

Region Quadtrees

Tobias Nipkow
Technical University of Munich

February 6, 2026

Abstract

These theories formalize *region quadtrees*, which are traditionally used to represent two-dimensional images of (black and white) pixels. Building on these quadtrees, addition and multiplication of recursive block matrices are verified. The generalization of region quadtrees to k dimensions is also formalized.

1 Introduction

These theories formalize so-called *region quadtrees*, as opposed to *point quadtrees* [5, 6, 1]. The following variants are covered:

- Ordinary region quadtrees.
- Block matrices based on region quadtrees. Operations: matrix addition and multiplication. Based on the work of Wise [7, 8, 9, 10, 11].
- A k -dimensional generalization of region quadtrees. This is inspired by the k -dimensional point trees by Bentley [2, 3] which have already been formalized by Rau [4].

For the details of the operations covered see the individual theories.

Contents

1	Introduction	1
2	Quad Tree Basics	3
3	Quad Trees	3
3.1	Compression	3
3.2	Abstraction function	4
3.3	Boolean Quadtrees	4
3.3.1	Abstraction of boolean quadtrees to sets of points	4

3.3.2	Union, Intersection Difference and Complement	6
3.4	Operation <i>put</i>	8
3.5	Extract Square	8
3.6	From Matrix to Quadtree	9
3.6.1	Matrix as list of lists	9
3.7	From Quadtree to Matrix	10
4	Block Matrices via Quad Trees	11
4.1	Square Matrices	11
4.2	Matrix Lemmas	12
4.3	Real Quad Trees and Abstraction to Matrices	12
4.4	Matrix Operations on Trees	13
4.5	Correctness of Quad Tree Implementations	14
4.5.1	<i>add</i>	14
4.5.2	<i>mult</i>	14
5	K-dimensional Region Trees	15
5.1	Subtree	16
5.2	Shifting a coordinate by a boolean vector	16
5.3	Points in a tree	17
5.4	Compression	17
5.5	Extracting a point from a tree	18
5.6	Modifying a point in a tree	18
5.7	Union	19
6	K-dimensional Region Trees - Version 2	20
6.1	Subtree	21
6.2	Shifting a coordinate by a boolean vector	22
6.3	Points in a tree	22
6.4	Compression	23
6.5	Union	23
6.6	Extracting a point from a tree	24
7	K-dimensional Region Trees - Nested Trees	24

2 Quad Tree Basics

```
theory Quad-Base
imports HOL-Library.Tree
begin

datatype 'a qtree = L 'a | Q 'a qtree 'a qtree 'a qtree 'a qtree

instantiation qtree :: (type)height
begin

fun height-qtree :: 'a qtree  $\Rightarrow$  nat where
  height (L _) = 0 |
  height (Q t0 t1 t2 t3) =
    Max {height t0, height t1, height t2, height t3} + 1

instance <proof>

end

end
```

3 Quad Trees

```
theory Quad-Tree
imports Quad-Base
begin

lemma mod-minus:  $\llbracket i < 2*m; \neg i < m \rrbracket \Longrightarrow i \bmod m = i - (m::nat)$ 
  <proof>

definition select :: bool  $\Rightarrow$  bool  $\Rightarrow$  'a  $\Rightarrow$  'a  $\Rightarrow$  'a  $\Rightarrow$  'a  $\Rightarrow$  'a where
  select x y t0 t1 t2 t3 =
    (if x then
      if y then t0 else t1
    else
      if y then t2 else t3)

abbreviation qf where
  qf q f i j d  $\equiv$  q (f i j) (f i (j+d)) (f (i+d) j) (f (i+d) (j+d))
```

3.1 Compression

```
fun compressed :: 'a qtree  $\Rightarrow$  bool where
  compressed (L _) = True |
  compressed (Q t0 t1 t2 t3) = ((compressed t0  $\wedge$  compressed t1  $\wedge$  compressed t2
 $\wedge$  compressed t3)
   $\wedge$   $\neg$  ( $\exists x. t0 = L x \wedge t1 = t0 \wedge t2 = t0 \wedge t3 = t0$ ))
```

```

fun Qc :: 'a qtree ⇒ 'a qtree ⇒ 'a qtree ⇒ 'a qtree ⇒ 'a qtree where
  Qc (L x0) (L x1) (L x2) (L x3) =
    (if x0=x1 ∧ x1=x2 ∧ x2=x3 then L x0 else Q (L x0) (L x1) (L x2) (L x3)) |
  Qc t0 t1 t2 t3 = Q t0 t1 t2 t3

```

Compressing version of Q :

```

lemma compressed-Qc: [[compressed t0; compressed t1; compressed t2; compressed
t3 ]] ⇒
  compressed (Qc t0 t1 t2 t3)
  <proof>

```

```

lemma compressedQD: compressed (Q t1 t2 t3 t4)
⇒ compressed t1 ∧ compressed t2 ∧ compressed t3 ∧ compressed t4
  <proof>

```

```

lemma height-Qc-Q: [[ height s0 ≤ n; height s1 ≤ n; height s2 ≤ n; height s3 ≤
n ]]
⇒ height (Qc s0 s1 s2 s3) ≤ Suc n
  <proof>

```

Modify a quadrant addressed by x and y , and put things back together with Qc :

```

fun modify :: ('a qtree ⇒ 'a qtree) ⇒ bool ⇒ bool ⇒ 'a qtree *'a qtree *'a qtree
*'a qtree ⇒ 'a qtree where
  modify f x y (t0, t1, t2, t3) =
    (if x then
      if y then Qc (f t0) t1 t2 t3 else Qc t0 (f t1) t2 t3
    else
      if y then Qc t0 t1 (f t2) t3 else Qc t0 t1 t2 (f t3))

```

3.2 Abstraction function

```

fun get :: nat ⇒ 'a qtree ⇒ nat ⇒ nat ⇒ 'a where
  get n (L b) - - = b |
  get (Suc n) (Q t0 t1 t2 t3) i j =
  get n (select (i < 2^n) (j < 2^n) t0 t1 t2 t3) (i mod 2^n) (j mod 2^n)

