*  $(\text{Node } ts\ t) = (  
(\exists ls\ tsub\ tsep. ts = (ls@[tsub,tsep])) \wedge \text{height } tsub = \text{height } t) \wedge$   
 $\text{nonempty-lasttreebal } t$   
 $)$

### 3.6 Proofs of functional correctness

**lemma** *split-set*:

**assumes** *split*  $ts\ z = (ls,(a,b)\#rs)$   
**shows**  $(a,b) \in \text{set } ts$   
**and**  $(x,y) \in \text{set } ls \implies (x,y) \in \text{set } ts$   
**and**  $(x,y) \in \text{set } rs \implies (x,y) \in \text{set } ts$   
**and**  $\text{set } ls \cup \text{set } rs \cup \{(a,b)\} = \text{set } ts$   
**and**  $\exists x \in \text{set } ts. b \in \text{Basic-BNFs.snds } x$   
 $\langle \text{proof} \rangle$

**lemma** *split-length*:

*split*  $ts\ x = (ls, rs) \implies \text{length } ls + \text{length } rs = \text{length } ts$   
 $\langle \text{proof} \rangle$

Isin proof

**thm** *isin-simps*

**lemma** *sorted-ConsD*: *sorted-less*  $(y \# xs) \implies x \leq y \implies x \notin \text{set } xs$   
 $\langle \text{proof} \rangle$

**lemma** *sorted-snocD*: *sorted-less*  $(xs @ [y]) \implies y \leq x \implies x \notin \text{set } xs$   
 $\langle \text{proof} \rangle$

**lemmas** *isin-simps2* = *sorted-lems sorted-ConsD sorted-snocD*

**lemma** *isin-sorted: sorted-less*  $(xs@a\#ys) \implies$   
 $(x \in \text{set } (xs@a\#ys)) = (\text{if } x < a \text{ then } x \in \text{set } xs \text{ else } x \in \text{set } (a\#ys))$   
 $\langle \text{proof} \rangle$

**lemma** *isin-sorted-split:*  
**assumes** *sorted-less*  $(\text{inorder } (\text{Node } ts \ t))$   
**and** *split*  $ts \ x = (ls, rs)$   
**shows**  $x \in \text{set } (\text{inorder } (\text{Node } ts \ t)) = (x \in \text{set } (\text{inorder-list } rs \ @ \ \text{inorder } t))$   
 $\langle \text{proof} \rangle$

**lemma** *isin-sorted-split-right:*  
**assumes** *split*  $ts \ x = (ls, (sub,sep)\#rs)$   
**and** *sorted-less*  $(\text{inorder } (\text{Node } ts \ t))$   
**and**  $sep \neq x$   
**shows**  $x \in \text{set } (\text{inorder-list } ((sub,sep)\#rs) \ @ \ \text{inorder } t) = (x \in \text{set } (\text{inorder } sub))$   
 $\langle \text{proof} \rangle$

**theorem** *isin-set-inorder: sorted-less*  $(\text{inorder } t) \implies \text{isin } t \ x = (x \in \text{set } (\text{inorder } t))$   
 $\langle \text{proof} \rangle$

**lemma** *node<sub>i</sub>-cases:*  $\text{length } xs \leq k \vee (\exists ls \ sub \ sep \ rs. \text{split-half } xs = (ls, (sub,sep)\#rs))$   
 $\langle \text{proof} \rangle$

**lemma** *root-order-tree<sub>i</sub>:*  $\text{root-order-up}_i \ (Suc \ k) \ t = \text{root-order } (Suc \ k) \ (tree_i \ t)$   
 $\langle \text{proof} \rangle$

**lemma** *node<sub>i</sub>-root-order:*  
**assumes**  $\text{length } ts > 0$   
**and**  $\text{length } ts \leq 4*k+1$   
**and**  $\forall x \in \text{set } (\text{subtrees } ts). \ \text{order } k \ x$   
**and**  $\text{order } k \ t$   
**shows**  $\text{root-order-up}_i \ k \ (\text{node}_i \ k \ ts \ t)$   
 $\langle \text{proof} \rangle$

**lemma** *node<sub>i</sub>-order-helper:*  
**assumes**  $\text{length } ts \geq k$   
**and**  $\text{length } ts \leq 4*k+1$   
**and**  $\forall x \in \text{set } (\text{subtrees } ts). \ \text{order } k \ x$



**and**  $order\ k\ t$   
**shows**  $case\ (node_i\ k\ ts\ t)\ of\ T_i\ t\ \Rightarrow\ order\ k\ t\ |\ -\ \Rightarrow\ True$   
 $\langle proof \rangle$

**lemma**  $node_i\text{-order}$ :  
**assumes**  $length\ ts\ \geq\ k$   
**and**  $length\ ts\ \leq\ 4*k+1$   
**and**  $\forall x \in set\ (subtrees\ ts).\ order\ k\ x$   
**and**  $order\ k\ t$   
**shows**  $order\text{-up}_i\ k\ (node_i\ k\ ts\ t)$   
 $\langle proof \rangle$

**lemma**  $ins\text{-order}$ :  
 $order\ k\ t\ \Longrightarrow\ order\text{-up}_i\ k\ (ins\ k\ x\ t)$   
 $\langle proof \rangle$

**lemma**  $ins\text{-root}\text{-order}$ :  
**assumes**  $root\text{-order}\ k\ t$   
**shows**  $root\text{-order}\text{-up}_i\ k\ (ins\ k\ x\ t)$   
 $\langle proof \rangle$

**lemma**  $height\text{-list}\text{-split}$ :  $height\text{-up}_i\ (Up_i\ (Node\ ls\ a)\ b\ (Node\ rs\ t)) = height\ (Node\ (ls@ (a,b)\#rs)\ t)$   
 $\langle proof \rangle$

**lemma**  $node_i\text{-height}$ :  $height\text{-up}_i\ (node_i\ k\ ts\ t) = height\ (Node\ ts\ t)$   
 $\langle proof \rangle$

**lemma**  $bal\text{-up}_i\text{-tree}$ :  $bal\text{-up}_i\ t = bal\ (tree_i\ t)$   
 $\langle proof \rangle$

**lemma**  $bal\text{-list}\text{-split}$ :  $bal\ (Node\ (ls@ (a,b)\#rs)\ t) \Longrightarrow bal\text{-up}_i\ (Up_i\ (Node\ ls\ a)\ b\ (Node\ rs\ t))$   
 $\langle proof \rangle$

**lemma**  $node_i\text{-bal}$ :  
**assumes**  $bal\ (Node\ ts\ t)$   
**shows**  $bal\text{-up}_i\ (node_i\ k\ ts\ t)$   
 $\langle proof \rangle$

**lemma** *height-up<sub>i</sub>-merge*:  $\text{height-up}_i (Up_i l a r) = \text{height } t \implies \text{height } (\text{Node } (ls @ (t, x) \# rs) tt) = \text{height } (\text{Node } (ls @ (l, a) \# (r, x) \# rs) tt)$   
 ⟨proof⟩

**lemma** *ins-height*:  $\text{height-up}_i (\text{ins } k x t) = \text{height } t$   
 ⟨proof⟩

**lemma** *ins-bal*:  $\text{bal } t \implies \text{bal-up}_i (\text{ins } k x t)$   
 ⟨proof⟩

**lemma** *node<sub>i</sub>-inorder*:  $\text{inorder-up}_i (\text{node}_i k ts t) = \text{inorder } (\text{Node } ts t)$   
 ⟨proof⟩

**corollary** *node<sub>i</sub>-inorder-simps*:  
 $\text{node}_i k ts t = T_i t' \implies \text{inorder } t' = \text{inorder } (\text{Node } ts t)$   
 $\text{node}_i k ts t = Up_i l a r \implies \text{inorder } l @ a \# \text{inorder } r = \text{inorder } (\text{Node } ts t)$   
 ⟨proof⟩

**lemma** *ins-sorted-inorder*:  $\text{sorted-less } (\text{inorder } t) \implies (\text{inorder-up}_i (\text{ins } k (x :: ('a :: \text{linorder}))) t) = \text{ins-list } x (\text{inorder } t)$   
 ⟨proof⟩

**lemma** *ins-list-split*:  
**assumes**  $\text{split } ts x = (ls, rs)$   
**and**  $\text{sorted-less } (\text{inorder } (\text{Node } ts t))$   
**shows**  $\text{ins-list } x (\text{inorder } (\text{Node } ts t)) = \text{inorder-list } ls @ \text{ins-list } x (\text{inorder-list } rs @ \text{inorder } t)$   
 ⟨proof⟩

**lemma** *ins-list-split-right-general*:  
**assumes**  $\text{split } ts x = (ls, (sub, sep) \# rs)$   
**and**  $\text{sorted-less } (\text{inorder-list } ts)$   
**and**  $sep \neq x$   
**shows**  $\text{ins-list } x (\text{inorder-list } ((sub, sep) \# rs) @ zs) = \text{ins-list } x (\text{inorder } sub) @ sep \# \text{inorder-list } rs @ zs$   
 ⟨proof⟩

**corollary** *ins-list-split-right*:  
**assumes**  $\text{split } ts x = (ls, (sub, sep) \# rs)$

**and** *sorted-less* (*inorder* (*Node ts t*))  
**and** *sep*  $\neq$  *x*  
**shows** *ins-list* *x* (*inorder-list* ((*sub,sep*)#*rs*) @ *inorder t*) = *ins-list* *x* (*inorder* *sub*) @ *sep* # *inorder-list* *rs* @ *inorder t*  
 <*proof*>

**lemma** *ins-list-idem-eq-isin*: *sorted-less xs*  $\implies$   $x \in \text{set } xs \iff (\text{ins-list } x \text{ } xs = xs)$   
 <*proof*>

**lemma** *ins-list-contains-idem*:  $\llbracket \text{sorted-less } xs; x \in \text{set } xs \rrbracket \implies (\text{ins-list } x \text{ } xs = xs)$   
 <*proof*>

**declare** *node<sub>i</sub>.simps* [*simp del*]  
**declare** *node<sub>i</sub>-inorder* [*simp add*]

**lemma** *ins-inorder*: *sorted-less* (*inorder t*)  $\implies$  (*inorder-up<sub>i</sub>* (*ins k x t*)) = *ins-list* *x* (*inorder t*)  
 <*proof*>

**declare** *node<sub>i</sub>.simps* [*simp add*]  
**declare** *node<sub>i</sub>-inorder* [*simp del*]

**thm** *ins.induct*  
**thm** *btree.induct*

**lemma** *tree<sub>i</sub>-bal*: *bal-up<sub>i</sub>* *u*  $\implies$  *bal* (*tree<sub>i</sub>* *u*)  
 <*proof*>

**lemma** *tree<sub>i</sub>-order*:  $\llbracket k > 0; \text{root-order-up}_i k u \rrbracket \implies \text{root-order } k (\text{tree}_i u)$   
 <*proof*>

**lemma** *tree<sub>i</sub>-inorder*: *inorder-up<sub>i</sub>* *u* = *inorder* (*tree<sub>i</sub>* *u*)  
 <*proof*>

**lemma** *insert-bal*: *bal t*  $\implies$  *bal* (*insert k x t*)  
 <*proof*>

**lemma** *insert-order*:  $\llbracket k > 0; \text{root-order } k t \rrbracket \implies \text{root-order } k (\text{insert } k \text{ } x \text{ } t)$   
 <*proof*>

**lemma** *insert-inorder*: *sorted-less* (*inorder t*)  $\implies$  *inorder* (*insert k x t*) = *ins-list* *x* (*inorder t*)

*<proof>*

Deletion proofs

**thm** *list.simps*

**lemma** *rebalance-middle-tree-height:*

**assumes** *height t = height sub*

**and** *case rs of (rsub,rsep) # list  $\Rightarrow$  height rsub = height t | []  $\Rightarrow$  True*

**shows** *height (rebalance-middle-tree k ls sub sep rs t) = height (Node (ls@(sub,sep)#rs) t)*

*<proof>*

**lemma** *rebalance-last-tree-height:*

**assumes** *height t = height sub*

**and** *ts = list@[sub,sep]*

**shows** *height (rebalance-last-tree k ts t) = height (Node ts t)*

*<proof>*

**lemma** *split-max-height:*

**assumes** *split-max k t = (sub,sep)*

**and** *nonempty-lasttreebal t*

**and** *t  $\neq$  Leaf*

**shows** *height sub = height t*

*<proof>*

**lemma** *order-bal-nonempty-lasttreebal:*  $\llbracket k > 0; \text{root-order } k \ t; \text{bal } t \rrbracket \Longrightarrow \text{nonempty-lasttreebal } t$

*<proof>*

**lemma** *bal-sub-height:*  $\text{bal } (\text{Node } (ls@a\#rs) \ t) \Longrightarrow (\text{case } rs \text{ of } [] \Rightarrow \text{True} \mid (sub,sep)\#- \Rightarrow \text{height } sub = \text{height } t)$

*<proof>*

**lemma** *del-height:*  $\llbracket k > 0; \text{root-order } k \ t; \text{bal } t \rrbracket \Longrightarrow \text{height } (\text{del } k \ x \ t) = \text{height } t$

*<proof>*

**lemma** *rebalance-middle-tree-inorder:*

**assumes** *height t = height sub*

**and** *case rs of (rsub,rsep) # list  $\Rightarrow$  height rsub = height t | []  $\Rightarrow$  True*

**shows** *inorder (rebalance-middle-tree k ls sub sep rs t) = inorder (Node (ls@(sub,sep)#rs) t)*

*<proof>*

**lemma** *rebalance-last-tree-inorder*:

**assumes**  $height\ t = height\ sub$

**and**  $ts = list@[sub,sep]$

**shows**  $inorder\ (rebalance-last-tree\ k\ ts\ t) = inorder\ (Node\ ts\ t)$

*<proof>*

**lemma** *butlast-inorder-app-id*:  $xs = xs' @ [(sub,sep)] \implies inorder-list\ xs' @ inorder$

$sub\ @\ [sep] = inorder-list\ xs$

*<proof>*

**lemma** *split-max-inorder*:

**assumes** *nonempty-lasttreebal*  $t$

**and**  $t \neq Leaf$

**shows**  $inorder-pair\ (split-max\ k\ t) = inorder\ t$

*<proof>*

**lemma** *height-bal-subtrees-merge*:  $\llbracket height\ (Node\ as\ a) = height\ (Node\ bs\ b); bal$

$(Node\ as\ a); bal\ (Node\ bs\ b) \rrbracket$

$\implies \forall x \in set\ (subtrees\ as) \cup \{a\}. height\ x = height\ b$

*<proof>*

**lemma** *bal-list-merge*:

**assumes**  $bal-up_i\ (Up_i\ (Node\ as\ a)\ x\ (Node\ bs\ b))$

**shows**  $bal\ (Node\ (as@[a,x])\ #bs)\ b$

*<proof>*

**lemma** *node<sub>i</sub>-bal-up<sub>i</sub>*:

**assumes**  $bal-up_i\ (node_i\ k\ ts\ t)$

**shows**  $bal\ (Node\ ts\ t)$

*<proof>*

**lemma** *node<sub>i</sub>-bal-simp*:  $bal-up_i\ (node_i\ k\ ts\ t) = bal\ (Node\ ts\ t)$

*<proof>*

**lemma** *rebalance-middle-tree-bal*:  $bal\ (Node\ (ls@[sub,sep])\ #rs)\ t \implies bal\ (rebalance-middle-tree$

$k\ ls\ sub\ sep\ rs\ t)$

*<proof>*

**lemma** *rebalance-last-tree-bal*:  $\llbracket bal\ (Node\ ts\ t); ts \neq [] \rrbracket \implies bal\ (rebalance-last-tree$