```

```

lemma get-Qc:
  height(Q t0 t1 t2 t3) ≤ n ⇒ get n (Qc t0 t1 t2 t3) i j = get n (Q t0 t1 t2 t3) i
j
  <proof>

```

3.3 Boolean Quadrees

```

type-synonym qtb = bool qtree

```

3.3.1 Abstraction of boolean quadrees to sets of points

Superseded by the more general get abstraction.

type-synonym $points = (nat \times nat) \text{ set}$

abbreviation $sq :: nat \Rightarrow points$ **where**

$$sq (n::nat) \equiv \{0..<2^{\wedge} n\} \times \{0..<2^{\wedge} n\}$$

definition $shift :: nat \Rightarrow nat \Rightarrow nat * nat \Rightarrow nat * nat$ **where**

$$shift \ di \ dj = (\lambda(i,j). (i+di, j+dj))$$

lemma $shift\text{-}pair[simp]: shift \ di \ dj \ (a,b) = (a+di,b+dj)$

$\langle proof \rangle$

lemma $in\text{-}shift\text{-}image: (x,y) \in shift \ di \ dj \ 'M \longleftrightarrow di \leq x \wedge dj \leq y \wedge (x-di,y-dj) \in M$

$\langle proof \rangle$

lemma $inj\text{-}shift: inj \ (shift \ i \ j)$

$\langle proof \rangle$

lemma $shift\text{-}disj\text{-}shift: \llbracket s \subseteq sq \ n; s' \subseteq sq \ n;$

$$i \geq i' + 2^{\wedge} n \vee i' \geq i + 2^{\wedge} n \vee j \geq j' + 2^{\wedge} n \vee j' \geq j + 2^{\wedge} n \rrbracket \Longrightarrow$$

$$shift \ i \ j \ 's \cap shift \ i' \ j' \ 's' = \{\}$$

$\langle proof \rangle$

Convention: $A, B :: points$

The layout of the 4 subquadrants $Q \ t0 \ t1 \ t2 \ t3 / Qsq \ A0 \ A1 \ A2 \ A3: 1 \ 3 \ 0 \ 2$ That is, the x and y coordinates are shifted as follows (where $1 = 2^n$):
 $(0,1) \ (1,1) \ (0,0) \ (1,0)$

definition $Qsq :: nat \Rightarrow points \Rightarrow points \Rightarrow points \Rightarrow points \Rightarrow points$ **where**

$$Qsq \ n \ A0 \ A1 \ A2 \ A3 =$$

$$shift \ 0 \ 0 \ 'A0 \cup shift \ 0 \ (2^{\wedge} n) \ 'A1 \cup shift \ (2^{\wedge} n) \ 0 \ 'A2 \cup shift \ (2^{\wedge} n) \ (2^{\wedge} n) \ 'A3$$

lemma $sq\text{-}Suc\text{-}Qsq: \{0..<2 * 2^{\wedge} n\} \times \{0..<2 * 2^{\wedge} n\} = Qsq \ n \ (sq \ n) \ (sq \ n) \ (sq \ n) \ (sq \ n)$

$\langle proof \rangle$

fun $points :: nat \Rightarrow qtb \Rightarrow (nat * nat) \text{ set}$ **where**

$$points \ n \ (L \ b) = (if \ b \ then \ sq \ n \ else \ \{\}) \ |$$

$points \ (Suc \ n) \ (Q \ t0 \ t1 \ t2 \ t3) = Qsq \ n \ (points \ n \ t0) \ (points \ n \ t1) \ (points \ n \ t2) \ (points \ n \ t3)$

lemma $points\text{-}subset: height \ t \leq n \Longrightarrow points \ n \ t \subseteq sq \ n$

$\langle proof \rangle$

lemma $point\text{-}Suc\text{-}Qc[simp]: points \ (Suc \ n) \ (Qc \ t0 \ t1 \ t2 \ t3) = points \ (Suc \ n) \ (Q \ t0 \ t1 \ t2 \ t3)$

$\langle proof \rangle$

lemma $get\text{-}points: \llbracket height \ t \leq n; (i,j) \in sq \ n \rrbracket \Longrightarrow get \ n \ t \ i \ j = ((i,j) \in points \ n \ t)$

<proof>

3.3.2 Union, Intersection Difference and Complement

fun *union* :: *qtb* \Rightarrow *qtb* \Rightarrow *qtb* **where**

union (*L b*) *t* = (*if b then L True else t*) |
union *t* (*L b*) = (*if b then L True else t*) |
union (*Q s1 s2 s3 s4*) (*Q t1 t2 t3 t4*) = *Qc* (*union s1 t1*) (*union s2 t2*) (*union s3 t3*) (*union s4 t4*)

fun *inter* :: *qtb* \Rightarrow *qtb* \Rightarrow *qtb* **where**

inter (*L b*) *t* = (*if b then t else L False*) |
inter *t* (*L b*) = (*if b then t else L False*) |
inter (*Q s1 s2 s3 s4*) (*Q t1 t2 t3 t4*) = *Qc* (*inter s1 t1*) (*inter s2 t2*) (*inter s3 t3*) (*inter s4 t4*)

fun *negate* :: *qtb* \Rightarrow *qtb* **where**

negate (*L b*) = *L*(\neg *b*) |
negate (*Q t1 t2 t3 t4*) = *Q* (*negate t1*) (*negate t2*) (*negate t3*) (*negate t4*)

fun *diff* :: *qtb* \Rightarrow *qtb* \Rightarrow *qtb* **where**

diff (*L b*) *t* = (*if b then negate t else L False*) |
diff *t* (*L b*) = (*if b then L False else t*) |
diff (*Q s1 s2 s3 s4*) (*Q t1 t2 t3 t4*) = *Qc* (*diff s1 t1*) (*diff s2 t2*) (*diff s3 t3*) (*diff s4 t4*)

lemma *Qsq-union*:

Qsq n A0 A1 A2 A3 \cup *Qsq n B0 B1 B2 B3* = *Qsq n* (*A0* \cup *B0*) (*A1* \cup *B1*) (*A2* \cup *B2*) (*A3* \cup *B3*)
<proof>

lemma *points-union*:

max (*height t1*) (*height t2*) $\leq n \implies$ *points n* (*union t1 t2*) = *points n t1* \cup *points n t2*
<proof>

lemma *height-union*: *height* (*union t1 t2*) \leq *max* (*height t1*) (*height t2*)