$k\ ts\ t)$

*<proof>*

**lemma** *split-max-bal*:

**assumes**  $bal\ t$

**and**  $t \neq Leaf$

**and** *nonempty-lasttreebal*  $t$

**shows**  $bal (fst (split-max k t))$   
 $\langle proof \rangle$

**lemma** *del-bal*:  
**assumes**  $k > 0$   
**and**  $root-order k t$   
**and**  $bal t$   
**shows**  $bal (del k x t)$   
 $\langle proof \rangle$

**lemma** *rebalance-middle-tree-order*:  
**assumes**  $almost-order k sub$   
**and**  $\forall s \in set (subtrees (ls@rs)). order k s order k t$   
**and**  $case rs of (rsub,rsep) \# list \Rightarrow height rsub = height t \mid [] \Rightarrow True$   
**and**  $length (ls@(sub,sep)\#rs) \leq 2*k$   
**and**  $height sub = height t$   
**shows**  $almost-order k (rebalance-middle-tree k ls sub sep rs t)$   
 $\langle proof \rangle$

**lemma** *rebalance-middle-tree-last-order*:  
**assumes**  $almost-order k t$   
**and**  $\forall s \in set (subtrees (ls@(sub,sep)\#rs)). order k s$   
**and**  $rs = []$   
**and**  $length (ls@(sub,sep)\#rs) \leq 2*k$   
**and**  $height sub = height t$   
**shows**  $almost-order k (rebalance-middle-tree k ls sub sep rs t)$   
 $\langle proof \rangle$

**lemma** *rebalance-last-tree-order*:  
**assumes**  $ts = ls@[sub,sep]$   
**and**  $\forall s \in set (subtrees (ts)). order k s almost-order k t$   
**and**  $length ts \leq 2*k$   
**and**  $height sub = height t$   
**shows**  $almost-order k (rebalance-last-tree k ts t)$   
 $\langle proof \rangle$

**lemma** *split-max-order*:  
**assumes**  $order k t$   
**and**  $t \neq Leaf$   
**and**  $nonempty-lasttreebal t$   
**shows**  $almost-order k (fst (split-max k t))$   
 $\langle proof \rangle$

**lemma** *del-order*:  
**assumes**  $k > 0$   
**and**  $root-order k t$

**and** *bal t*  
**shows** *almost-order k (del k x t)*  
*<proof>*

**thm** *del-list-sorted*

**lemma** *del-list-split:*

**assumes** *split ts x = (ls, rs)*  
**and** *sorted-less (inorder (Node ts t))*  
**shows** *del-list x (inorder (Node ts t)) = inorder-list ls @ del-list x (inorder-list rs @ inorder t)*  
*<proof>*

**lemma** *del-list-split-right:*

**assumes** *split ts x = (ls, (sub,sep)#rs)*  
**and** *sorted-less (inorder (Node ts t))*  
**and** *sep ≠ x*  
**shows** *del-list x (inorder-list ((sub,sep)#rs) @ inorder t) = del-list x (inorder sub) @ sep # inorder-list rs @ inorder t*  
*<proof>*

**thm** *del-list-idem*

**lemma** *del-inorder:*

**assumes** *k > 0*  
**and** *root-order k t*  
**and** *bal t*  
**and** *sorted-less (inorder t)*  
**shows** *inorder (del k x t) = del-list x (inorder t)*  
*<proof>*

**lemma** *reduce-root-order:  $\llbracket k > 0; \text{almost-order } k \ t \rrbracket \implies \text{root-order } k \ (\text{reduce-root } t)$*   
*<proof>*

**lemma** *reduce-root-bal: bal (reduce-root t) = bal t*  
*<proof>*

**lemma** *reduce-root-inorder: inorder (reduce-root t) = inorder t*  
*<proof>*

**lemma** *delete-order:  $\llbracket k > 0; \text{bal } t; \text{root-order } k \ t \rrbracket \implies \text{root-order } k \ (\text{delete } k \ x \ t)$*   
*<proof>*

**lemma** *delete-bal*:  $\llbracket k > 0; \text{bal } t; \text{root-order } k \ t \rrbracket \implies \text{bal } (\text{delete } k \ x \ t)$   
 ⟨proof⟩

**lemma** *delete-inorder*:  $\llbracket k > 0; \text{bal } t; \text{root-order } k \ t; \text{sorted-less } (\text{inorder } t) \rrbracket \implies$   
 $\text{inorder } (\text{delete } k \ x \ t) = \text{del-list } x \ (\text{inorder } t)$   
 ⟨proof⟩

### 3.7 Set specification by inorder

**interpretation** *S-ordered*: *Set-by-Ordered* **where**

*empty* = *empty-btree* **and**  
*insert* = *insert* (*Suc* *k*) **and**  
*delete* = *delete* (*Suc* *k*) **and**  
*isin* = *isin* **and**  
*inorder* = *inorder* **and**  
*inv* = *invar-inorder* (*Suc* *k*)  
 ⟨proof⟩

**declare** *node<sub>i</sub>.simps*[*simp del*]

**end**

**end**

**theory** *BTree-Split*  
**imports** *BTree-Set*  
**begin**

## 4 Abstract split functions

### 4.1 Linear split

Finally we show that the split axioms are feasible by providing an example split function

**fun** *linear-split-help*::  $(-\times'a::\text{linorder}) \text{ list} \Rightarrow - \Rightarrow (-\times-) \text{ list} \Rightarrow ((-\times-) \text{ list} \times (-\times-) \text{ list})$  **where**

*linear-split-help* [] *x prev* = (*prev*, []) |  
*linear-split-help* ((*sub*, *sep*)#*xs*) *x prev* = (if *sep* < *x* then *linear-split-help* *xs* *x* (*prev* @ [(*sub*, *sep*)]) else (*prev*, (*sub,sep*)#*xs*))

**fun** *linear-split*::  $(-\times'a::\text{linorder}) \text{ list} \Rightarrow - \Rightarrow ((-\times-) \text{ list} \times (-\times-) \text{ list})$  **where**  
*linear-split* *xs* *x* = *linear-split-help* *xs* *x* []

Linear split is similar to well known functions, therefore a quick proof can be done.



**lemma** *linear-split-alt*:  $linear-split\ xs\ x = (takeWhile\ (\lambda(-,s).\ s < x)\ xs,\ dropWhile\ (\lambda(-,s).\ s < x)\ xs)$   
 ⟨proof⟩

**global-interpretation** *btree-linear-search*: *split linear-split*

**defines** *btree-ls-isin* = *btree-linear-search.isin*  
**and** *btree-ls-ins* = *btree-linear-search.ins*  
**and** *btree-ls-insert* = *btree-linear-search.insert*  
**and** *btree-ls-del* = *btree-linear-search.del*  
**and** *btree-ls-delete* = *btree-linear-search.delete*  
 ⟨proof⟩

Some examples follow to show that the implementation works and the above lemmas make sense. The examples are visualized in the thesis.

**abbreviation**  $btree_i \equiv btree-ls-insert$

**abbreviation**  $btree_d \equiv btree-ls-delete$

**value**  $let\ k=2::nat;\ x::nat\ btree = (Node\ [(Node\ [(Leaf,\ 3),(Leaf,\ 5),(Leaf,\ 6)]\ Leaf,\ 10)]\ (Node\ [(Leaf,\ 14),\ (Leaf,\ 20)]\ Leaf))\ in$   
*root-order*  $k\ x$

**value**  $let\ k=2::nat;\ x::nat\ btree = (Node\ [(Node\ [(Leaf,\ 3),(Leaf,\ 5),(Leaf,\ 6)]\ Leaf,\ 10)]\ (Node\ [(Leaf,\ 14),\ (Leaf,\ 20)]\ Leaf))\ in$   
*bal*  $x$

**value**  $let\ k=2::nat;\ x::nat\ btree = (Node\ [(Node\ [(Leaf,\ 3),(Leaf,\ 5),(Leaf,\ 6)]\ Leaf,\ 10)]\ (Node\ [(Leaf,\ 14),\ (Leaf,\ 20)]\ Leaf))\ in$   
*sorted-less* (*inorder*  $x$ )

**value**  $let\ k=2::nat;\ x::nat\ btree = (Node\ [(Node\ [(Leaf,\ 3),(Leaf,\ 5),(Leaf,\ 6)]\ Leaf,\ 10)]\ (Node\ [(Leaf,\ 14),\ (Leaf,\ 20)]\ Leaf))\ in$   
 $x$

**value**  $let\ k=2::nat;\ x::nat\ btree = (Node\ [(Node\ [(Leaf,\ 3),(Leaf,\ 5),(Leaf,\ 6)]\ Leaf,\ 10)]\ (Node\ [(Leaf,\ 14),\ (Leaf,\ 20)]\ Leaf))\ in$   
*btree\_i*  $k\ 9\ x$

**value**  $let\ k=2::nat;\ x::nat\ btree = (Node\ [(Node\ [(Leaf,\ 3),(Leaf,\ 5),(Leaf,\ 6)]\ Leaf,\ 10)]\ (Node\ [(Leaf,\ 14),\ (Leaf,\ 20)]\ Leaf))\ in$   
*btree\_i*  $k\ 1\ (btree_i\ k\ 9\ x)$

**value**  $let\ k=2::nat;\ x::nat\ btree = (Node\ [(Node\ [(Leaf,\ 3),(Leaf,\ 5),(Leaf,\ 6)]\ Leaf,\ 10)]\ (Node\ [(Leaf,\ 14),\ (Leaf,\ 20)]\ Leaf))\ in$   
*btree\_d*  $k\ 10\ (btree_i\ k\ 1\ (btree_i\ k\ 9\ x))$

**value**  $let\ k=2::nat;\ x::nat\ btree = (Node\ [(Node\ [(Leaf,\ 3),(Leaf,\ 5),(Leaf,\ 6)]\ Leaf,\ 10)]\ (Node\ [(Leaf,\ 14),\ (Leaf,\ 20)]\ Leaf))\ in$   
*btree\_d*  $k\ 3\ (btree_d\ k\ 10\ (btree_i\ k\ 1\ (btree_i\ k\ 9\ x)))$

For completeness, we also proved an explicit proof of the locale requirements.

**lemma** *some-child-sm*:  $linear-split-help\ t\ y\ xs = (ls,(sub,sep)\#rs) \implies y \leq sep$   
 ⟨proof⟩

**lemma** *linear-split-append*:  $linear-split-help\ xs\ p\ ys = (ls,rs) \implies ls@rs = ys@xs$   
 ⟨proof⟩

**lemma** *linear-split-sm*:  $\llbracket linear-split-help\ xs\ p\ ys = (ls,rs); sorted-less\ (separators\ (ys@xs)); \forall sep \in set\ (separators\ ys). p > sep \rrbracket \implies \forall sep \in set\ (separators\ ls). p > sep$   
 ⟨proof⟩

**value** *linear-split*  $\llbracket (Leaf::nat\ btree), 2 \rrbracket (1::nat)$

**lemma** *linear-split-gr*:  
 $\llbracket linear-split-help\ xs\ p\ ys = (ls,rs); sorted-less\ (separators\ (ys@xs)); \forall (sub,sep) \in set\ ys. p > sep \rrbracket \implies$   
 $(case\ rs\ of\ [] \Rightarrow True \mid (-,sep)\#- \Rightarrow p \leq sep)$   
 ⟨proof⟩

**lemma** *linear-split-req*:  
**assumes**  $linear-split\ xs\ p = (ls,(sub,sep)\#rs)$   
**and**  $sorted-less\ (separators\ xs)$   
**shows**  $p \leq sep$   
 ⟨proof⟩

**lemma** *linear-split-req2*:  
**assumes**  $linear-split\ xs\ p = (ls@[sub,sep],rs)$   
**and**  $sorted-less\ (separators\ xs)$   
**shows**  $sep < p$   
 ⟨proof⟩

**interpretation** *split linear-split*  
 ⟨proof⟩

## 4.2 Binary split

It is possible to define a binary split predicate. However, even proving that it terminates is uncomfortable.

**function** (*sequential*) *binary-split-help*:  $(-\times'a::linorder)\ list \Rightarrow (-\times'a)\ list \Rightarrow (-\times'a)\ list \Rightarrow 'a \Rightarrow ((-\times-) list \times (-\times-) list)$  **where**  
 $binary-split-help\ ls\ []\ rs\ x = (ls,rs) \mid$   
 $binary-split-help\ ls\ as\ rs\ x = (let\ (mls, mrs) = split-half\ as\ in (\$   
 $case\ mrs\ of\ (sub,sep)\#mrrs \Rightarrow (\$   
 $if\ x < sep\ then\ binary-split-help\ ls\ mls\ (mrs@rs)\ x$   
 $else\ if\ x > sep\ then\ binary-split-help\ (ls@mrs@[sub,sep])\ mrrs\ rs\ x$   
 $else\ (ls@mrs, mrs@rs)$   
 $)$   
 $)$   
 $)$

```

    <proof>
termination
    <proof>

```

```

fun binary-split where
    binary-split as x = binary-split-help [] as [] x

```

We can show that it will return sublists that concatenate to the original list again but will not show that it fulfils sortedness properties.

```

lemma binary-split-help as bs cs x = (ls,rs)  $\implies$  (as@bs@cs) = (ls@rs)
    <proof>

```

```

lemma [sorted-less (separators (as@bs@cs)); binary-split-help as bs cs x = (ls,rs);
 $\forall y \in \text{set } (\text{separators } \textit{as}). y < x$ ]
 $\implies \forall y \in \text{set } (\text{separators } \textit{ls}). y < x$ 
    <proof>

```

```

end
theory BPlusTree
    imports Main HOL-Data-Structures.Sorted-Less HOL-Data-Structures.Cmp
    HOL-Library.Multiset
begin

```

```

hide-const (open) Sorted-Less.sorted
abbreviation sorted-less  $\equiv$  Sorted-Less.sorted

```

## 5 Definition of the B-Plus-Tree

### 5.1 Datatype definition

B-Plus-Trees are basically B-Trees, that don't have empty Leafs but Leafs that contain the relevant data.

```

datatype 'a bplustree = Leaf (vals: 'a list) | Node (keyvals: ('a bplustree * 'a) list)
    (lasttree: 'a bplustree)

```

```

type-synonym 'a bplustree-list = ('a bplustree * 'a) list
type-synonym 'a bplustree-pair = ('a bplustree * 'a)

```

```

abbreviation subtrees where subtrees xs  $\equiv$  (map fst xs)
abbreviation separators where separators xs  $\equiv$  (map snd xs)

```

## 5.2 Inorder and Set

The set of B-Plus-tree needs to be manually defined, regarding only the leaves. This overrides the default instantiation.

```
fun set-nodes :: 'a bplustree  $\Rightarrow$  'a set where
  set-nodes (Leaf ks) = {} |
  set-nodes (Node ts t) =  $\bigcup$ (set (map set-nodes (subtrees ts)))  $\cup$  (set (separators
ts))  $\cup$  set-nodes t
```

```
fun set-leaves :: 'a bplustree  $\Rightarrow$  'a set where
  set-leaves (Leaf ks) = set ks |
  set-leaves (Node ts t) =  $\bigcup$ (set (map set-leaves (subtrees ts)))  $\cup$  set-leaves t
```

The inorder is a view of only internal separators

```
fun inorder :: 'a bplustree  $\Rightarrow$  'a list where
  inorder (Leaf ks) = [] |
  inorder (Node ts t) = concat (map ( $\lambda$  (sub, sep). inorder sub @ [sep]) ts) @
inorder t
```

```
abbreviation inorder-list ts  $\equiv$  concat (map ( $\lambda$  (sub, sep). inorder sub @ [sep]) ts)
```

The leaves view considers only its leafs.

```
fun leaves :: 'a bplustree  $\Rightarrow$  'a list where
  leaves (Leaf ks) = ks |
  leaves (Node ts t) = concat (map leaves (subtrees ts)) @ leaves t
```

```
abbreviation leaves-list ts  $\equiv$  concat (map leaves (subtrees ts))
```

```
fun leaf-nodes where
  leaf-nodes (Leaf xs) = [Leaf xs] |
  leaf-nodes (Node ts t) = concat (map leaf-nodes (subtrees ts)) @ leaf-nodes t
```

```
abbreviation leaf-nodes-list ts  $\equiv$  concat (map leaf-nodes (subtrees ts))
```