<proof>

lemma *height-union2*: \llbracket *height t1* $\leq n$; *height t2* $\leq n$ $\rrbracket \implies$ *height* (*union t1 t2*) $\leq n$

<proof>

lemma *get-union*:

max (*height t1*) (*height t2*) $\leq n \implies$ *get n* (*union t1 t2*) *i j* = (*get n t1 i j* \vee *get n t2 i j*)
<proof>

lemma *compressed-union*: $\text{compressed } t1 \implies \text{compressed } t2 \implies \text{compressed}(\text{union } t1 \ t2)$

<proof>

lemma *Qsq-inter*:

$\llbracket A0 \subseteq \text{sq } n; A1 \subseteq \text{sq } n; A2 \subseteq \text{sq } n; A3 \subseteq \text{sq } n;$
 $B0 \subseteq \text{sq } n; B1 \subseteq \text{sq } n; B2 \subseteq \text{sq } n; B3 \subseteq \text{sq } n \rrbracket$
 $\implies \text{Qsq } n \ A0 \ A1 \ A2 \ A3 \cap \text{Qsq } n \ B0 \ B1 \ B2 \ B3 = \text{Qsq } n \ (A0 \cap B0) \ (A1 \cap B1)$
 $(A2 \cap B2) \ (A3 \cap B3)$

<proof>

lemma *points-inter*: $n \geq \max(\text{height } t1) \ (\text{height } t2) \implies$

$\text{points } n \ (\text{inter } t1 \ t2) = \text{points } n \ t1 \cap \text{points } n \ t2$

<proof>

lemma *height-inter*: $\text{height}(\text{inter } t1 \ t2) \leq \max(\text{height } t1) \ (\text{height } t2)$

<proof>

lemma *height-inter2*: $\llbracket \text{height } t1 \leq n; \text{height } t2 \leq n \rrbracket \implies \text{height}(\text{inter } t1 \ t2) \leq n$

<proof>

lemma *get-inter*:

$\llbracket \text{height } t1 \leq n; \text{height } t2 \leq n \rrbracket \implies \text{get } n \ (\text{inter } t1 \ t2) \ i \ j = (\text{get } n \ t1 \ i \ j \wedge \text{get } n \ t2 \ i \ j)$

<proof>

lemma *compressed-inter*: $\text{compressed } t1 \implies \text{compressed } t2 \implies \text{compressed}(\text{inter } t1 \ t2)$

<proof>

lemma *Qsq-diff*: $\llbracket B0 \subseteq \text{sq } n; B1 \subseteq \text{sq } n; B2 \subseteq \text{sq } n; B3 \subseteq \text{sq } n; A0 \subseteq \text{sq } n; A1 \subseteq \text{sq } n; A2 \subseteq \text{sq } n; A3 \subseteq \text{sq } n \rrbracket \implies$

$\text{Qsq } n \ B0 \ B1 \ B2 \ B3 - \text{Qsq } n \ A0 \ A1 \ A2 \ A3 = \text{Qsq } n \ (B0 - A0) \ (B1 - A1)$
 $(B2 - A2) \ (B3 - A3)$

<proof>

lemma *points-negate*: $n \geq \text{height } t \implies \text{points } n \ (\text{negate } t) = \text{sq } n - \text{points } n \ t$

<proof>

lemma *negate-eq-L-iff*: $\text{compressed } t \implies \text{negate } t = L \ x \longleftrightarrow t = L(\neg x)$

<proof>

lemma *compressed-negate*: $\text{compressed } t \implies \text{compressed}(\text{negate } t)$

<proof>

lemma *points-diff*: $n \geq \max(\text{height } t1) \ (\text{height } t2) \implies$

$\text{points } n \ (\text{diff } t1 \ t2) = \text{points } n \ t1 - \text{points } n \ t2$

<proof>

lemma *compressed-diff*: $\text{compressed } t1 \implies \text{compressed } t2 \implies \text{compressed}(\text{diff } t1 \ t2)$

<proof>

3.4 Operation put

fun *put* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow 'a \Rightarrow \text{nat} \Rightarrow 'a \text{ qtree} \Rightarrow 'a \text{ qtree}$ **where**

put *i j a 0* (L -) = L a |

put *i j a* (Suc *n*) *t* = *modify* (*put* (*i mod 2ⁿ*) (*j mod 2ⁿ*) *a n*) (*i < 2ⁿ*) (*j < 2ⁿ*)

(*case t of* L *b* \Rightarrow (L *b*, L *b*, L *b*, L *b*) | Q *t0 t1 t2 t3* \Rightarrow (*t0,t1,t2,t3*))

lemma *points-put*: $\llbracket \text{height } t \leq n; (i,j) \in \text{sq } n \rrbracket \implies$

points n (*put i j b n t*) = (*if b then points n t* \cup $\{(i,j)\}$ *else points n t* - $\{(i,j)\}$)

<proof>

lemma *height-put*: $\text{height } t \leq n \implies \text{height}(\text{put } i \ j \ a \ n \ t) \leq n$

<proof>

lemma *get-put*: $\llbracket \text{height } t \leq n; (i,j) \in \text{sq } n; (i',j') \in \text{sq } n \rrbracket \implies$

get n (*put i j a n t*) *i' j'* = (*if i'=i* \wedge *j'=j* *then a* *else get n t i' j'*)

<proof>

lemma *compressed-put*:

$\llbracket \text{height } t \leq n; \text{compressed } t \rrbracket \implies \text{compressed}(\text{put } i \ j \ a \ n \ t)$