And the elems view contains all elements of the tree

```
fun elems :: 'a bplustree  $\Rightarrow$  'a list where
  elems (Leaf ks) = ks |
  elems (Node ts t) = concat (map ( $\lambda$  (sub, sep). elems sub @ [sep]) ts) @ elems t
```

```
abbreviation elems-list ts  $\equiv$  concat (map ( $\lambda$  (sub, sep). elems sub @ [sep]) ts)
```

```
thm leaves.simps
thm inorder.simps
thm elems.simps
```

```
value leaves (Node [(Leaf [], (0::nat)), (Node [(Leaf [], 1), (Leaf [], 10)] (Leaf []),
12), ((Leaf [], 30), ((Leaf [], 100)] (Leaf []))
```

### 5.3 Height and Balancedness

```

class height =
  fixes height :: 'a ⇒ nat

instantiation bplustree :: (type) height
begin

fun height-bplustree :: 'a bplustree ⇒ nat where
  height (Leaf ks) = 0 |
  height (Node ts t) = Suc (Max (height ` (set (subtrees ts@[t])))

instance ⟨proof⟩

end

```

Balancedness is defined in close accordance to the definition by Ernst

```

fun bal:: 'a bplustree ⇒ bool where
  bal (Leaf ks) = True |
  bal (Node ts t) = (
    (∀ sub ∈ set (subtrees ts). height sub = height t) ∧
    (∀ sub ∈ set (subtrees ts). bal sub) ∧ bal t
  )

value height (Node [(Leaf [], (0::nat)), (Node [(Leaf [], 1), (Leaf [], 10)] (Leaf []),
12), ((Leaf [], 30), ((Leaf [], 100)] (Leaf []))
value bal (Node [(Leaf [], (0::nat)), (Node [(Leaf [], 1), (Leaf [], 10)] (Leaf []),
12), ((Leaf [], 30), ((Leaf [], 100)] (Leaf []))

```

### 5.4 Order

The order of a B-tree is defined just as in the original paper by Bayer.

```

fun order:: nat ⇒ 'a bplustree ⇒ bool where
  order k (Leaf ks) = ((length ks ≥ k) ∧ (length ks ≤ 2*k)) |
  order k (Node ts t) = (
    (length ts ≥ k) ∧
    (length ts ≤ 2*k) ∧
    (∀ sub ∈ set (subtrees ts). order k sub) ∧ order k t
  )

```

The special condition for the root is called *root\_order*

```

fun root-order:: nat ⇒ 'a bplustree ⇒ bool where
  root-order k (Leaf ks) = (length ks ≤ 2*k) |
  root-order k (Node ts t) = (
    (length ts > 0) ∧
    (length ts ≤ 2*k) ∧
    (∀ s ∈ set (subtrees ts). order k s) ∧ order k t
  )

```

## 5.5 Auxiliary Lemmas

**lemma** *separators-split*:

$$\text{set } (\text{separators } (l@(\text{a},\text{b})\#r)) = \text{set } (\text{separators } l) \cup \text{set } (\text{separators } r) \cup \{\text{b}\}$$

*<proof>*

**lemma** *subtrees-split*:

$$\text{set } (\text{subtrees } (l@(\text{a},\text{b})\#r)) = \text{set } (\text{subtrees } l) \cup \text{set } (\text{subtrees } r) \cup \{\text{a}\}$$

*<proof>*

**lemma** *finite-set-ins-swap*:

**assumes** *finite A*

$$\text{shows } \max \text{ a } (\text{Max } (\text{Set.insert } \text{b } A)) = \max \text{ b } (\text{Max } (\text{Set.insert } \text{a } A))$$

*<proof>*

**lemma** *finite-set-in-idem*:

**assumes** *finite A*

$$\text{shows } \max \text{ a } (\text{Max } (\text{Set.insert } \text{a } A)) = \text{Max } (\text{Set.insert } \text{a } A)$$

*<proof>*

**lemma** *height-Leaf*:  $\text{height } t = 0 \longleftrightarrow (\exists \text{ ks. } t = (\text{Leaf } \text{ks}))$

*<proof>*

**lemma** *height-bplustree-order*:

$$\text{height } (\text{Node } (\text{ls}@[\text{a}]) t) = \text{height } (\text{Node } (\text{a}\#\text{ls}) t)$$

*<proof>*

**lemma** *height-bplustree-sub*:

$$\text{height } (\text{Node } ((\text{sub},\text{x})\#\text{ls}) t) = \max (\text{height } (\text{Node } \text{ls } t)) (\text{Suc } (\text{height } \text{sub}))$$

*<proof>*

**lemma** *height-bplustree-last*:

$$\text{height } (\text{Node } ((\text{sub},\text{x})\#\text{ts}) t) = \max (\text{height } (\text{Node } \text{ts } \text{sub})) (\text{Suc } (\text{height } t))$$

*<proof>*

**lemma** *set-leaves-leaves*:  $\text{set } (\text{leaves } t) = \text{set-leaves } t$

*<proof>*

**lemma** *set-nodes-nodes*:  $\text{set } (\text{inorder } t) = \text{set-nodes } t$

*<proof>*

**lemma** *child-subset-leaves*:  $p \in \text{set } t \implies \text{set-leaves } (\text{fst } p) \subseteq \text{set-leaves } (\text{Node } t \text{ n})$

*<proof>*

**lemma** *child-subset*:  $p \in \text{set } t \implies \text{set-nodes } (\text{fst } p) \subseteq \text{set-nodes } (\text{Node } t \text{ n})$

$\langle proof \rangle$

**lemma** *some-child-sub*:

**assumes**  $(sub, sep) \in set\ t$   
**shows**  $sub \in set\ (subtrees\ t)$   
**and**  $sep \in set\ (separators\ t)$   
 $\langle proof \rangle$

**lemma** *bal-all-subtrees-equal*:  $bal\ (Node\ ts\ t) \implies (\forall s1 \in set\ (subtrees\ ts). \forall s2 \in set\ (subtrees\ ts). height\ s1 = height\ s2)$   
 $\langle proof \rangle$

**lemma** *fold-max-set*:  $\forall x \in set\ t. x = f \implies fold\ max\ t\ f = f$   
 $\langle proof \rangle$

**lemma** *height-bal-tree*:  $bal\ (Node\ ts\ t) \implies height\ (Node\ ts\ t) = Suc\ (height\ t)$   
 $\langle proof \rangle$

**lemma** *bal-split-last*:

**assumes**  $bal\ (Node\ (ls@(sub,sep)\#rs)\ t)$   
**shows**  $bal\ (Node\ (ls@rs)\ t)$   
**and**  $height\ (Node\ (ls@(sub,sep)\#rs)\ t) = height\ (Node\ (ls@rs)\ t)$   
 $\langle proof \rangle$

**lemma** *bal-split-right*:

**assumes**  $bal\ (Node\ (ls@rs)\ t)$   
**shows**  $bal\ (Node\ rs\ t)$   
**and**  $height\ (Node\ rs\ t) = height\ (Node\ (ls@rs)\ t)$   
 $\langle proof \rangle$

**lemma** *bal-split-left*:

**assumes**  $bal\ (Node\ (ls@(a,b)\#rs)\ t)$   
**shows**  $bal\ (Node\ ls\ a)$   
**and**  $height\ (Node\ ls\ a) = height\ (Node\ (ls@(a,b)\#rs)\ t)$   
 $\langle proof \rangle$

**lemma** *bal-substitute*:  $\llbracket bal\ (Node\ (ls@(a,b)\#rs)\ t); height\ t = height\ c; bal\ c \rrbracket \implies bal\ (Node\ (ls@(c,b)\#rs)\ t)$   
 $\langle proof \rangle$

**lemma** *bal-substitute-subtree*:  $\llbracket bal\ (Node\ (ls@(a,b)\#rs)\ t); height\ a = height\ c; bal$

$c \llbracket \implies \text{bal } (\text{Node } (\text{ls}@ (c,b) \#rs) t)$   
 $\langle \text{proof} \rangle$

**lemma** *bal-substitute-separator*:  $\text{bal } (\text{Node } (\text{ls}@ (a,b) \#rs) t) \implies \text{bal } (\text{Node } (\text{ls}@ (a,c) \#rs) t)$   
 $\langle \text{proof} \rangle$

**lemma** *order-impl-root-order*:  $\llbracket k > 0; \text{order } k t \rrbracket \implies \text{root-order } k t$   
 $\langle \text{proof} \rangle$

**lemma** *sorted-inorder-list-separators*:  $\text{sorted-less } (\text{inorder-list } ts) \implies \text{sorted-less } (\text{separators } ts)$   
 $\langle \text{proof} \rangle$

**corollary** *sorted-inorder-separators*:  $\text{sorted-less } (\text{inorder } (\text{Node } ts) t) \implies \text{sorted-less } (\text{separators } ts)$   
 $\langle \text{proof} \rangle$

**lemma** *sorted-inorder-list-subtrees*:  
 $\text{sorted-less } (\text{inorder-list } ts) \implies \forall \text{sub} \in \text{set } (\text{subtrees } ts). \text{sorted-less } (\text{inorder } \text{sub})$   
 $\langle \text{proof} \rangle$

**corollary** *sorted-inorder-subtrees*:  $\text{sorted-less } (\text{inorder } (\text{Node } ts) t) \implies \forall \text{sub} \in \text{set } (\text{subtrees } ts). \text{sorted-less } (\text{inorder } \text{sub})$   
 $\langle \text{proof} \rangle$

**lemma** *sorted-inorder-list-induct-subtree*:  
 $\text{sorted-less } (\text{inorder-list } (\text{ls}@ (\text{sub}, \text{sep}) \#rs)) \implies \text{sorted-less } (\text{inorder } \text{sub})$   
 $\langle \text{proof} \rangle$

**corollary** *sorted-inorder-induct-subtree*:  
 $\text{sorted-less } (\text{inorder } (\text{Node } (\text{ls}@ (\text{sub}, \text{sep}) \#rs) t)) \implies \text{sorted-less } (\text{inorder } \text{sub})$   
 $\langle \text{proof} \rangle$

**lemma** *sorted-inorder-induct-last*:  $\text{sorted-less } (\text{inorder } (\text{Node } ts) t) \implies \text{sorted-less } (\text{inorder } t)$   
 $\langle \text{proof} \rangle$

**lemma** *sorted-leaves-list-subtrees*:  
 $\text{sorted-less } (\text{leaves-list } ts) \implies \forall \text{sub} \in \text{set } (\text{subtrees } ts). \text{sorted-less } (\text{leaves } \text{sub})$   
 $\langle \text{proof} \rangle$



**corollary** *sorted-leaves-subtrees*:  $\text{sorted-less } (\text{leaves } (\text{Node } ts \ t)) \implies \forall \text{ sub} \in \text{set } (\text{subtrees } ts). \text{sorted-less } (\text{leaves } \text{sub})$

*<proof>*

**lemma** *sorted-leaves-list-induct-subtree*:

$\text{sorted-less } (\text{leaves-list } (ls@(sub,sep)\#rs)) \implies \text{sorted-less } (\text{leaves } \text{sub})$

*<proof>*

**corollary** *sorted-leaves-induct-subtree*:

$\text{sorted-less } (\text{leaves } (\text{Node } (ls@(sub,sep)\#rs) \ t)) \implies \text{sorted-less } (\text{leaves } \text{sub})$

*<proof>*

**lemma** *sorted-leaves-induct-last*:  $\text{sorted-less } (\text{leaves } (\text{Node } ts \ t)) \implies \text{sorted-less } (\text{leaves } t)$

*<proof>*

Additional lemmas on the sortedness of the whole tree, which is correct alignment of navigation structure and leave data

**fun** *inbetween where*

$\text{inbetween } fl \ \text{Nil } t \ u = fl \ t \ u \ |$

$\text{inbetween } fl \ ((sub,sep)\#xs) \ t \ u = (fl \ sub \ sep \wedge \text{inbetween } f \ sep \ xs \ t \ u)$

**thm** *fold-cong*

**lemma** *cong-inbetween[fundef-cong]*:

$\llbracket a = b; xs = ys; \bigwedge l' \ u' \ sub \ sep. (sub,sep) \in \text{set } ys \implies f \ l' \ sub \ u' = g \ l' \ sub \ u'; \bigwedge l' \ u'. f \ l' \ a \ u' = g \ l' \ b \ u' \rrbracket$

$\implies \text{inbetween } fl \ xs \ a \ u = \text{inbetween } g \ l \ ys \ b \ u$

*<proof>*

**fun** *aligned :: 'a :: linorder  $\Rightarrow$  - where*

$\text{aligned } l \ (\text{Leaf } ks) \ u = (l < u \wedge (\forall x \in \text{set } ks. l < x \wedge x \leq u)) \ |$

$\text{aligned } l \ (\text{Node } ts \ t) \ u = (\text{inbetween } \text{aligned } l \ ts \ t \ u)$

**lemma** *sorted-less-merge*:  $\text{sorted-less } (as@[a]) \implies \text{sorted-less } (a\#bs) \implies \text{sorted-less } (as@a\#bs)$

*<proof>*

**thm** *aligned.simps*

**lemma** *leaves-cases*:  $x \in \text{set } (\text{leaves } (\text{Node } ts \ t)) \implies (\exists (sub,sep) \in \text{set } ts. x \in \text{set } (\text{leaves } \text{sub})) \vee x \in \text{set } (\text{leaves } t)$

*<proof>*

**lemma** *align-sub*:  $\text{aligned } l \ (\text{Node } ts \ t) \ u \implies (sub,sep) \in \text{set } ts \implies \exists l' \in \text{set } (\text{separators } ts) \cup \{l\}. \text{aligned } l' \ sub \ sep$

*<proof>*

**lemma** *align-last*:  $\text{aligned } l \text{ (Node (ts@[sub,sep])) } t \text{ } u \implies \text{aligned sep } t \text{ } u$   
 ⟨proof⟩

**lemma** *align-last'*:  $\text{aligned } l \text{ (Node ts } t \text{ ) } u \implies \exists l' \in \text{set (separators ts)} \cup \{l\}.$   
 $\text{aligned } l' \text{ } t \text{ } u$   
 ⟨proof⟩

**lemma** *aligned-sorted-inorder*:  $\text{aligned } l \text{ } t \text{ } u \implies \text{sorted-less (l\#(inorder t)@[u])}$   
 ⟨proof⟩

**lemma** *separators-in-inorder-list*:  $\text{set (separators ts)} \subseteq \text{set (inorder-list ts)}$   
 ⟨proof⟩

**lemma** *separators-in-inorder*:  $\text{set (separators ts)} \subseteq \text{set (inorder (Node ts } t \text{))}$   
 ⟨proof⟩

**lemma** *aligned-sorted-separators*:  $\text{aligned } l \text{ (Node ts } t \text{ ) } u \implies \text{sorted-less (l\#(separators ts)@[u])}$   
 ⟨proof⟩

**lemma** *aligned-leaves-inbetween*:  $\text{aligned } l \text{ } t \text{ } u \implies \forall x \in \text{set (leaves } t \text{)}. l < x \wedge x \leq u$   
 ⟨proof⟩

**lemma** *aligned-leaves-list-inbetween*:  $\text{aligned } l \text{ (Node ts } t \text{ ) } u \implies \forall x \in \text{set (leaves-list ts)}. l < x \wedge x \leq u$   
 ⟨proof⟩

**lemma** *aligned-split-left*:  $\text{aligned } l \text{ (Node (ls@[sub,sep]\#rs) } t \text{ ) } u \implies \text{aligned } l \text{ (Node ls } \text{sub) sep}$   
 ⟨proof⟩

**lemma** *aligned-split-right*:  $\text{aligned } l \text{ (Node (ls@[sub,sep]\#rs) } t \text{ ) } u \implies \text{aligned sep (Node rs } t \text{ ) } u$   
 ⟨proof⟩