<proof>

3.5 Extract Square

fun *get-sq* :: $\text{nat} \Rightarrow 'a \text{ qtree} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ qtree}$ **where**

get-sq n (L *b*) *m i j* = L *b* |

get-sq n t 0 i j = L (*get n t i j*) |

get-sq (Suc *n*) (Q *t0 t1 t2 t3*) (Suc *m*) *i j* =

(*if i mod 2ⁿ + 2^(m+1) \leq 2ⁿ \wedge j mod 2ⁿ + 2^(m+1) \leq 2ⁿ*
then get-sq n (*select* (*i < 2ⁿ*) (*j < 2ⁿ*) *t0 t1 t2 t3*) (*m+1*) (*i mod 2ⁿ*) (*j mod 2ⁿ*)

else qf Qc (*get-sq* (Suc *n*) (Q *t0 t1 t2 t3*) *m*) *i j* (2^{*m*})

lemma *shift-shift*: $\text{shift } i \ j \ 's = \text{shift } (i+i') \ (j+j') \ 's$

<proof>

lemma *shift-shift2*: $\text{shift } i \ j \ 's = \text{shift } (i'+i) \ (j'+j) \ 's$

<proof>

lemma *shift-split*: $\text{shift } i \ j \ 's =$

shift (*i - i mod 2ⁿ*) (*j - j mod 2ⁿ*) ('*shift* (*i mod 2ⁿ*) (*j mod 2ⁿ*) 's)

<proof>

lemma *plus-pow-aux*: $(i::\text{nat}) + 2^m \leq 2 * 2^n \implies i < 2 * 2^n$

<proof>

lemma *Qsq-lem*: $\llbracket A0 \subseteq \text{sq } n; A1 \subseteq \text{sq } n; A2 \subseteq \text{sq } n; A3 \subseteq \text{sq } n;$
 $i + 2^{\wedge} m \leq 2^{\wedge} \text{Suc } n; j + 2^{\wedge} m \leq 2^{\wedge} \text{Suc } n;$
 $i \bmod 2^{\wedge} n + 2^{\wedge} m \leq 2^{\wedge} n; j \bmod 2^{\wedge} n + 2^{\wedge} m \leq 2^{\wedge} n \rrbracket \implies$
 $Q\text{sq } n A0 A1 A2 A3 \cap \text{shift } i j \text{ ' sq } m =$
 $\text{shift } (i - i \bmod 2^{\wedge} n) (j - j \bmod 2^{\wedge} n) \text{ ' select } (i < 2^{\wedge} n) (j < 2^{\wedge} n) A0 A1$
 $A2 A3 \cap \text{shift } i j \text{ ' sq } m$
<proof>

lemma *f-select*: $f (\text{select } x y a b c d) = \text{select } x y (f a) (f b) (f c) (f d)$
<proof>

lemma *height-get-sq*: $m \leq n \implies \text{height } (\text{get-sq } n t m i j) \leq m$
<proof>

lemma *shift-Qsq*: $\text{shift } i j \text{ ' } Q\text{sq } n A0 A1 A2 A3 =$
 $Q\text{sq } n (\text{shift } i j \text{ ' } A0) (\text{shift } i j \text{ ' } A1) (\text{shift } i j \text{ ' } A2) (\text{shift } i j \text{ ' } A3)$
<proof>

lemma *points-get-sq*:
 $\llbracket \text{height } t \leq n; i + 2^{\wedge} m \leq 2^{\wedge} n; j + 2^{\wedge} m \leq 2^{\wedge} n \rrbracket \implies$
 $\text{shift } i j \text{ ' points } m (\text{get-sq } n t m i j) = \text{points } n t \cap (\text{shift } i j \text{ ' sq } m)$
<proof>

lemma *get-get-sq*:
 $\llbracket \text{height } t \leq n; i + 2^{\wedge} m \leq 2^{\wedge} n; j + 2^{\wedge} m \leq 2^{\wedge} n; i' < 2^{\wedge} m; j' < 2^{\wedge} m \rrbracket \implies$
 $\text{get } m (\text{get-sq } n t m i j) i' j' = \text{get } n t (i+i') (j+j')$
<proof>

lemma *compressed-get-sq*:
 $\llbracket \text{height } t \leq n; \text{compressed } t \rrbracket \implies \text{compressed } (\text{get-sq } n t m i j)$
<proof>

3.6 From Matrix to Quadtree

3.6.1 Matrix as list of lists

type-synonym *'a mx* = *'a list list*

definition *sq-mx* $n \text{ mx} = (\text{length } \text{mx} = 2^{\wedge} n \wedge (\forall xs \in \text{set } \text{mx}. \text{length } xs = 2^{\wedge} n))$

lemma *sq-mx-0*: $\text{sq-mx } 0 \text{ mx} = (\exists x. \text{mx} = [[x]])$
<proof>

Decompose matrix into submatrices

definition *decomp where*

$\text{decomp } n \text{ mx} = (\text{let } \text{mx01} = \text{take } (2^{\wedge} n) \text{ mx}; \text{mx23} = \text{drop } (2^{\wedge} n) \text{ mx}$
 $\text{in } (\text{map } (\text{take } (2^{\wedge} n)) \text{ mx01}, \text{map } (\text{drop } (2^{\wedge} n)) \text{ mx01}, \text{map } (\text{take } (2^{\wedge} n)) \text{ mx23},$
 $\text{map } (\text{drop } (2^{\wedge} n)) \text{ mx23}))$

lemma *decomp-sq-mx*: $sq\text{-}mx\ (Suc\ n)\ mx \implies (mx0, mx1, mx2, mx3) = decomp\ n\ mx \implies$
 $sq\text{-}mx\ n\ mx0 \wedge sq\text{-}mx\ n\ mx1 \wedge sq\text{-}mx\ n\ mx2 \wedge sq\text{-}mx\ n\ mx3$
 $\langle proof \rangle$