**lemma** *aligned-subst*:  $\text{aligned } l \text{ (Node (ls@[sub',subl]\#(sub,subsep)\#rs) } t \text{ ) } u \implies \text{aligned } \text{subl } \text{subsub } \text{subsep} \implies \text{aligned } l \text{ (Node (ls@[sub',subl]\#(subsub,subsep)\#rs) } t \text{ ) } u$   
 ⟨proof⟩

**lemma** *aligned-subst-emptyls*:  $\text{aligned } l \text{ (Node ((sub,subsep)\#rs) } t \text{ ) } u \implies \text{aligned } l \text{ subsub } \text{subsep} \implies \text{aligned } l \text{ (Node ((subsub,subsep)\#rs) } t \text{ ) } u$   
 ⟨proof⟩

**lemma** *aligned-subst-last*:  $\text{aligned } l \text{ (Node (ts'@[sub',sep']) } t \text{ ) } u \implies \text{aligned sep'}$

$t' u \implies$   
*aligned l (Node (ts'@[sub', sep']) t) u*  
 ⟨proof⟩

**fun** *Laligned* :: 'a :: linorder bplustree  $\implies$  - **where**  
*Laligned (Leaf ks) u* = ( $\forall x \in \text{set } ks. x \leq u$ ) |  
*Laligned (Node ts t) u* = (case ts of []  $\implies$  (*Laligned t u*) |  
 (sub,sep)#ts'  $\implies$  ((*Laligned sub sep*)  $\wedge$  *inbetween aligned sep ts' t u*))

**lemma** *Laligned-nonempty-Node*: *Laligned (Node ((sub,sep)#ts') t) u* =  
 ((*Laligned sub sep*)  $\wedge$  *inbetween aligned sep ts' t u*)  
 ⟨proof⟩

**lemma** *aligned-imp-Laligned*: *aligned l t u*  $\implies$  *Laligned t u*  
 ⟨proof⟩

**lemma** *Laligned-split-left*: *Laligned (Node (ls@(sub,sep)#rs) t) u*  $\implies$  *Laligned*  
*(Node ls sub) sep*  
 ⟨proof⟩

**lemma** *Laligned-split-right*: *Laligned (Node (ls@(sub,sep)#rs) t) u*  $\implies$  *aligned sep*  
*(Node rs t) u*  
 ⟨proof⟩

**lemma** *Lalign-sub*: *Laligned (Node ((a,b)#ts) t) u*  $\implies$  (sub,sep)  $\in$  set ts  $\implies$   $\exists l'$   
 $\in$  set (separators ts)  $\cup$  {b}. *aligned l' sub sep*  
 ⟨proof⟩

**lemma** *Lalign-last*: *Laligned (Node (ts@[sub,sep]) t) u*  $\implies$  *aligned sep t u*  
 ⟨proof⟩

**lemma** *Lalign-last'*: *Laligned (Node ((a,b)#ts) t) u*  $\implies$   $\exists l' \in$  set (separators ts)  
 $\cup$  {b}. *aligned l' t u*  
 ⟨proof⟩

**lemma** *Lalign-Llast*: *Laligned (Node ts t) u*  $\implies$  *Laligned t u*  
 ⟨proof⟩

**lemma** *Laligned-sorted-inorder*: *Laligned t u*  $\implies$  *sorted-less ((inorder t)@[u])*  
 ⟨proof⟩

**lemma** *Laligned-sorted-separators*: *Laligned (Node ts t) u*  $\implies$  *sorted-less ((separators*  
*ts)@[u])*  
 ⟨proof⟩

**lemma** *Laligned-leaves-inbetween*: *Laligned t u*  $\implies$   $\forall x \in$  set (leaves t).  $x \leq u$   
 ⟨proof⟩

**lemma** *Laligned-leaves-list-inbetween*:  $Laligned (Node\ ts\ t)\ u \implies \forall x \in set\ (leaves-list\ ts).\ x \leq u$   
 ⟨proof⟩

**lemma** *Laligned-subst-last*:  $Laligned (Node\ (ts'@[sub',\ sep'])\ t)\ u \implies aligned\ sep'\ t'\ u \implies$   
 $Laligned (Node\ (ts'@[sub',\ sep'])\ t')\ u$   
 ⟨proof⟩

**lemma** *Laligned-subst*:  $Laligned (Node\ (ls@[sub',\ subl]#\ (sub,subsep)\ #rs)\ t)\ u \implies$   
 $aligned\ subl\ subsub\ subsep \implies$   
 $Laligned (Node\ (ls@[sub',\ subl]#\ (subsub,subsep)\ #rs)\ t)\ u$   
 ⟨proof⟩

**lemma** *concat-leaf-nodes-leaves*:  $(concat\ (map\ leaves\ (leaf-nodes\ t))) = leaves\ t$   
 ⟨proof⟩

**lemma** *leaf-nodes-not-empty*:  $leaf-nodes\ t \neq []$   
 ⟨proof⟩

**end**  
**theory** *BPlusTree-Split*  
**imports** *BPlusTree*  
**begin**

## 5.6 Auxiliary functions

**fun** *split-half*::  $- list \Rightarrow - list \times - list$  **where**  
 $split-half\ xs = (take\ ((length\ xs + 1)\ div\ 2)\ xs,\ drop\ ((length\ xs + 1)\ div\ 2)\ xs)$

**lemma** *split-half-conc*:  $split-half\ xs = (ls,\ rs) = (xs = ls@rs \wedge length\ ls = (length\ xs + 1)\ div\ 2)$   
 ⟨proof⟩

**lemma** *drop-not-empty*:  $xs \neq [] \implies drop\ (length\ xs\ div\ 2)\ xs \neq []$   
 ⟨proof⟩

**lemma** *take-not-empty*:  $xs \neq [] \implies take\ ((length\ xs + 1)\ div\ 2)\ xs \neq []$   
 ⟨proof⟩

**lemma** *split-half-not-empty*:  $length\ xs \geq 1 \implies \exists ls\ a\ rs.\ split-half\ xs = (ls@[a],rs)$   
 ⟨proof⟩

## 5.7 The split function locale

Here, we abstract away the inner workings of the split function for B-tree operations.

**lemma** *leaves-conc*:  $leaves\ (Node\ (ls@rs)\ t) = leaves-list\ ls\ @\ leaves-list\ rs\ @\ leaves$

$t$   
 $\langle proof \rangle$

**locale** *split-tree* =

**fixes** *split* :: ('a bplustree × 'a::{linorder,order-top}) list ⇒ 'a ⇒ (('a bplustree × 'a) list × ('a bplustree × 'a) list)

**assumes** *split-req*:

$\llbracket split\ xs\ p = (ls,rs) \rrbracket \implies xs = ls @ rs$

$\llbracket split\ xs\ p = (ls@[sub,sep],rs); sorted-less\ (separators\ xs) \rrbracket \implies sep < p$

$\llbracket split\ xs\ p = (ls,(sub,sep)\#rs); sorted-less\ (separators\ xs) \rrbracket \implies p \leq sep$

**begin**

**lemmas** *split-conc* = *split-req*(1)

**lemmas** *split-sorted* = *split-req*(2,3)

**lemma** [*termination-simp*]:  $(ls, (sub, sep) \# rs) = split\ ts\ y \implies$   
 $size\ sub < Suc\ (size-list\ (\lambda x. Suc\ (size\ (fst\ x)))\ ts + size\ l)$   
 $\langle proof \rangle$

**lemma** *leaves-split*:  $split\ ts\ x = (ls,rs) \implies leaves\ (Node\ ts\ t) = leaves-list\ ls @$   
 $leaves-list\ rs @ leaves\ t$   
 $\langle proof \rangle$

**end**

**locale** *split-list* =

**fixes** *split-list* :: ('a::{linorder,order-top}) list ⇒ 'a ⇒ 'a list × 'a list

**assumes** *split-list-req*:

$\llbracket split-list\ ks\ p = (kls,krs) \rrbracket \implies ks = kls @ krs$

$\llbracket split-list\ ks\ p = (kls@[sep],krs); sorted-less\ ks \rrbracket \implies sep < p$

$\llbracket split-list\ ks\ p = (kls,(sep)\#krs); sorted-less\ ks \rrbracket \implies p \leq sep$

**locale** *split-full* = *split-tree*: *split-tree* *split* + *split-list* *split-list*

**for** *split*::

('a bplustree × 'a::{linorder,order-top}) list ⇒ 'a

⇒ ('a bplustree × 'a) list × ('a bplustree × 'a) list

**and** *split-list*::

'a::{linorder,order-top} list ⇒ 'a

⇒ 'a list × 'a list

## 6 Abstract split functions

### 6.1 Linear split

Finally we show that the split axioms are feasible by providing an example split function

Linear split is similar to well known functions, therefore a quick proof can be done.

**fun** *linear-split* **where** *linear-split* *xs* *x* = (*takeWhile* ( $\lambda(-,s). s < x$ ) *xs*, *dropWhile* ( $\lambda(-,s). s < x$ ) *xs*)

**fun** *linear-split-list* **where** *linear-split-list* *xs* *x* = (*takeWhile* ( $\lambda s. s < x$ ) *xs*, *dropWhile* ( $\lambda s. s < x$ ) *xs*)

**end**  
**theory** *BPlusTree-Set*  
**imports**  
   *BPlusTree-Split*  
   *HOL-Data-Structures.Set-Specs*  
**begin**

## 7 Set interpretation

**lemma** *insert-list-length[simp]*:  
**assumes** *sorted-less* *ks*  
   **and** *set* (*insert-list* *k* *ks*) = *set* *ks*  $\cup$  {*k*}  
   **and** *sorted-less* *ks*  $\implies$  *sorted-less* (*insert-list* *k* *ks*)  
**shows** *length* (*insert-list* *k* *ks*) = *length* *ks* + (if *k*  $\in$  *set* *ks* then 0 else 1)  
<proof>

**lemma** *delete-list-length[simp]*:  
**assumes** *sorted-less* *ks*  
   **and** *set* (*delete-list* *k* *ks*) = *set* *ks* - {*k*}  
   **and** *sorted-less* *ks*  $\implies$  *sorted-less* (*delete-list* *k* *ks*)  
**shows** *length* (*delete-list* *k* *ks*) = *length* *ks* - (if *k*  $\in$  *set* *ks* then 1 else 0)  
<proof>

**lemma** *ins-list-length[simp]*:  
**assumes** *sorted-less* *ks*  
**shows** *length* (*ins-list* *k* *ks*) = *length* *ks* + (if *k*  $\in$  *set* *ks* then 0 else 1)  
<proof>

**lemma** *del-list-length[simp]*:  
**assumes** *sorted-less* *ks*  
**shows** *length* (*del-list* *k* *ks*) = *length* *ks* - (if *k*  $\in$  *set* *ks* then 1 else 0)  
<proof>

**locale** *split-set* = *split-tree*: *split-tree split*  
**for** *split*::  
    (*a bplustree* × *a::*{*linorder,order-top*}) *list* ⇒ *a*  
    ⇒ (*a bplustree* × *a*) *list* × (*a bplustree* × *a*) *list* +  
**fixes** *isin-list* :: *a* ⇒ (*a::*{*linorder,order-top*}) *list* ⇒ *bool*  
**and** *insert-list* :: *a* ⇒ (*a::*{*linorder,order-top*}) *list* ⇒ *a list*  
**and** *delete-list* :: *a* ⇒ (*a::*{*linorder,order-top*}) *list* ⇒ *a list*  
**assumes** *insert-list-req*:  
  
    *sorted-less ks* ⇒ *isin-list x ks* = (*x* ∈ *set ks*)  
    *sorted-less ks* ⇒ *insert-list x ks* = *ins-list x ks*  
    *sorted-less ks* ⇒ *delete-list x ks* = *del-list x ks*  
**begin**  
  
**lemmas** *split-req* = *split-tree.split-req*  
**lemmas** *split-conc* = *split-tree.split-req*(1)  
**lemmas** *split-sorted* = *split-tree.split-req*(2,3)  
  
**lemma** *insert-list-length[simp]*:  
    **assumes** *sorted-less ks*  
    **shows** *length (insert-list k ks)* = *length ks* + (*if k* ∈ *set ks* *then 0 else 1*)  
    ⟨*proof*⟩  
  
**lemma** *set-insert-list[simp]*:  
    *sorted-less ks* ⇒ *set (insert-list k ks)* = *set ks* ∪ {*k*}  
    ⟨*proof*⟩  
  
**lemma** *sorted-insert-list[simp]*:  
    *sorted-less ks* ⇒ *sorted-less (insert-list k ks)*  
    ⟨*proof*⟩  
  
**lemma** *delete-list-length[simp]*:  
    **assumes** *sorted-less ks*  
    **shows** *length (delete-list k ks)* = *length ks* − (*if k* ∈ *set ks* *then 1 else 0*)  
    ⟨*proof*⟩  
  
**lemma** *set-delete-list[simp]*:  
    *sorted-less ks* ⇒ *set (delete-list k ks)* = *set ks* − {*k*}  
    ⟨*proof*⟩  
  
**lemma** *sorted-delete-list[simp]*:  
    *sorted-less ks* ⇒ *sorted-less (delete-list k ks)*  
    ⟨*proof*⟩  
  
**definition** *empty-bplustree* = (*Leaf []*)

## 7.1 Membership

```

fun isin:: 'a bplustree  $\Rightarrow$  'a  $\Rightarrow$  bool where
  isin (Leaf ks) x = (isin-list x ks) |
  isin (Node ts t) x = (
    case split ts x of (-(sub,sep)#rs)  $\Rightarrow$  (
      isin sub x
    )
  | (-,[])  $\Rightarrow$  isin t x
  )

```

Isin proof

**thm** *isin-simps*

**lemma** *sorted-ConsD*: *sorted-less* (y # xs)  $\Longrightarrow$   $x \leq y \Longrightarrow x \notin \text{set } xs$   
 $\langle \text{proof} \rangle$

**lemma** *sorted-snocD*: *sorted-less* (xs @ [y])  $\Longrightarrow$   $y \leq x \Longrightarrow x \notin \text{set } xs$   
 $\langle \text{proof} \rangle$

**lemmas** *isin-simps2* = *sorted-lems sorted-ConsD sorted-snocD*

**lemma** *isin-sorted*: *sorted-less* (xs@a#ys)  $\Longrightarrow$   
 $(x \in \text{set } (xs@a\#ys)) = (\text{if } x < a \text{ then } x \in \text{set } xs \text{ else } x \in \text{set } (a\#ys))$   
 $\langle \text{proof} \rangle$

**lemma** *isin-sorted-split*:

```

assumes Laligned (Node ts t) u
and sorted-less (leaves (Node ts t))
and split ts x = (ls, rs)
shows  $x \in \text{set } (\text{leaves } (\text{Node } ts \ t)) = (x \in \text{set } (\text{leaves-list } rs \ @ \ \text{leaves } t))$ 
 $\langle \text{proof} \rangle$ 

```

**lemma** *isin-sorted-split-right*:

```

assumes split ts x = (ls, (sub,sep)#rs)
and sorted-less (leaves (Node ts t))
and Laligned (Node ts t) u
shows  $x \in \text{set } (\text{leaves-list } ((sub,sep)\#rs) \ @ \ \text{leaves } t) = (x \in \text{set } (\text{leaves } sub))$ 
 $\langle \text{proof} \rangle$ 

```

**theorem** *isin-set-inorder*:

```

assumes sorted-less (leaves t)
and aligned l t u
shows  $\text{isin } t \ x = (x \in \text{set } (\text{leaves } t))$ 

```



*<proof>*

**theorem** *isin-set-Linorder*:

**assumes** *sorted-less* (*leaves t*)

**and** *Laligned t u*

**shows** *isin t x = (x ∈ set (leaves t))*

*<proof>*

**corollary** *isin-set-Linorder-top*:

**assumes** *sorted-less* (*leaves t*)

**and** *Laligned t top*

**shows** *isin t x = (x ∈ set (leaves t))*

*<proof>*

## 7.2 Insertion

The insert function requires an auxiliary data structure and auxiliary invariant functions.

**datatype** *'b up<sub>i</sub> = T<sub>i</sub> 'b bplustree | Up<sub>i</sub> 'b bplustree 'b 'b bplustree*

**fun** *order-up<sub>i</sub>* **where**

*order-up<sub>i</sub> k (T<sub>i</sub> sub) = order k sub |*

*order-up<sub>i</sub> k (Up<sub>i</sub> l a r) = (order k l ∧ order k r)*

**fun** *root-order-up<sub>i</sub>* **where**

*root-order-up<sub>i</sub> k (T<sub>i</sub> sub) = root-order k sub |*

*root-order-up<sub>i</sub> k (Up<sub>i</sub> l a r) = (order k l ∧ order k r)*

**fun** *height-up<sub>i</sub>* **where**

*height-up<sub>i</sub> (T<sub>i</sub> t) = height t |*

*height-up<sub>i</sub> (Up<sub>i</sub> l a r) = max (height l) (height r)*

**fun** *bal-up<sub>i</sub>* **where**

*bal-up<sub>i</sub> (T<sub>i</sub> t) = bal t |*

*bal-up<sub>i</sub> (Up<sub>i</sub> l a r) = (height l = height r ∧ bal l ∧ bal r)*

**fun** *inorder-up<sub>i</sub>* **where**

*inorder-up<sub>i</sub> (T<sub>i</sub> t) = inorder t |*

*inorder-up<sub>i</sub> (Up<sub>i</sub> l a r) = inorder l @ [a] @ inorder r*

**fun** *leaves-up<sub>i</sub>* **where**

*leaves-up<sub>i</sub> (T<sub>i</sub> t) = leaves t |*

*leaves-up<sub>i</sub> (Up<sub>i</sub> l a r) = leaves l @ leaves r*

**fun** *aligned-up<sub>i</sub>* **where**

*aligned-up<sub>i</sub> l (T<sub>i</sub> t) u = aligned l t u |*

*aligned-up<sub>i</sub> l (Up<sub>i</sub> lt a rt) u = (aligned l lt a ∧ aligned a rt u)*

**fun** *Laligned-up<sub>i</sub>* **where**

*Laligned-up<sub>i</sub>* (*T<sub>i</sub>* *t*) *u* = *Laligned* *t* *u* |

*Laligned-up<sub>i</sub>* (*Up<sub>i</sub>* *lt* *a* *rt*) *u* = (*Laligned* *lt* *a* ∧ *aligned* *a* *rt* *u*)

The following function merges two nodes and returns separately split nodes if an overflow occurs

**fun** *node<sub>i</sub>*:: *nat* ⇒ ('*a* *bplustree* × '*a*) *list* ⇒ '*a* *bplustree* ⇒ '*a* *up<sub>i</sub>* **where**

*node<sub>i</sub>* *k* *ts* *t* = (

if *length* *ts* ≤ 2\**k* then *T<sub>i</sub>* (*Node* *ts* *t*)

else (

case *split-half* *ts* of (*ls*, *rs*) ⇒

case *last* *ls* of (*sub*,*sep*) ⇒

*Up<sub>i</sub>* (*Node* (*butlast* *ls*) *sub*) *sep* (*Node* *rs* *t*)

)

)

**fun** *Lnode<sub>i</sub>*:: *nat* ⇒ '*a* *list* ⇒ '*a* *up<sub>i</sub>* **where**

*Lnode<sub>i</sub>* *k* *ts* = (

if *length* *ts* ≤ 2\**k* then *T<sub>i</sub>* (*Leaf* *ts*)

else (

case *split-half* *ts* of (*ls*, *rs*) ⇒

*Up<sub>i</sub>* (*Leaf* *ls*) (*last* *ls*) (*Leaf* *rs*)

)

)

**fun** *ins*:: *nat* ⇒ '*a* ⇒ '*a* *bplustree* ⇒ '*a* *up<sub>i</sub>* **where**

*ins* *k* *x* (*Leaf* *ks*) = *Lnode<sub>i</sub>* *k* (*insert-list* *x* *ks*) |

*ins* *k* *x* (*Node* *ts* *t*) = (

case *split* *ts* *x* of

(*ls*,(*sub*,*sep*)#*rs*) ⇒

(case *ins* *k* *x* *sub* of

*Up<sub>i</sub>* *l* *a* *r* ⇒

*node<sub>i</sub>* *k* (*ls*@(*l*,*a*)#(*r*,*sep*)#*rs*) *t* |

*T<sub>i</sub>* *a* ⇒

*T<sub>i</sub>* (*Node* (*ls*@(*a*,*sep*)#*rs*) *t*)) |

(*ls*, []) ⇒

(case *ins* *k* *x* *t* of

*Up<sub>i</sub>* *l* *a* *r* ⇒

*node<sub>i</sub>* *k* (*ls*@[(*l*,*a*)] *r*) |

*T<sub>i</sub>* *a* ⇒

*T<sub>i</sub>* (*Node* *ls* *a*)

)

)

**fun** *tree<sub>i</sub>*:: '*a* *up<sub>i</sub>* ⇒ '*a* *bplustree* **where**

*tree<sub>i</sub>* (*T<sub>i</sub>* *sub*) = *sub* |

$tree_i (Up_i l a r) = (Node [(l,a)] r)$

**fun**  $insert::nat \Rightarrow 'a \Rightarrow 'a\ bplustree \Rightarrow 'a\ bplustree$  **where**  
 $insert\ k\ x\ t = tree_i (ins\ k\ x\ t)$

### 7.3 Proofs of functional correctness

**lemma**  $node_i\text{-ti-simp}$ :  $node_i\ k\ ts\ t = T_i\ x \Longrightarrow x = Node\ ts\ t$   
 $\langle proof \rangle$

**lemma**  $Lnode_i\text{-ti-simp}$ :  $Lnode_i\ k\ ts = T_i\ x \Longrightarrow x = Leaf\ ts$   
 $\langle proof \rangle$

**lemma**  $split\text{-set}$ :

**assumes**  $split\ ts\ z = (ls,(a,b)\#rs)$

**shows**  $(a,b) \in set\ ts$

**and**  $(x,y) \in set\ ls \Longrightarrow (x,y) \in set\ ts$

**and**  $(x,y) \in set\ rs \Longrightarrow (x,y) \in set\ ts$

**and**  $set\ ls \cup set\ rs \cup \{(a,b)\} = set\ ts$

**and**  $\exists x \in set\ ts. b \in Basic\text{-BNFs.snds}\ x$

$\langle proof \rangle$

**lemma**  $split\text{-length}$ :

$split\ ts\ x = (ls, rs) \Longrightarrow length\ ls + length\ rs = length\ ts$

$\langle proof \rangle$

**lemma**  $node_i\text{-cases}$ :  $length\ xs \leq k \vee (\exists\ ls\ sub\ sep\ rs. split\text{-half}\ xs = (ls@[sub,sep],rs))$   
 $\langle proof \rangle$

**lemma**  $Lnode_i\text{-cases}$ :  $length\ xs \leq k \vee (\exists\ ls\ sep\ rs. split\text{-half}\ xs = (ls@[sep],rs))$   
 $\langle proof \rangle$

**lemma**  $root\text{-order-tree}_i$ :  $root\text{-order-up}_i (Suc\ k)\ t = root\text{-order}\ (Suc\ k)\ (tree_i\ t)$   
 $\langle proof \rangle$

**lemma**  $length\text{-take-left}$ :  $length\ (take\ ((length\ ts + 1)\ div\ 2)\ ts) = (length\ ts + 1)\ div\ 2$   
 $\langle proof \rangle$

**lemma**  $node_i\text{-root-order}$ :

**assumes**  $length\ ts > 0$

**and**  $length\ ts \leq 4*k+1$

**and**  $\forall x \in set\ (subtrees\ ts). order\ k\ x$

**and**  $order\ k\ t$

**shows**  $root\text{-order-up}_i\ k\ (node_i\ k\ ts\ t)$

*<proof>*

**lemma** *node<sub>i</sub>-order-helper*:  
 **assumes** *length ts ≥ k*  
 **and** *length ts ≤ 4\*k+1*  
 **and**  $\forall x \in \text{set } (\text{subtrees } ts). \text{ order } k \ x$   
 **and** *order k t*  
 **shows** *case (node<sub>i</sub> k ts t) of T<sub>i</sub> t ⇒ order k t | - ⇒ True*  
*<proof>*

**lemma** *node<sub>i</sub>-order*:  
 **assumes** *length ts ≥ k*  
 **and** *length ts ≤ 4\*k+1*  
 **and**  $\forall x \in \text{set } (\text{subtrees } ts). \text{ order } k \ x$   
 **and** *order k t*  
 **shows** *order-up<sub>i</sub> k (node<sub>i</sub> k ts t)*  
*<proof>*

**lemma** *Lnode<sub>i</sub>-root-order*:  
 **assumes** *length ts > 0*  
 **and** *length ts ≤ 4\*k*  
 **shows** *root-order-up<sub>i</sub> k (Lnode<sub>i</sub> k ts)*  
*<proof>*

**lemma** *Lnode<sub>i</sub>-order-helper*:  
 **assumes** *length ts ≥ k*  
 **and** *length ts ≤ 4\*k+1*  
 **shows** *case (Lnode<sub>i</sub> k ts) of T<sub>i</sub> t ⇒ order k t | - ⇒ True*  
*<proof>*

**lemma** *Lnode<sub>i</sub>-order*:  
 **assumes** *length ts ≥ k*  
 **and** *length ts ≤ 4\*k*  
 **shows** *order-up<sub>i</sub> k (Lnode<sub>i</sub> k ts)*  
*<proof>*

**lemma** *ins-order*:  
 *k > 0 ⇒ sorted-less (leaves t) ⇒ order k t ⇒ order-up<sub>i</sub> k (ins k x t)*  
*<proof>*

**lemma** *ins-root-order*:  
 **assumes** *k > 0 sorted-less (leaves t) root-order k t*  
 **shows** *root-order-up<sub>i</sub> k (ins k x t)*

*<proof>*

**lemma** *height-list-split*:  $\text{height-up}_i (\text{Up}_i (\text{Node } ls \ a) \ b (\text{Node } rs \ t)) = \text{height} (\text{Node } (ls@ (a,b)\#rs) \ t)$   
*<proof>*

**lemma** *node<sub>i</sub>-height*:  $\text{height-up}_i (\text{node}_i \ k \ ts \ t) = \text{height} (\text{Node } ts \ t)$   
*<proof>*

**lemma** *Lnode<sub>i</sub>-height*:  $\text{height-up}_i (\text{Lnode}_i \ k \ xs) = \text{height} (\text{Leaf } xs)$   
*<proof>*

**lemma** *bal-up<sub>i</sub>-tree*:  $\text{bal-up}_i \ t = \text{bal} (\text{tree}_i \ t)$   
*<proof>*

**lemma** *bal-list-split*:  $\text{bal} (\text{Node } (ls@ (a,b)\#rs) \ t) \implies \text{bal-up}_i (\text{Up}_i (\text{Node } ls \ a) \ b (\text{Node } rs \ t))$   
*<proof>*

**lemma** *node<sub>i</sub>-bal*:  
  **assumes**  $\text{bal} (\text{Node } ts \ t)$   
  **shows**  $\text{bal-up}_i (\text{node}_i \ k \ ts \ t)$   
*<proof>*

**lemma** *node<sub>i</sub>-aligned*:  
  **assumes**  $\text{aligned } l (\text{Node } ts \ t) \ u$   
  **shows**  $\text{aligned-up}_i \ l (\text{node}_i \ k \ ts \ t) \ u$   
*<proof>*

**lemma** *node<sub>i</sub>-Laligned*:  
  **assumes**  $\text{Laligned} (\text{Node } ts \ t) \ u$   
  **shows**  $\text{Laligned-up}_i (\text{node}_i \ k \ ts \ t) \ u$   
*<proof>*

**lemma** *length-right-side*:  $\text{length } xs > 1 \implies \text{length} (\text{drop } ((\text{length } xs + 1) \ \text{div } 2) \ xs) > 0$   
*<proof>*

**lemma** *Lnode<sub>i</sub>-aligned*:  
  **assumes**  $\text{aligned } l (\text{Leaf } ks) \ u$   
  **and**  $\text{sorted-less } ks$   
  **and**  $k > 0$   
  **shows**  $\text{aligned-up}_i \ l (\text{Lnode}_i \ k \ ks) \ u$   
*<proof>*

**lemma** *height-up<sub>i</sub>-merge*:  $\text{height-up}_i (Up_i l a r) = \text{height } t \implies \text{height } (\text{Node } (ls @ (t, x) \# rs) tt) = \text{height } (\text{Node } (ls @ (l, a) \# (r, x) \# rs) tt)$

*<proof>*

**lemma** *ins-height*:  $\text{height-up}_i (\text{ins } k x t) = \text{height } t$

*<proof>*

**lemma** *ins-bal*:  $\text{bal } t \implies \text{bal-up}_i (\text{ins } k x t)$

*<proof>*

**lemma** *node<sub>i</sub>-leaves*:  $\text{leaves-up}_i (\text{node}_i k ts t) = \text{leaves } (\text{Node } ts t)$

*<proof>*

**corollary** *node<sub>i</sub>-leaves-simps*:

$\text{node}_i k ts t = T_i t' \implies \text{leaves } t' = \text{leaves } (\text{Node } ts t)$

$\text{node}_i k ts t = Up_i l a r \implies \text{leaves } l @ \text{leaves } r = \text{leaves } (\text{Node } ts t)$

*<proof>*

**lemma** *Lnode<sub>i</sub>-leaves*:  $\text{leaves-up}_i (\text{Lnode}_i k xs) = \text{leaves } (\text{Leaf } xs)$

*<proof>*

**corollary** *Lnode<sub>i</sub>-leaves-simps*:

$\text{Lnode}_i k xs = T_i t \implies \text{leaves } t = \text{leaves } (\text{Leaf } xs)$

$\text{Lnode}_i k xs = Up_i l a r \implies \text{leaves } l @ \text{leaves } r = \text{leaves } (\text{Leaf } xs)$

*<proof>*

**lemma** *ins-list-split*:

**assumes** *Laligned*  $(\text{Node } ts t) u$

**and** *sorted-less*  $(\text{leaves } (\text{Node } ts t))$

**and** *split*  $ts x = (ls, rs)$

**shows** *ins-list*  $x (\text{leaves } (\text{Node } ts t)) = \text{leaves-list } ls @ \text{ins-list } x (\text{leaves-list } rs @ \text{leaves } t)$

*<proof>*

**lemma** *ins-list-split-right*:

**assumes** *split*  $ts x = (ls, (sub, sep) \# rs)$

**and** *sorted-less*  $(\text{leaves } (\text{Node } ts t))$

**and** *Laligned*  $(\text{Node } ts t) u$

**shows** *ins-list*  $x (\text{leaves-list } ((sub, sep) \# rs) @ \text{leaves } t) = \text{ins-list } x (\text{leaves } sub) @ \text{leaves-list } rs @ \text{leaves } t$

$\langle \text{proof} \rangle$

**lemma** *ins-list-idem-eq-isin*:  $\text{sorted-less } xs \implies x \in \text{set } xs \iff (\text{ins-list } x \text{ } xs = xs)$   
 $\langle \text{proof} \rangle$

**lemma** *ins-list-contains-idem*:  $\llbracket \text{sorted-less } xs; x \in \text{set } xs \rrbracket \implies (\text{ins-list } x \text{ } xs = xs)$   
 $\langle \text{proof} \rangle$

**lemma** *aligned-insert-list*:  $\text{sorted-less } ks \implies l < x \implies x \leq u \implies \text{aligned } l \text{ (Leaf } ks) u \implies \text{aligned } l \text{ (Leaf (insert-list } x \text{ } ks)) } u$   
 $\langle \text{proof} \rangle$