Quadtree of matrix:

fun *qt-of* :: $nat \Rightarrow 'a\ mx \Rightarrow 'a\ qtree$ **where**
 $qt\text{-}of\ (Suc\ n)\ mx =$
 $(let\ (mx0, mx1, mx2, mx3) = decomp\ n\ mx$
 $in\ Qc\ (qt\text{-}of\ n\ mx0)\ (qt\text{-}of\ n\ mx1)\ (qt\text{-}of\ n\ mx2)\ (qt\text{-}of\ n\ mx3)) \mid$
 $qt\text{-}of\ 0\ [[x]] = L\ x$

lemma *height-qt-of*: $sq\text{-}mx\ n\ mx \implies height(qt\text{-}of\ n\ mx) \leq n$
 $\langle proof \rangle$

lemma *compressed-qt-of*: $sq\text{-}mx\ n\ mx \implies compressed(qt\text{-}of\ n\ mx)$
 $\langle proof \rangle$

lemma *points-qt-of*: $sq\text{-}mx\ n\ mx \implies points\ n\ (qt\text{-}of\ n\ mx) = \{(i, j) \in sq\ n.\ mx\ !\ i\ !\ j\}$
 $\langle proof \rangle$

lemma *get-qt-of*: $\llbracket sq\text{-}mx\ n\ mx; (i, j) \in sq\ n \rrbracket \implies get\ n\ (qt\text{-}of\ n\ mx)\ i\ j = mx\ !\ i\ !\ j$
 $\langle proof \rangle$

3.7 From Quadtree to Matrix

definition *Qmx* :: $'a\ mx \Rightarrow 'a\ mx \Rightarrow 'a\ mx \Rightarrow 'a\ mx \Rightarrow 'a\ mx$ **where**
 $Qmx\ mx0\ mx1\ mx2\ mx3 = map2\ (@)\ mx0\ mx1\ @\ map2\ (@)\ mx2\ mx3$

fun *mx-of* :: $nat \Rightarrow 'a\ qtree \Rightarrow 'a\ mx$ **where**
 $mx\text{-}of\ n\ (L\ x) = replicate\ (2^{\wedge}n)\ (replicate\ (2^{\wedge}n)\ x) \mid$
 $mx\text{-}of\ (Suc\ n)\ (Q\ t0\ t1\ t2\ t3) =$
 $Qmx\ (mx\text{-}of\ n\ t0)\ (mx\text{-}of\ n\ t1)\ (mx\text{-}of\ n\ t2)\ (mx\text{-}of\ n\ t3)$

lemma *nth-Qmx-select*: $\llbracket sq\text{-}mx\ n\ mx0; sq\text{-}mx\ n\ mx1; sq\text{-}mx\ n\ mx2; sq\text{-}mx\ n\ mx3;$
 $i < 2 * 2^{\wedge}n; j < 2 * 2^{\wedge}n \rrbracket \implies$
 $Qmx\ mx0\ mx1\ mx2\ mx3\ !\ i\ !\ j = select\ (i < 2^{\wedge}n)\ (j < 2^{\wedge}n)\ mx0\ mx1\ mx2\ mx3$
 $!\ (i\ mod\ 2^{\wedge}n)\ !\ (j\ mod\ 2^{\wedge}n)$
 $\langle proof \rangle$

lemma *sq-mx-mx-of*: $height\ t \leq n \implies sq\text{-}mx\ n\ (mx\text{-}of\ n\ t)$
 $\langle proof \rangle$

lemma *mx-of-points*: $height\ t \leq n \implies points\ n\ t = \{(i, j) \in sq\ n.\ mx\text{-}of\ n\ t\ !\ i\ !\ j\}$
 $\langle proof \rangle$

lemma *mx-of-get*: $\llbracket \text{height } t \leq n; (i,j) \in \text{sq } n \rrbracket \implies \text{mx-of } n \ t \ ! \ i \ ! \ j = \text{get } n \ t \ i \ j$
<proof>

end

4 Block Matrices via Quad Trees

theory *Quad-Matrix*

imports

Complex-Main

Quad-Base

begin

There are two possible representations of matrices as quadtrees. In this file we use the standard quadtree with two constructors L and Q . $L \ x$ represents the x -diagonal matrix of arbitrary dimension. In particular $L \ 0$ is the "empty" case. Because $L \ x$ can be of arbitrary dimension, it can be added and multiplied with Q .

In the second representation (not covered in this theory) $L \ x$ is the 1×1 matrix x . The advantage is that there are fewer cases in function definitions because one cannot add/multiply L and Q : they have different dimensions. However, $L \ 0$ is special: it still represents the 0 matrix of arbitrary dimension. This leads to a more complicated invariant wrt dimension. Or one introduces a new constructor, eg *Empty*.

4.1 Square Matrices

type-synonym $ma = nat \Rightarrow nat \Rightarrow real$

Implicitly entries outside the dimensions of the matrix are 0. This is maintained by addition; multiplication and diagonal need an explicit argument n to maintain it.

definition $mk\text{-}sq :: nat \Rightarrow ma \Rightarrow ma$ **where**

$mk\text{-}sq \ n \ a = (\lambda i \ j. \text{if } i < 2^n \wedge j < 2^n \text{ then } a \ i \ j \text{ else } 0)$

abbreviation $sq\text{-}ma \ n \ (a :: ma) \equiv (\forall i \ j. 2^n \leq i \vee 2^n \leq j \longrightarrow a \ i \ j = 0)$

Without *mk-sq* a number of lemmas like *mult-ma-diag-ma-diag-ma* don't hold.

definition $diag\text{-}ma :: nat \Rightarrow real \Rightarrow ma$ **where**

$diag\text{-}ma \ n \ x = mk\text{-}sq \ n \ (\lambda i \ j. \text{if } i=j \text{ then } x \text{ else } 0)$

definition $add\text{-}ma :: ma \Rightarrow ma \Rightarrow ma$ **where**

$add\text{-}ma \ a \ b = (\lambda i \ j. a \ i \ j + b \ i \ j)$

definition $mult\text{-}ma :: nat \Rightarrow ma \Rightarrow ma \Rightarrow ma$ **where**
 $mult\text{-}ma\ n\ a\ b = (\lambda i\ j. \sum_{k=0..<2^{\widehat{n}}} a\ i\ k * b\ k\ j)$