**lemma** *align-subst-two*:  $\text{aligned } l \text{ (Node (ts@[sub,sepl])) } t) u \implies \text{aligned sep } lt \text{ } a \implies \text{aligned } a \text{ } rt \text{ } u \implies \text{aligned } l \text{ (Node (ts@[sub,sepl],(lt,a])) } rt) u$   
 $\langle \text{proof} \rangle$

**lemma** *align-subst-three*:  $\text{aligned } l \text{ (Node (ls@[subl,sepl]\#(subr,sepr)\#rs) } t) u \implies \text{aligned sepl } lt \text{ } a \implies \text{aligned } a \text{ } rt \text{ } sepr \implies \text{aligned } l \text{ (Node (ls@[subl,sepl]\#(lt,a)\#(rt,sepr)\#rs) } t) u$   
 $\langle \text{proof} \rangle$

**declare** *node<sub>i</sub>.simps* [simp del]  
**declare** *node<sub>i</sub>-leaves* [simp add]

**lemma** *ins-inorder*:  
  **assumes**  $k > 0$   
  **and**  $\text{aligned } l \text{ } t \text{ } u$   
  **and**  $\text{sorted-less (leaves } t)$   
  **and**  $\text{root-order } k \text{ } t$   
  **and**  $l < x \text{ } x \leq u$   
  **shows**  $(\text{leaves-up}_i (\text{ins } k \text{ } x \text{ } t)) = \text{ins-list } x \text{ (leaves } t) \wedge \text{aligned-up}_i l \text{ (ins } k \text{ } x \text{ } t) u$   
 $\langle \text{proof} \rangle$

**declare** *node<sub>i</sub>.simps* [simp add]  
**declare** *node<sub>i</sub>-leaves* [simp del]

**lemma** *Laligned-insert-list*:  $\text{sorted-less } ks \implies x \leq u \implies \text{Laligned (Leaf } ks) u \implies \text{Laligned (Leaf (insert-list } x \text{ } ks)) } u$   
 $\langle \text{proof} \rangle$

**lemma** *Lalign-subst-two*:  $\text{Laligned (Node (ts@[sub,sepl])) } t) u \implies \text{aligned sep } lt \text{ } a \implies \text{aligned } a \text{ } rt \text{ } u \implies \text{Laligned (Node (ts@[sub,sepl],(lt,a])) } rt) u$   
 $\langle \text{proof} \rangle$

**lemma** *Lalign-subst-three*:  $\text{Laligned (Node (ls@[subl,sepl]\#(subr,sepr)\#rs) } t) u \implies \text{aligned sepl } lt \text{ } a \implies \text{aligned } a \text{ } rt \text{ } sepr \implies \text{Laligned (Node (ls@[subl,sepl]\#(lt,a)\#(rt,sepr)\#rs) } t) u$

*t*) *u*  
⟨*proof*⟩

**lemma** *Lnode<sub>i</sub>-Laligned*:  
  **assumes** *Laligned* (*Leaf ks*) *u*  
  **and** *sorted-less ks*  
  **and**  $k > 0$   
  **shows** *Laligned-up<sub>i</sub>* (*Lnode<sub>i</sub> k ks*) *u*  
  ⟨*proof*⟩

**declare** *node<sub>i</sub>.simps* [*simp del*]  
**declare** *node<sub>i</sub>-leaves* [*simp add*]

**lemma** *ins-Linorder*:  
  **assumes**  $k > 0$   
  **and** *Laligned t u*  
  **and** *sorted-less* (*leaves t*)  
  **and** *root-order k t*  
  **and**  $x \leq u$   
  **shows** (*leaves-up<sub>i</sub>* (*ins k x t*)) = *ins-list x* (*leaves t*)  $\wedge$  *Laligned-up<sub>i</sub>* (*ins k x t*) *u*  
  ⟨*proof*⟩

**declare** *node<sub>i</sub>.simps* [*simp add*]  
**declare** *node<sub>i</sub>-leaves* [*simp del*]

**thm** *ins.induct*  
**thm** *bplustree.induct*

**lemma** *tree<sub>i</sub>-bal*: *bal-up<sub>i</sub> u*  $\implies$  *bal* (*tree<sub>i</sub> u*)  
  ⟨*proof*⟩

**lemma** *tree<sub>i</sub>-order*:  $\llbracket k > 0; \text{root-order-up}_i k u \rrbracket \implies$  *root-order k* (*tree<sub>i</sub> u*)  
  ⟨*proof*⟩

**lemma** *tree<sub>i</sub>-inorder*: *inorder-up<sub>i</sub> u* = *inorder* (*tree<sub>i</sub> u*)  
  ⟨*proof*⟩

**lemma** *tree<sub>i</sub>-leaves*: *leaves-up<sub>i</sub> u* = *leaves* (*tree<sub>i</sub> u*)  
  ⟨*proof*⟩

**lemma** *tree<sub>i</sub>-aligned*: *aligned-up<sub>i</sub> l a u*  $\implies$  *aligned l* (*tree<sub>i</sub> a*) *u*  
  ⟨*proof*⟩

**lemma** *tree<sub>i</sub>-Laligned*: *Laligned-up<sub>i</sub> a u*  $\implies$  *Laligned* (*tree<sub>i</sub> a*) *u*  
  ⟨*proof*⟩



**lemma** *insert-bal*:  $bal\ t \implies bal\ (insert\ k\ x\ t)$   
 ⟨proof⟩

**lemma** *insert-order*:  $\llbracket k > 0; sorted-less\ (leaves\ t); root-order\ k\ t \rrbracket \implies root-order\ k\ (insert\ k\ x\ t)$   
 ⟨proof⟩

**lemma** *insert-inorder*:  
**assumes**  $k > 0\ root-order\ k\ t\ sorted-less\ (leaves\ t)\ aligned\ l\ t\ u\ l < x\ x \leq u$   
**shows**  $leaves\ (insert\ k\ x\ t) = ins-list\ x\ (leaves\ t)$   
**and**  $aligned\ l\ (insert\ k\ x\ t)\ u$   
 ⟨proof⟩

**lemma** *insert-Linorder*:  
**assumes**  $k > 0\ root-order\ k\ t\ sorted-less\ (leaves\ t)\ Laligned\ t\ u\ x \leq u$   
**shows**  $leaves\ (insert\ k\ x\ t) = ins-list\ x\ (leaves\ t)$   
**and**  $Laligned\ (insert\ k\ x\ t)\ u$   
 ⟨proof⟩

**corollary** *insert-Linorder-top*:  
**assumes**  $k > 0\ root-order\ k\ t\ sorted-less\ (leaves\ t)\ Laligned\ t\ top$   
**shows**  $leaves\ (insert\ k\ x\ t) = ins-list\ x\ (leaves\ t)$   
**and**  $Laligned\ (insert\ k\ x\ t)\ top$   
 ⟨proof⟩

## 7.4 Deletion

The following deletion method is inspired by Bauer (70) and Fielding (?). Rather than stealing only a single node from the neighbour, the neighbour is fully merged with the potentially underflowing node. If the resulting node is still larger than allowed, the merged node is split again, using the rules known from insertion splits. If the resulting node has admissable size, it is simply kept in the tree.

**fun** *rebalance-middle-tree* **where**  
*rebalance-middle-tree*  $k\ ls\ (Leaf\ ms)\ sep\ rs\ (Leaf\ ts) =$  (  
 if  $length\ ms \geq k \wedge length\ ts \geq k$  then  
    $Node\ (ls@((Leaf\ ms,sep)\#rs))\ (Leaf\ ts)$   
 else (  
   case  $rs$  of []  $\Rightarrow$  (  
     case  $Lnode_i\ k\ (ms@ts)$  of  
        $T_i\ u \Rightarrow$   
        $Node\ ls\ u \mid$   
        $Up_i\ l\ a\ r \Rightarrow$   
        $Node\ (ls@[l,a])\ r \mid$   
      $(Leaf\ rrs,rsep)\#rs \Rightarrow$  (  
       case  $Lnode_i\ k\ (ms@rrs)$  of  
        $T_i\ u \Rightarrow$

```

    Node (ls@(u,rsep)#rs) (Leaf ts) |
    Up_i l a r =>
    Node (ls@(l,a)#(r,rsep)#rs) (Leaf ts))
  )) |
  rebalance-middle-tree k ls (Node mts mt) sep rs (Node tts tt) = (
  if length mts >= k ^ length tts >= k then
    Node (ls@(Node mts mt,sep)#rs) (Node tts tt)
  else (
    case rs of [] => (
      case node_i k (mts@(mt,sep)#tts) tt of
        T_i u =>
          Node ls u |
          Up_i l a r =>
            Node (ls@[l,a]) r) |
      (Node rts rt,rsep)#rs => (
        case node_i k (mts@(mt,sep)#rts) rt of
          T_i u =>
            Node (ls@(u,rsep)#rs) (Node tts tt) |
            Up_i l a r =>
              Node (ls@(l,a)#(r,rsep)#rs) (Node tts tt))
    ))
  ))

```

All trees are merged with the right neighbour on underflow. Obviously for the last tree this would not work since it has no right neighbour. Therefore this tree, as the only exception, is merged with the left neighbour. However since we it does not make a difference, we treat the situation as if the second to last tree underflowed.

```

fun rebalance-last-tree where
  rebalance-last-tree k ts t = (
  case last ts of (sub,sep) =>
    rebalance-middle-tree k (butlast ts) sub sep [] t
  )

```

Rather than deleting the minimal key from the right subtree, we remove the maximal key of the left subtree. This is due to the fact that the last tree can easily be accessed and the left neighbour is way easier to access than the right neighbour, it resides in the same pair as the separating element to be removed.

```

fun del where
  del k x (Leaf xs) = (Leaf (delete-list x xs)) |
  del k x (Node ts t) = (
  case split ts x of
    (ls,[]) =>
      rebalance-last-tree k ls (del k x t)
  | (ls,(sub,sep)#rs) => (
    rebalance-middle-tree k ls (del k x sub) sep rs t
  )
  )

```

```

fun reduce-root where
  reduce-root (Leaf xs) = (Leaf xs) |
  reduce-root (Node ts t) = (case ts of
    []  $\Rightarrow$  t |
    -  $\Rightarrow$  (Node ts t)
  )

```

```

fun delete where delete k x t = reduce-root (del k x t)

```

An invariant for intermediate states at deletion. In particular we allow for an underflow to 0 subtrees.

```

fun almost-order where
  almost-order k (Leaf xs) = (length xs  $\leq$  2*k) |
  almost-order k (Node ts t) = (
    (length ts  $\leq$  2*k)  $\wedge$ 
    ( $\forall$  s  $\in$  set (subtrees ts). order k s)  $\wedge$ 
    order k t
  )

```

Deletion proofs

```

thm list.simps

```

**lemma** rebalance-middle-tree-height:

```

assumes height t = height sub
and case rs of (rsub,rsep) # list  $\Rightarrow$  height rsub = height t | []  $\Rightarrow$  True
shows height (rebalance-middle-tree k ls sub sep rs t) = height (Node (ls@(sub,sep)#rs)
t)
<proof>

```

**lemma** rebalance-last-tree-height:

```

assumes height t = height sub
and ts = list@[ (sub,sep) ]
shows height (rebalance-last-tree k ts t) = height (Node ts t)
<proof>

```

```

lemma bal-sub-height: bal (Node (ls@a#rs) t)  $\Longrightarrow$  (case rs of []  $\Rightarrow$  True | (sub,sep)#-
 $\Rightarrow$  height sub = height t)
<proof>

```

```

lemma del-height: [ $k > 0$ ; root-order k t; bal t]  $\Longrightarrow$  height (del k x t) = height t
<proof>

```

**lemma** *rebalance-middle-tree-inorder*:

**assumes**  $height\ t = height\ sub$

**and** *case*  $rs$  of  $(rsub, rsep) \# list \Rightarrow height\ rsub = height\ t \mid [] \Rightarrow True$

**shows**  $leaves\ (rebalance-middle-tree\ k\ ls\ sub\ sep\ rs\ t) = leaves\ (Node\ (ls@(\sub,sep)\#rs)\ t)$

*<proof>*

**lemma** *rebalance-last-tree-inorder*:

**assumes**  $height\ t = height\ sub$

**and**  $ts = list@[(sub,sep)]$

**shows**  $leaves\ (rebalance-last-tree\ k\ ts\ t) = leaves\ (Node\ ts\ t)$

*<proof>*

**lemma** *butlast-inorder-app-id*:  $xs = xs' @ [(sub,sep)] \Longrightarrow inorder-list\ xs' @ inorder\ sub\ @\ [sep] = inorder-list\ xs$

*<proof>*

**lemma** *height-bal-subtrees-merge*:  $\llbracket height\ (Node\ as\ a) = height\ (Node\ bs\ b); bal\ (Node\ as\ a); bal\ (Node\ bs\ b) \rrbracket$

$\Longrightarrow \forall x \in set\ (subtrees\ as) \cup \{a\}. height\ x = height\ b$

*<proof>*

**lemma** *bal-list-merge*:

**assumes**  $bal-up_i\ (Up_i\ (Node\ as\ a)\ x\ (Node\ bs\ b))$

**shows**  $bal\ (Node\ (as@(a,x)\#bs)\ b)$

*<proof>*

**lemma** *node<sub>i</sub>-bal-up<sub>i</sub>*:

**assumes**  $bal-up_i\ (node_i\ k\ ts\ t)$

**shows**  $bal\ (Node\ ts\ t)$

*<proof>*

**lemma** *node<sub>i</sub>-bal-simp*:  $bal-up_i\ (node_i\ k\ ts\ t) = bal\ (Node\ ts\ t)$

*<proof>*

**lemma** *rebalance-middle-tree-bal*:

**assumes**  $bal\ (Node\ (ls@(\sub,sep)\#rs)\ t)$

**shows**  $bal\ (rebalance-middle-tree\ k\ ls\ sub\ sep\ rs\ t)$

*<proof>*

**lemma** *rebalance-last-tree-bal*:  $\llbracket bal\ (Node\ ts\ t); ts \neq [] \rrbracket \Longrightarrow bal\ (rebalance-last-tree\ k\ ts\ t)$

*<proof>*

**lemma** *Leaf-merge-aligned*:  $aligned\ l\ (Leaf\ ms)\ m \Longrightarrow aligned\ m\ (Leaf\ rs)\ r \Longrightarrow$

*aligned l (Leaf (ms@rs)) r*  
 ⟨proof⟩

**lemma** *Node-merge-aligned:*

*inbetween aligned l mts mt sep*  $\implies$   
*inbetween aligned sep tts tt u*  $\implies$   
*inbetween aligned l (mts @ (mt, sep) # tts) tt u*  
 ⟨proof⟩

**lemma** *aligned-subst-last-merge:* *aligned l (Node (ts'@[sub', sep'],(sub,sep))) t) u*  
 $\implies$  *aligned sep' t' u*  $\implies$   
*aligned l (Node (ts'@[sub', sep'])) t') u*  
 ⟨proof⟩

**lemma** *aligned-subst-last-merge-two:* *aligned l (Node (ts@[sub',sep'],(sub,sep))) t)*  
*u*  $\implies$  *aligned sep' lt a*  $\implies$  *aligned a rt u*  $\implies$  *aligned l (Node (ts@[sub',sep'],(lt,a)))*  
*rt) u*  
 ⟨proof⟩

**lemma** *aligned-subst-merge:* *aligned l (Node (ls@(lsub, lsep)#(sub,sep)#(rsub,rsep)#rs)*  
*t) u*  $\implies$  *aligned lsep sub' rsep*  $\implies$   
*aligned l (Node (ls@(lsub, lsep)#(sub', rsep)#rs) t) u*  
 ⟨proof⟩