4.2 Matrix Lemmas

lemma $add\text{-}ma\text{-}diag\text{-}ma[simp]$: $add\text{-}ma\ (diag\text{-}ma\ n\ x)\ (diag\text{-}ma\ n\ y) = diag\text{-}ma\ n\ (x+y)$
 $\langle proof \rangle$

lemma $add\text{-}ma\text{-}diag\text{-}ma\text{-}0[simp]$: $add\text{-}ma\ (diag\text{-}ma\ n\ 0)\ a = a$
 $\langle proof \rangle$

lemma $add\text{-}ma\text{-}diag\text{-}ma\text{-}02[simp]$: $add\text{-}ma\ a\ (diag\text{-}ma\ n\ 0) = a$
 $\langle proof \rangle$

lemma $mult\text{-}ma\text{-}diag\text{-}ma\text{-}0[simp]$: $mult\text{-}ma\ n\ (diag\text{-}ma\ n\ 0)\ a = diag\text{-}ma\ n\ 0$
 $\langle proof \rangle$

lemma $mult\text{-}ma\text{-}diag\text{-}ma\text{-}02[simp]$: $mult\text{-}ma\ n\ a\ (diag\text{-}ma\ n\ 0) = diag\text{-}ma\ n\ 0$
 $\langle proof \rangle$

lemma $mult\text{-}ma\text{-}diag\text{-}ma\text{-}diag\text{-}ma[simp]$: $mult\text{-}ma\ n\ (diag\text{-}ma\ n\ x)\ (diag\text{-}ma\ n\ y) = diag\text{-}ma\ n\ (x*y)$
 $\langle proof \rangle$

4.3 Real Quad Trees and Abstraction to Matrices

type-synonym $qtr = real\ qtree$

fun $compressed :: qtr \Rightarrow bool$ **where**
 $compressed\ (L\ x) = True \mid$
 $compressed\ (Q\ (L\ x0)\ (L\ x1)\ (L\ x2)\ (L\ x3)) = (\neg (x1=0 \wedge x2=0 \wedge x0=x3)) \mid$
 $compressed\ (Q\ t0\ t1\ t2\ t3) = (compressed\ t0 \wedge compressed\ t1 \wedge compressed\ t2 \wedge compressed\ t3)$

lemma $compressed\text{-}Q$:
 $compressed\ (Q\ t0\ t1\ t2\ t3) \implies (compressed\ t0 \wedge compressed\ t1 \wedge compressed\ t2 \wedge compressed\ t3)$
 $\langle proof \rangle$

definition $Qma :: nat \Rightarrow ma \Rightarrow ma \Rightarrow ma \Rightarrow ma \Rightarrow ma$ **where**
 $Qma\ n\ a\ b\ c\ d =$
 $(\lambda i\ j. \text{if } i < 2^{\widehat{n}} \text{ then if } j < 2^{\widehat{n}} \text{ then } a\ i\ j \text{ else } b\ i\ (j - 2^{\widehat{n}}) \text{ else}$
 $\text{if } j < 2^{\widehat{n}} \text{ then } c\ (i - 2^{\widehat{n}})\ j \text{ else } d\ (i - 2^{\widehat{n}})\ (j - 2^{\widehat{n}}))$

lemma $add\text{-}ma\text{-}Qma$:
 $add\text{-}ma\ (Qma\ n\ a\ b\ c\ d)\ (Qma\ n\ a'\ b'\ c'\ d') =$
 $Qma\ n\ (add\text{-}ma\ a\ a')\ (add\text{-}ma\ b\ b')\ (add\text{-}ma\ c\ c')\ (add\text{-}ma\ d\ d')$
 $\langle proof \rangle$

lemma *add-ma-diag-ma-Qma*: $add\text{-}ma\ (diag\text{-}ma\ (Suc\ n)\ x)\ (Qma\ n\ a\ b\ c\ d) =$
 $Qma\ n\ (add\text{-}ma\ (diag\text{-}ma\ n\ x)\ a)\ b\ c\ (add\text{-}ma\ (diag\text{-}ma\ n\ x)\ d)$
 ⟨proof⟩

lemma *add-ma-Qma-diag-ma*: $add\text{-}ma\ (Qma\ n\ a\ b\ c\ d)\ (diag\text{-}ma\ (Suc\ n)\ x) =$
 $Qma\ n\ (add\text{-}ma\ a\ (diag\text{-}ma\ n\ x))\ b\ c\ (add\text{-}ma\ d\ (diag\text{-}ma\ n\ x))$
 ⟨proof⟩

lemma *diag-ma-Suc*: $diag\text{-}ma\ (Suc\ n)\ x = Qma\ n\ (diag\text{-}ma\ n\ x)\ (diag\text{-}ma\ n\ 0)$
 $(diag\text{-}ma\ n\ 0)\ (diag\text{-}ma\ n\ x)$
 ⟨proof⟩

Abstraction function:

fun *ma* :: $nat \Rightarrow qtr \Rightarrow ma$ **where**
 $ma\ n\ (L\ x) = diag\text{-}ma\ n\ x \mid$
 $ma\ (Suc\ n)\ (Q\ t0\ t1\ t2\ t3) =$
 $Qma\ n\ (ma\ n\ t0)\ (ma\ n\ t1)\ (ma\ n\ t2)\ (ma\ n\ t3)$

4.4 Matrix Operations on Trees

fun *Qc* :: $qtr \Rightarrow qtr \Rightarrow qtr \Rightarrow qtr \Rightarrow qtr$ **where**
 $Qc\ (L\ x0)\ (L\ x1)\ (L\ x2)\ (L\ x3) =$
 $(if\ x1=0 \wedge x2=0 \wedge x0=x3\ then\ L\ x0\ else\ Q\ (L\ x0)\ (L\ x1)\ (L\ x2)\ (L\ x3)) \mid$
 $Qc\ t0\ t1\ t2\ t3 = Q\ t0\ t1\ t2\ t3$

lemma *ma-Suc-Qc*: $ma\ (Suc\ n)\ (Qc\ t0\ t1\ t2\ t3) = ma\ (Suc\ n)\ (Q\ t0\ t1\ t2\ t3)$
 ⟨proof⟩

lemma *compressed-Qc*:
 $compressed\ (Qc\ t0\ t1\ t2\ t3) = (compressed\ t0 \wedge compressed\ t1 \wedge compressed\ t2$
 $\wedge\ compressed\ t3)$
 ⟨proof⟩

lemma *height-Qc-Q*:
 $height\ (Qc\ t0\ t1\ t2\ t3) \leq height\ (Q\ t0\ t1\ t2\ t3)$
 ⟨proof⟩

fun *add* :: $qtr \Rightarrow qtr \Rightarrow qtr$ **where**
 $add\ (Q\ s0\ s1\ s2\ s3)\ (Q\ t0\ t1\ t2\ t3) = Qc\ (add\ s0\ t0)\ (add\ s1\ t1)\ (add\ s2\ t2)$
 $(add\ s3\ t3) \mid$
 $add\ (L\ x)\ (L\ y) = L(x+y) \mid$
 $add\ (L\ x)\ (Q\ t0\ t1\ t2\ t3) = Qc\ (add\ (L\ x)\ t0)\ t1\ t2\ (add\ (L\ x)\ t3) \mid$
 $add\ (Q\ t0\ t1\ t2\ t3)\ (L\ x) = Qc\ (add\ t0\ (L\ x))\ t1\ t2\ (add\ t3\ (L\ x))$

fun *mult* :: $qtr \Rightarrow qtr \Rightarrow qtr$ **where**
 $mult\ (Q\ s0\ s1\ s2\ s3)\ (Q\ t0\ t1\ t2\ t3) =$
 $Qc\ (add\ (mult\ s0\ t0)\ (mult\ s1\ t2))$
 $(add\ (mult\ s0\ t1)\ (mult\ s1\ t3))$
 $(add\ (mult\ s2\ t0)\ (mult\ s3\ t2))$
 $(add\ (mult\ s2\ t1)\ (mult\ s3\ t3)) \mid$

```

mult (L x) (Q t0 t1 t2 t3) =
  Qc (mult (L x) t0)
    (mult (L x) t1)
    (mult (L x) t2)
    (mult (L x) t3) |
mult (Q t0 t1 t2 t3) (L x) =
  Qc (mult t0 (L x))
    (mult t1 (L x))
    (mult t2 (L x))
    (mult t3 (L x)) |
mult (L x) (L y) = L(x*y)

```

Initialization of *qtr* from *ma*

```

fun qtr :: nat => ma => qtr where
qtr 0 a = L(a 0 0) |
qtr (Suc n) a =
  (let t0 = qtr n a; t1 = qtr n (λi j. a i (j+2^n));
    t2 = qtr n (λi j. a (i+2^n) j); t3 = qtr n (λi j. a (i+2^n) (j+2^n))
  in Q t0 t1 t2 t3)

```

4.5 Correctness of Quad Tree Implementations

4.5.1 add

lemma *ma-add*: $\llbracket \text{height } s \leq n; \text{height } t \leq n \rrbracket \implies$
 $\text{ma } n \text{ (add } s \text{ t)} = \text{add-ma (ma } n \text{ s) (ma } n \text{ t)}$
<proof>

lemma *height-add*: $\text{height (add } s \text{ t)} \leq \max (\text{height } s) (\text{height } t)$
<proof>

lemma *compressed-add*: $\llbracket \text{compressed } s; \text{compressed } t \rrbracket \implies \text{compressed (add } s \text{ t)}$
<proof>

lemma *Max4*: $\text{Max}\{n0, n1, n2, n3\} = \max n0 (\max n1 (\max n2 n3))$ *<proof>*

lemma *height-mult*: $\text{height (mult } s \text{ t)} \leq \max (\text{height } s) (\text{height } t)$
<proof>

4.5.2 mult

lemma *bij-betw-minus-ivlco-nat*: $n \leq a \implies C = \{a-n..<b-n\} \implies \text{bij-betw } (\lambda k::\text{nat. } k-n) \{a..<b\} C$
<proof>

lemma *mult-ma-Qma-Qma*:
 $\text{mult-ma (Suc } n) (Qma \text{ } n \text{ } a \text{ } b \text{ } c \text{ } d) (Qma \text{ } n \text{ } a' \text{ } b' \text{ } c' \text{ } d') =$
 $(Qma \text{ } n (\text{add-ma (mult-ma } n \text{ } a \text{ } a') (\text{mult-ma } n \text{ } b \text{ } c'))$
 $(\text{add-ma (mult-ma } n \text{ } a \text{ } b') (\text{mult-ma } n \text{ } b \text{ } d'))$
 $(\text{add-ma (mult-ma } n \text{ } c \text{ } a') (\text{mult-ma } n \text{ } d \text{ } c'))$

```

      (add-ma (mult-ma n c b') (mult-ma n d d'))
⟨proof⟩

lemma ma-mult: [ height s ≤ n; height t ≤ n ] ⇒
  ma n (mult s t) = mult-ma n (ma n s) (ma n t)
⟨proof⟩

lemma compressed-mult: [ compressed s; compressed t ] ⇒ compressed (mult s
t)
⟨proof⟩

end

```

5 K-dimensional Region Trees

```

theory KD-Region-Tree
imports
  HOL-Library.NList
  HOL-Library.Tree
begin

```

Generalizes quadtrees. Instead of having 2^n direct children of a node, the children are arranged in a binary tree where each *Split* splits along one dimension.

```

datatype 'a kdt = Box 'a | Split 'a kdt 'a kdt

```

```

datatype-compat kdt

```

```

type-synonym kdtb = bool kdt

```

A *kdt* is most easily explained by showing how quad trees are represented: $Q\ t0\ t1\ t2\ t3$ becomes $Split\ (Split\ t0'\ t1')\ (Split\ t2'\ t3')$ where ti' is the representation of ti ; $L\ a$ becomes $Box\ a$. In general, each level of an abstract k dimensional tree subdivides space into 2^k subregions. This subdivision is represented by a *kdt* of depth at most k . Further subdivisions of the subregions are seamlessly represented as the subtrees at depth k . $Box\ a$ represents a subregion entirely filled with a 's. In contrast to quad trees, cubes can also occur half way down the subdivision. For example, $Q\ (L\ a)\ (L\ b)\ (L\ c)$ becomes $Split\ (Box\ a)\ (Split\ (Box\ b)\ (Box\ c))$.

```

instantiation kdt :: (type)height
begin