**lemma** *aligned-subst-merge-two:* *aligned l (Node (ls@(lsub, lsep)#(sub,sep)#(rsub,rsep)#rs)*  
*t) u*  $\implies$  *aligned lsep sub' a*  $\implies$   
*aligned a rsub' rsep*  $\implies$  *aligned l (Node (ls@(lsub, lsep)#(sub',a)#(rsub', rsep)#rs)*  
*t) u*  
 ⟨proof⟩

**lemma** *rebalance-middle-tree-aligned:*

**assumes** *aligned l (Node (ls@(sub,sep)#rs) t) u*  
**and** *height t = height sub*  
**and** *sorted-less (leaves (Node (ls@(sub,sep)#rs) t))*  
**and** *k > 0*  
**and** *case rs of (rsub,rsep) # list  $\Rightarrow$  height rsub = height t | []  $\Rightarrow$  True*  
**shows** *aligned l (rebalance-middle-tree k ls sub sep rs t) u*  
 ⟨proof⟩

**lemma** *Node-merge-Laligned:*

*Laligned (Node mts mt) sep*  $\implies$   
*inbetween aligned sep tts tt u*  $\implies$   
*Laligned (Node (mts @ (mt, sep) # tts) tt) u*  
 ⟨proof⟩

**lemma** *Laligned-subst-last-merge:* *Laligned (Node (ts'@[sub', sep'],(sub,sep))) t)*  
*u*  $\implies$  *aligned sep' t' u*  $\implies$   
*Laligned (Node (ts'@[sub', sep'])) t') u*  
 ⟨proof⟩

**lemma** *Laligned-subst-last-merge-two*:  $Laligned (Node (ts@[sub',sep']), (sub,sep))$   
 $t) u \implies aligned\ sep' lt a \implies aligned\ a\ rt\ u \implies Laligned (Node (ts@[sub',sep'], (lt,a)))$   
 $rt) u$   
 $\langle proof \rangle$

**lemma** *Laligned-subst-merge*:  $Laligned (Node (ls@(lsub, lsep) \# (sub,sep) \# (rsub,rsep) \# rs))$   
 $t) u \implies aligned\ lsep\ sub'\ rsep \implies$   
 $Laligned (Node (ls@(lsub, lsep) \# (sub', rsep) \# rs) t) u$   
 $\langle proof \rangle$

**lemma** *Laligned-subst-merge-two*:  $Laligned (Node (ls@(lsub, lsep) \# (sub,sep) \# (rsub,rsep) \# rs))$   
 $t) u \implies aligned\ lsep\ sub'\ a \implies$   
 $aligned\ a\ rsub'\ rsep \implies Laligned (Node (ls@(lsub, lsep) \# (sub',a) \# (rsub', rsep) \# rs))$   
 $t) u$   
 $\langle proof \rangle$

**lemma** *xs-front*:  $xs @ [(a,b)] = (x,y) \# xs' \implies xs @ [(a,b),(c,d)] = (z,zz) \# xs'' \implies$   
 $(x,y) = (z,zz)$   
 $\langle proof \rangle$

**lemma** *rebalance-middle-tree-Laligned*:  
**assumes**  $Laligned (Node (ls@(sub,sep) \# rs) t) u$   
**and**  $height\ t = height\ sub$   
**and**  $sorted-less\ (leaves\ (Node\ (ls@(sub,sep) \# rs)\ t))$   
**and**  $k > 0$   
**and**  $case\ rs\ of\ (rsub,rsep) \# list \implies height\ rsub = height\ t \mid [] \implies True$   
**shows**  $Laligned\ (rebalance-middle-tree\ k\ ls\ sub\ sep\ rs\ t) u$   
 $\langle proof \rangle$

**lemma** *rebalance-last-tree-aligned*:  
**assumes**  $aligned\ l\ (Node\ (ls@[sub,sep]))\ t) u$   
**and**  $height\ t = height\ sub$   
**and**  $sorted-less\ (leaves\ (Node\ (ls@[sub,sep]))\ t)$   
**and**  $k > 0$   
**shows**  $aligned\ l\ (rebalance-last-tree\ k\ (ls@[sub,sep]))\ t) u$   
 $\langle proof \rangle$

**lemma** *rebalance-last-tree-Laligned*:  
**assumes**  $Laligned (Node (ls@[sub,sep])) t) u$   
**and**  $height\ t = height\ sub$   
**and**  $sorted-less\ (leaves\ (Node\ (ls@[sub,sep]))\ t)$   
**and**  $k > 0$   
**shows**  $Laligned\ (rebalance-last-tree\ k\ (ls@[sub,sep]))\ t) u$   
 $\langle proof \rangle$

**lemma** *del-bal*:  
**assumes**  $k > 0$   
**and**  $root-order\ k\ t$

**and** *bal t*  
**shows** *bal (del k x t)*  
 ⟨*proof*⟩

**lemma** *rebalance-middle-tree-order:*

**assumes** *almost-order k sub*  
**and**  $\forall s \in \text{set } (\text{subtrees } (ls@rs)). \text{ order } k \ s \ \text{order } k \ t$   
**and** *case rs of (rsub,rsep) # list  $\Rightarrow$  height rsub = height t | []  $\Rightarrow$  True*  
**and** *length (ls@(sub,sep)#rs)  $\leq$  2\*k*  
**and** *height sub = height t*  
**shows** *almost-order k (rebalance-middle-tree k ls sub sep rs t)*  
 ⟨*proof*⟩

**lemma** *rebalance-middle-tree-last-order:*

**assumes** *almost-order k t*  
**and**  $\forall s \in \text{set } (\text{subtrees } (ls@(sub,sep)#rs)). \text{ order } k \ s$   
**and** *rs = []*  
**and** *length (ls@(sub,sep)#rs)  $\leq$  2\*k*  
**and** *height sub = height t*  
**shows** *almost-order k (rebalance-middle-tree k ls sub sep rs t)*  
 ⟨*proof*⟩

**lemma** *rebalance-last-tree-order:*

**assumes** *ts = ls@[sub,sep]*  
**and**  $\forall s \in \text{set } (\text{subtrees } (ts)). \text{ order } k \ s \ \text{almost-order } k \ t$   
**and** *length ts  $\leq$  2\*k*  
**and** *height sub = height t*  
**shows** *almost-order k (rebalance-last-tree k ts t)*  
 ⟨*proof*⟩

**lemma** *del-order:*

**assumes** *k > 0*  
**and** *root-order k t*  
**and** *bal t*  
**and** *sorted (leaves t)*  
**shows** *almost-order k (del k x t)*  
 ⟨*proof*⟩

**thm** *del-list-sorted*

**lemma** *del-list-split:*

**assumes** *Laligned (Node ts t) u*  
**and** *sorted-less (leaves (Node ts t))*

**and** *split*  $ts\ x = (ls, rs)$   
**shows**  $del\text{-list}\ x\ (leaves\ (Node\ ts\ t)) = leaves\text{-list}\ ls\ @\ del\text{-list}\ x\ (leaves\text{-list}\ rs\ @\ leaves\ t)$   
*<proof>*

**corollary** *del-list-split-aligned:*

**assumes** *aligned*  $l\ (Node\ ts\ t)\ u$   
**and** *sorted-less*  $(leaves\ (Node\ ts\ t))$   
**and** *split*  $ts\ x = (ls, rs)$   
**shows**  $del\text{-list}\ x\ (leaves\ (Node\ ts\ t)) = leaves\text{-list}\ ls\ @\ del\text{-list}\ x\ (leaves\text{-list}\ rs\ @\ leaves\ t)$   
*<proof>*

**lemma** *del-list-split-right:*

**assumes** *Laligned*  $(Node\ ts\ t)\ u$   
**and** *sorted-less*  $(leaves\ (Node\ ts\ t))$   
**and** *split*  $ts\ x = (ls, (sub, sep)\#rs)$   
**shows**  $del\text{-list}\ x\ (leaves\text{-list}\ ((sub, sep)\#rs)\ @\ leaves\ t) = del\text{-list}\ x\ (leaves\ sub)\ @\ leaves\text{-list}\ rs\ @\ leaves\ t$   
*<proof>*

**corollary** *del-list-split-right-aligned:*

**assumes** *aligned*  $l\ (Node\ ts\ t)\ u$   
**and** *sorted-less*  $(leaves\ (Node\ ts\ t))$   
**and** *split*  $ts\ x = (ls, (sub, sep)\#rs)$   
**shows**  $del\text{-list}\ x\ (leaves\text{-list}\ ((sub, sep)\#rs)\ @\ leaves\ t) = del\text{-list}\ x\ (leaves\ sub)\ @\ leaves\text{-list}\ rs\ @\ leaves\ t$   
*<proof>*

**thm** *del-list-idem*

**lemma** *del-inorder:*

**assumes**  $k > 0$   
**and** *root-order*  $k\ t$   
**and** *bal*  $t$   
**and** *sorted-less*  $(leaves\ t)$   
**and** *aligned*  $l\ t\ u$   
**and**  $l < x\ x \leq u$   
**shows**  $leaves\ (del\ k\ x\ t) = del\text{-list}\ x\ (leaves\ t) \wedge aligned\ l\ (del\ k\ x\ t)\ u$   
*<proof>*

**lemma** *del-Linorder:*

**assumes**  $k > 0$   
**and** *root-order*  $k\ t$   
**and** *bal*  $t$   
**and** *sorted-less*  $(leaves\ t)$   
**and** *Laligned*  $t\ u$



**and**  $x \leq u$   
**shows**  $\text{leaves } (\text{del } k \ x \ t) = \text{del-list } x \ (\text{leaves } t) \wedge \text{Laligned } (\text{del } k \ x \ t) \ u$   
 $\langle \text{proof} \rangle$

**lemma** *reduce-root-order*:  $\llbracket k > 0; \text{almost-order } k \ t \rrbracket \implies \text{root-order } k \ (\text{reduce-root } t)$   
 $\langle \text{proof} \rangle$

**lemma** *reduce-root-bal*:  $\text{bal } (\text{reduce-root } t) = \text{bal } t$   
 $\langle \text{proof} \rangle$

**lemma** *reduce-root-inorder*:  $\text{leaves } (\text{reduce-root } t) = \text{leaves } t$   
 $\langle \text{proof} \rangle$

**lemma** *reduce-root-Laligned*:  $\text{Laligned } (\text{reduce-root } t) \ u = \text{Laligned } t \ u$   
 $\langle \text{proof} \rangle$

**lemma** *delete-order*:  $\llbracket k > 0; \text{bal } t; \text{root-order } k \ t; \text{sorted-less } (\text{leaves } t) \rrbracket \implies$   
 $\text{root-order } k \ (\text{delete } k \ x \ t)$   
 $\langle \text{proof} \rangle$

**lemma** *delete-bal*:  $\llbracket k > 0; \text{bal } t; \text{root-order } k \ t \rrbracket \implies \text{bal } (\text{delete } k \ x \ t)$   
 $\langle \text{proof} \rangle$

**lemma** *delete-Linorder*:  
**assumes**  $k > 0 \ \text{root-order } k \ t \ \text{sorted-less } (\text{leaves } t) \ \text{Laligned } t \ u \ \text{bal } t \ x \leq u$   
**shows**  $\text{leaves } (\text{delete } k \ x \ t) = \text{del-list } x \ (\text{leaves } t)$   
**and**  $\text{Laligned } (\text{delete } k \ x \ t) \ u$   
 $\langle \text{proof} \rangle$

**corollary** *delete-Linorder-top*:  
**assumes**  $k > 0 \ \text{root-order } k \ t \ \text{sorted-less } (\text{leaves } t) \ \text{Laligned } t \ \text{top} \ \text{bal } t$   
**shows**  $\text{leaves } (\text{delete } k \ x \ t) = \text{del-list } x \ (\text{leaves } t)$   
**and**  $\text{Laligned } (\text{delete } k \ x \ t) \ \text{top}$   
 $\langle \text{proof} \rangle$

## 7.5 Set specification by inorder

**fun** *invar-leaves* **where**  $\text{invar-leaves } k \ t = ($   
 $\text{bal } t \wedge$   
 $\text{root-order } k \ t \wedge$   
 $\text{Laligned } t \ \text{top}$   
 $)$

**interpretation** *S-ordered*: *Set-by-Ordered* **where**  
 $\text{empty} = \text{empty-bplustree}$  **and**  
 $\text{insert} = \text{insert } (\text{Suc } k)$  **and**

```

delete = delete (Suc k) and
isin = isin and
inorder = leaves and
inv = invar-leaves (Suc k)
⟨proof⟩

```

```

declare nodei.simps[simp del]

```

```

end

```

```

lemma sorted-ConsD: sorted-less (y # xs) ⇒ x ≤ y ⇒ x ∉ set xs
⟨proof⟩

```

```

lemma sorted-snocD: sorted-less (xs @ [y]) ⇒ y ≤ x ⇒ x ∉ set xs
⟨proof⟩

```

```

lemmas isin-simps2 = sorted-lems sorted-ConsD sorted-snocD

```

```

lemma isin-sorted: sorted-less (xs@a#ys) ⇒
(x ∈ set (xs@a#ys)) = (if x < a then x ∈ set xs else x ∈ set (a#ys))
⟨proof⟩

```

```

context split-list
begin

```

```

fun isin-list :: 'a ⇒ 'a list ⇒ bool where
  isin-list x ks = (case split-list ks x of
    (ls,Nil) ⇒ False |
    (ls,sep#rs) ⇒ sep = x
  )

```

```

fun insert-list where
  insert-list x ks = (case split-list ks x of
    (ls,Nil) ⇒ ls@[x] |
    (ls,sep#rs) ⇒ if sep = x then ks else ls@x#sep#rs
  )

```

```

fun delete-list where
  delete-list x ks = (case split-list ks x of
    (ls,Nil) ⇒ ks |
    (ls,sep#rs) ⇒ if sep = x then ls@rs else ks
  )

```

```

lemmas split-list-conc = split-list-req(1)

```

**lemmas** *split-list-sorted* = *split-list-req*(2,3)

**lemma** *isin-sorted-split-list*:

**assumes** *sorted-less xs*

**and** *split-list xs x = (ls, rs)*

**shows**  $(x \in \text{set } xs) = (x \in \text{set } rs)$

*<proof>*

**lemma** *isin-sorted-split-list-right*:

**assumes** *split-list ts x = (ls, sep#rs)*

**and** *sorted-less ts*

**shows**  $x \in \text{set } (sep\#rs) = (x = sep)$

*<proof>*

**theorem** *isin-list-set*:

**assumes** *sorted-less xs*

**shows** *isin-list x xs = (x ∈ set xs)*

*<proof>*

**lemma** *insert-sorted-split-list*:

**assumes** *sorted-less xs*

**and** *split-list xs x = (ls, rs)*

**shows** *ins-list x xs = ls @ ins-list x rs*

*<proof>*

**lemma** *insert-sorted-split-list-right*:

**assumes** *split-list ts x = (ls, sep#rs)*

**and** *sorted-less ts*

**and**  $x \neq sep$

**shows** *ins-list x (sep#rs) = (x#sep#rs)*

*<proof>*

**theorem** *insert-list-set*:

**assumes** *sorted-less xs*

**shows** *insert-list x xs = ins-list x xs*

*<proof>*

**lemma** *delete-sorted-split-list*:

**assumes** *sorted-less xs*

**and** *split-list xs x = (ls, rs)*

**shows** *del-list x xs = ls @ del-list x rs*

*<proof>*

**lemma** *delete-sorted-split-list-right*:

```

assumes split-list  $ts\ x = (ls, sep\#rs)$ 
and sorted-less  $ts$ 
and  $x \neq sep$ 
shows del-list  $x\ (sep\#rs) = sep\#rs$ 
⟨proof⟩

```

```

theorem delete-list-set:
assumes sorted-less  $xs$ 
shows delete-list  $x\ xs = del-list\ x\ xs$ 
⟨proof⟩

```

**end**

```

context split-full
begin

```

```

sublocale split-set split isin-list insert-list delete-list
⟨proof⟩

```

**end**

```

end
theory BPlusTree-Range
imports BPlusTree
         HOL-Data-Structures.Set-Specs
         HOL-Library.Sublist
         BPlusTree-Split
begin

```

Lrange describes all elements in a set that are greater or equal to  $l$ , a lower bounded range (with no upper bound)

```

definition Lrange where
  Lrange  $l\ X = \{x \in X. x \geq l\}$ 

```

```

definition lrange-filter  $l = filter\ (\lambda x. x \geq l)$ 

```

```

lemma lrange-filter-iff-Lrange: set (lrange-filter  $l\ xs$ ) = Lrange  $l$  (set  $xs$ )
⟨proof⟩

```

```

fun lrange-list where
  lrange-list  $l\ (x\#xs) = (if\ x \geq l\ then\ (x\#xs)\ else\ lrange-list\ l\ xs)\ |$ 
  lrange-list  $l\ [] = []$ 

```

```

lemma sorted-leq-lrange: sorted-wrt  $(\leq)\ xs \implies lrange-list\ (l::'a::linorder)\ xs =$ 
lrange-filter  $l\ xs$ 
⟨proof⟩

```

**lemma** *sorted-less-lrange*: *sorted-less xs*  $\implies$  *lrange-list* (*l*::'a::linorder) *xs* = *lrange-filter* *l xs*

*<proof>*

**lemma** *lrange-list-sorted*: *sorted-less (xs@x#ys)*  $\implies$   
*lrange-list l (xs@x#ys)* =  
*(if l < x then (lrange-list l xs)@x#ys else lrange-list l (x#ys))*  
*<proof>*

**lemma** *lrange-filter-sorted*: *sorted-less (xs@x#ys)*  $\implies$   
*lrange-filter l (xs@x#ys)* =  
*(if l < x then (lrange-filter l xs)@x#ys else lrange-filter l (x#ys))*  
*<proof>*

**lemma** *lrange-suffix*: *suffix (lrange-list l xs) xs*  
*<proof>*

**locale** *split-range* = *split-tree split*

**for** *split*::

*( 'a bplustree  $\times$  'a::{linorder,order-top} ) list  $\Rightarrow$  'a*  
 $\Rightarrow$  *( 'a bplustree  $\times$  'a ) list  $\times$  ( 'a bplustree  $\times$  'a ) list +*

**fixes** *lrange-list* :: 'a  $\Rightarrow$  ('a::{linorder,order-top}) list  $\Rightarrow$  'a list

**assumes** *lrange-list-req*:

*sorted-less ks*  $\implies$  *lrange-list l ks* = *lrange-filter l ks*

**begin**

**fun** *lrange*:: 'a bplustree  $\Rightarrow$  'a  $\Rightarrow$  'a list **where**

*lrange (Leaf ks) x* = *(lrange-list x ks)* |

*lrange (Node ts t) x* = (

*case split ts x of* (*-, (sub, sep)#rs*)  $\Rightarrow$  (

*lrange sub x @ leaves-list rs @ leaves t*

)

| (*-, []*)  $\Rightarrow$  *lrange t x*

)

*lrange proof*

**lemma** *lrange-sorted-split*:

**assumes** *Laligned (Node ts t) u*

**and** *sorted-less (leaves (Node ts t))*

**and** *split ts x = (ls, rs)*

**shows** *lrange-filter x (leaves (Node ts t)) = lrange-filter x (leaves-list rs @ leaves t)*

*<proof>*

**lemma** *lrange-sorted-split-right*:

**assumes**  $split\ ts\ x = (ls, (sub,sep)\#rs)$   
**and**  $sorted-less\ (leaves\ (Node\ ts\ t))$   
**and**  $Laligned\ (Node\ ts\ t)\ u$   
**shows**  $lrange-filter\ x\ (leaves-list\ ((sub,sep)\#rs)\ @\ leaves\ t) = lrange-filter\ x$   
 $(leaves\ sub)\@leaves-list\ rs\@leaves\ t$   
 $\langle proof \rangle$

**theorem**  $lrange-set$ :  
**assumes**  $sorted-less\ (leaves\ t)$   
**and**  $aligned\ l\ t\ u$   
**shows**  $lrange\ t\ x = lrange-filter\ x\ (leaves\ t)$   
 $\langle proof \rangle$

Now the alternative explanation that first obtains the correct leaf node and in a second step obtains the correct element from the leaf node.

**fun**  $leaf-nodes-lrange$ ::  $'a\ bplustree \Rightarrow 'a \Rightarrow 'a\ bplustree\ list$  **where**  
 $leaf-nodes-lrange\ (Leaf\ ks)\ x = [Leaf\ ks]\ |$   
 $leaf-nodes-lrange\ (Node\ ts\ t)\ x = ($   
 $\quad case\ split\ ts\ x\ of\ (-,(sub,sep)\#rs) \Rightarrow ($   
 $\quad\quad leaf-nodes-lrange\ sub\ x\ @\ leaf-nodes-list\ rs\ @\ leaf-nodes\ t$   
 $\quad)$   
 $\quad | (-,[]) \Rightarrow leaf-nodes-lrange\ t\ x$   
 $)$

$lrange$  proof

**lemma**  $concat-leaf-nodes-leaves-list$ :  $(concat\ (map\ leaves\ (leaf-nodes-list\ ts))) =$   
 $leaves-list\ ts$   
 $\langle proof \rangle$

**theorem**  $leaf-nodes-lrange-set$ :  
**assumes**  $sorted-less\ (leaves\ t)$   
**and**  $aligned\ l\ t\ u$   
**shows**  $suffix\ (lrange-filter\ x\ (leaves\ t))\ (concat\ (map\ leaves\ (leaf-nodes-lrange\ t$   
 $x)))$   
 $\langle proof \rangle$

**lemma**  $leaf-nodes-lrange-not-empty$ :  $\exists\ ks\ list.\ leaf-nodes-lrange\ t\ x = (Leaf\ ks)\#list$   
 $\wedge\ (Leaf\ ks) \in set\ (leaf-nodes\ t)$   
 $\langle proof \rangle$

Note that, conveniently, this argument is purely syntactic, we do not need to show that this has anything to do with linear orders

**lemma**  $leaf-nodes-lrange-pre-lrange$ :  $leaf-nodes-lrange\ t\ x = (Leaf\ ks)\#list \implies$   
 $lrange-list\ x\ ks\ @\ (concat\ (map\ leaves\ list)) = lrange\ t\ x$   
 $\langle proof \rangle$

We finally obtain a function that is way easier to reason about in the imperative setting

```

fun concat-leaf-nodes-lrange where
  concat-leaf-nodes-lrange t x = (case leaf-nodes-lrange t x of (Leaf ks)#list ⇒
lrange-list x ks @ (concat (map leaves list)))

lemma concat-leaf-nodes-lrange-lrange: concat-leaf-nodes-lrange t x = lrange t x
⟨proof⟩

end

context split-list
begin

definition lrange-split where
lrange-split l xs = (case split-list xs l of (ls,rs) ⇒ rs)

lemma lrange-filter-split:
  assumes sorted-less xs
    and split-list xs l = (ls,rs)
  shows lrange-list l xs = rs
  find-theorems split-list
⟨proof⟩

lemma lrange-split-req:
  assumes sorted-less xs
  shows lrange-split l xs = lrange-filter l xs
  ⟨proof⟩

end

context split-full
begin

sublocale split-range split lrange-split
  ⟨proof⟩

end

end
theory BPlusTree-SplitCE
  imports
    BPlusTree-Set
    BPlusTree-Range
begin

global-interpretation bplustree-linear-search-list: split-list linear-split-list
defines bplustree-ls-isin-list = bplustree-linear-search-list.isin-list
and bplustree-ls-insert-list = bplustree-linear-search-list.insert-list
and bplustree-ls-delete-list = bplustree-linear-search-list.delete-list
and bplustree-ls-lrange-list = bplustree-linear-search-list.lrange-split

```

```

    <proof>

declare bplustree-linear-search-list.isin-list.simps[code]
declare bplustree-linear-search-list.insert-list.simps[code]
declare bplustree-linear-search-list.delete-list.simps[code]

global-interpretation bplustree-linear-search:
    split-full linear-split linear-split-list

defines bplustree-ls-isin = bplustree-linear-search.isin
and bplustree-ls-ins = bplustree-linear-search.ins
and bplustree-ls-insert = bplustree-linear-search.insert
and bplustree-ls-del = bplustree-linear-search.del
and bplustree-ls-delete = bplustree-linear-search.delete
and bplustree-ls-lrange = bplustree-linear-search.lrange
    <proof>

lemma [code]: bplustree-ls-isin (Leaf ks) x = bplustree-ls-isin-list x ks
    <proof>
declare bplustree-linear-search.isin.simps(2)[code]

lemma [code]: bplustree-ls-ins k x (Leaf ks) =
bplustree-linear-search.Lnodei k (bplustree-ls-insert-list x ks)
    <proof>
declare bplustree-linear-search.ins.simps(2)[code]

lemma [code]: bplustree-ls-del k x (Leaf ks) =
Leaf (bplustree-ls-delete-list x ks)
    <proof>
declare bplustree-linear-search.del.simps(2)[code]

find-theorems bplustree-ls-isin

Some examples follow to show that the implementation works and the above
lemmas make sense. The examples are visualized in the thesis.

abbreviation bplustreeq ≡ bplustree-ls-isin
abbreviation bplustreei ≡ bplustree-ls-insert
abbreviation bplustreed ≡ bplustree-ls-delete

definition uint8-max ≡  $2^8 - 1 :: nat$ 
declare uint8-max-def[simp]

typedef uint8 = {n :: nat. n ≤ uint8-max}
    <proof>

setup-lifting type-definition-uint8

```



```

instantiation uint8 :: linorder
begin

lift-definition less-eq-uint8 :: uint8  $\Rightarrow$  uint8  $\Rightarrow$  bool
  is (less-eq::nat  $\Rightarrow$  nat  $\Rightarrow$  bool)  $\langle$ proof $\rangle$ 

lift-definition less-uint8 :: uint8  $\Rightarrow$  uint8  $\Rightarrow$  bool
  is (less::nat  $\Rightarrow$  nat  $\Rightarrow$  bool)  $\langle$ proof $\rangle$ 

instance
   $\langle$ proof $\rangle$ 
end

instantiation uint8 :: order-top
begin

lift-definition top-uint8 :: uint8 is uint8-max::nat
   $\langle$ proof $\rangle$ 

instance
   $\langle$ proof $\rangle$ 
end

instantiation uint8 :: numeral
begin

lift-definition one-uint8 :: uint8 is 1::nat
   $\langle$ proof $\rangle$ 

lift-definition plus-uint8 :: uint8  $\Rightarrow$  uint8  $\Rightarrow$  uint8
  is  $\lambda a b. \min (a + b) \text{uint8-max}$ 
   $\langle$ proof $\rangle$ 

instance  $\langle$ proof $\rangle$ 
end

instantiation uint8 :: equal
begin

lift-definition equal-uint8 :: uint8  $\Rightarrow$  uint8  $\Rightarrow$  bool
  is (=)  $\langle$ proof $\rangle$ 

instance  $\langle$ proof $\rangle$ 
end

value uint8-max

```

```

value let k=2::nat; x::uint8 bplustree = (Node [(Node [(Leaf [1,2], 2),(Leaf [3,4],
4),(Leaf [5,6,7], 8)] (Leaf [9,10]), 10)] (Node [(Leaf [11,12,13,14], 14), (Leaf
[15,17], 20)] (Leaf [21,22,23]))) in
  root-order k x
value let k=2::nat; x::uint8 bplustree = (Node [(Node [(Leaf [1,2], 2),(Leaf [3,4],
4),(Leaf [5,6,7], 8)] (Leaf [9,10]), 10)] (Node [(Leaf [11,12,13,14], 14), (Leaf
[15,17], 20)] (Leaf [21,22,23]))) in
  bal x
value let k=2::nat; x::uint8 bplustree = (Node [(Node [(Leaf [1,2], 2),(Leaf [3,4],
4),(Leaf [5,6,7], 8)] (Leaf [9,10]), 10)] (Node [(Leaf [11,12,13,14], 14), (Leaf
[15,17], 20)] (Leaf [50,55,56]))) in
  sorted-less (leaves x)
value let k=2::nat; x::uint8 bplustree = (Node [(Node [(Leaf [1,2], 2),(Leaf [3,4],
4),(Leaf [5,6,7], 8)] (Leaf [9,10]), 10)] (Node [(Leaf [11,12,13,14], 14), (Leaf
[15,17], 20)] (Leaf [50,55,56]))) in
  Laligned x top
value let k=2::nat; x::uint8 bplustree = (Node [(Node [(Leaf [1,2], 2),(Leaf [3,4],
4),(Leaf [5,6,7], 8)] (Leaf [9,10]), 10)] (Node [(Leaf [11,12,13,14], 14), (Leaf
[15,17], 20)] (Leaf [50,55,56]))) in
  x
value let k=2::nat; x::uint8 bplustree = (Node [(Node [(Leaf [1,2], 2),(Leaf [3,4],
4),(Leaf [5,6,7], 8)] (Leaf [9,10]), 10)] (Node [(Leaf [11,12,13,14], 14), (Leaf
[15,17], 20)] (Leaf [50,55,56]))) in
  bplustreeq x 4
value let k=2::nat; x::uint8 bplustree = (Node [(Node [(Leaf [1,2], 2),(Leaf [3,4],
4),(Leaf [5,6,7], 8)] (Leaf [9,10]), 10)] (Node [(Leaf [11,12,13,14], 14), (Leaf
[15,17], 20)] (Leaf [50,55,56]))) in
  bplustreeq x 20
value let k=2::nat; x::uint8 bplustree = (Node [(Node [(Leaf [1,2], 2),(Leaf [3,4],
4),(Leaf [5,6,7], 8)] (Leaf [9,10]), 10)] (Node [(Leaf [11,12,13,14], 14), (Leaf
[15,17], 20)] (Leaf [50,55,56]))) in
  bplustreei k 9 x
value let k=2::nat; x::uint8 bplustree = (Node [(Node [(Leaf [1,2], 2),(Leaf [3,4],
4),(Leaf [5,6,7], 8)] (Leaf [9,10]), 10)] (Node [(Leaf [11,12,13,14], 14), (Leaf
[15,17], 20)] (Leaf [50,55,56]))) in
  bplustreei k 1 (bplustreei k 9 x)
value let k=2::nat; x::uint8 bplustree = (Node [(Node [(Leaf [1,2], 2),(Leaf [3,4],
4),(Leaf [5,6,7], 8)] (Leaf [9,10]), 10)] (Node [(Leaf [11,12,13,14], 14), (Leaf
[15,17], 20)] (Leaf [50,55,56]))) in
  bplustreed k 10 (bplustreei k 1 (bplustreei k 9 x))
value let k=2::nat; x::uint8 bplustree = (Node [(Node [(Leaf [1,2], 2),(Leaf [3,4],
4),(Leaf [5,6,7], 8)] (Leaf [9,10]), 10)] (Node [(Leaf [11,12,13,14], 14), (Leaf
[15,17], 20)] (Leaf [50,55,56]))) in
  bplustreed k 3 (bplustreed k 10 (bplustreei k 1 (bplustreei k 9 x)))

```

**end**

## References

- [1] Rudolf Bayer and Edward M. McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 1:173–189, 1972. doi:10.1007/BF00288683. URL <https://doi.org/10.1007/BF00288683>.
- [2] Peter Lammich. The imperative refinement framework. *Archive of Formal Proofs*, August 2016. ISSN 2150-914x. [https://isa-afp.org/entries/Refine\\_Imperative\\_HOL.html](https://isa-afp.org/entries/Refine_Imperative_HOL.html), Formal proof development.
- [3] Niels Mündler. A verified imperative implementation of b-trees. Bachelor's thesis, Technische Universität München, München, 2021. URL <https://mediatum.ub.tum.de/1596550>.