```

```

fun height-kdt :: 'a kdt ⇒ nat where
  height (Box -) = 0 |

```

$height (Split\ l\ r) = max (height\ l) (height\ r) + 1$

instance $\langle proof \rangle$

end

lemma *height-0-iff*: $height\ t = 0 \longleftrightarrow (\exists x. t = Box\ x)$
 $\langle proof \rangle$

definition *bits* :: $nat \Rightarrow bool\ list\ set$
where $bits\ n = nlists\ n\ UNIV$

lemma *bits-code* [*code*]:
 $bits\ n = nlists\ n\ \{False, True\}$
 $\langle proof \rangle$

5.1 Subtree

fun *subtree* :: $'a\ kdt \Rightarrow bool\ list \Rightarrow 'a\ kdt$ **where**
 $subtree\ t\ [] = t$ |
 $subtree\ (Box\ x)\ _ = Box\ x$ |
 $subtree\ (Split\ l\ r)\ (b\#\ bs) = subtree\ (if\ b\ then\ r\ else\ l)\ bs$

lemma *subtree-Box*[*simp*]: $subtree\ (Box\ x)\ bs = Box\ x$
 $\langle proof \rangle$

lemma *height-subtree*: $height\ (subtree\ t\ bs) \leq height\ t - length\ bs$
 $\langle proof \rangle$

lemma *height-subtree2*: $\llbracket height\ t \leq k * (Suc\ n); length\ bs = k \rrbracket \Longrightarrow height\ (subtree\ t\ bs) \leq k * n$
 $\langle proof \rangle$

lemma *subtree-Split-Box*: $length\ bs \neq 0 \Longrightarrow subtree\ (Split\ (Box\ b)\ (Box\ b))\ bs = Box\ b$
 $\langle proof \rangle$

5.2 Shifting a coordinate by a boolean vector

definition *mv* :: $nat \Rightarrow bool\ list \Rightarrow nat\ list \Rightarrow nat\ list$ **where**
 $mv\ d = map2\ (\lambda b\ x. x + (if\ b\ then\ 0\ else\ d))$

lemma *map-zip1*: $\llbracket length\ xs = length\ ys; \forall p \in set(zip\ xs\ ys). f\ p = fst\ p \rrbracket \Longrightarrow map\ f\ (zip\ xs\ ys) = xs$
 $\langle proof \rangle$

lemma *map-mv1*: $\llbracket ps \in nlists\ (length\ bs)\ \{0..<n\}; length\ ps = length\ bs \rrbracket \Longrightarrow map\ (\lambda i. i < n)\ (mv\ (n)\ bs\ ps) = bs$
 $\langle proof \rangle$

lemma *map-zip2*: $\llbracket \text{length } xs = \text{length } ys; \forall p \in \text{set}(\text{zip } xs \text{ } ys). f p = \text{snd } p \rrbracket \implies$
 $\text{map } f (\text{zip } xs \text{ } ys) = ys$
 $\langle \text{proof} \rangle$

lemma *map-mv2*: $\llbracket ps \in \text{nlists } (\text{length } bs) \{0..<2^n\} \rrbracket \implies \text{map } (\lambda x. x \bmod 2^n)$
 $(\text{mv } (2^n) \text{ } bs \text{ } ps) = ps$
 $\langle \text{proof} \rangle$

lemma *mv-map-map*: $\text{set } ps \subseteq \{0..<2 * n\} \implies \text{mv } (n) (\text{map } (\lambda x. x < n) \text{ } ps)$
 $(\text{map } (\lambda x. x \bmod n) \text{ } ps) = ps$
 $\langle \text{proof} \rangle$

lemma *mv-in-nlists*:
 $\llbracket p \in \text{nlists } k \{0..<2^n\}; bs \in \text{bits } k \rrbracket \implies \text{mv } (2^n) \text{ } bs \text{ } p \in \text{nlists } k \{0..<2 *$
 $2^n\}$
 $\langle \text{proof} \rangle$

lemma *in-nlists2D*: $xs \in \text{nlists } k \{0..<2 * 2^n\} \implies \exists bs \in \text{bits } k. xs \in \text{mv } (2^n)$
 $bs \text{ } \text{nlists } k \{0..<2^n\}$
 $\langle \text{proof} \rangle$

lemma *nlists2-simp*: $\text{nlists } k \{0..<2 * 2^n\} = (\bigcup bs \in \text{bits } k. \text{mv } (2^n) \text{ } bs \text{ } \text{nlists}$
 $k \{0..<2^n\})$
 $\langle \text{proof} \rangle$

lemma *in-mv-image*: $\llbracket ps \in \text{nlists } k \{0..<2*2^n\}; Ps \subseteq \text{nlists } k \{0..<2^n\}; bs \in$
 $\text{bits } k \rrbracket \implies$
 $ps \in \text{mv } (2^n) \text{ } bs \text{ } Ps \iff \text{map } (\lambda x. x \bmod 2^n) \text{ } ps \in Ps \wedge (bs = \text{map } (\lambda i. i <$
 $2^n) \text{ } ps)$
 $\langle \text{proof} \rangle$

5.3 Points in a tree

fun *cube* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat list set}$ **where**
 $\text{cube } k \text{ } n = \text{nlists } k \{0..<2^n\}$

fun *points* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{kdtb} \Rightarrow \text{nat list set}$ **where**
 $\text{points } k \text{ } n (\text{Box } b) = (\text{if } b \text{ then } \text{cube } k \text{ } n \text{ else } \{\}) \mid$
 $\text{points } k (\text{Suc } n) \text{ } t = (\bigcup bs \in \text{bits } k. \text{mv } (2^n) \text{ } bs \text{ } \text{points } k \text{ } n (\text{subtree } t \text{ } bs))$

lemma *points-Suc*: $\text{points } k (\text{Suc } n) \text{ } t = (\bigcup bs \in \text{bits } k. \text{mv } (2^n) \text{ } bs \text{ } \text{points } k \text{ } n$
 $(\text{subtree } t \text{ } bs))$
 $\langle \text{proof} \rangle$

lemma *points-subset*: $\text{height } t \leq k * n \implies \text{points } k \text{ } n \text{ } t \subseteq \text{nlists } k \{0..<2^n\}$
 $\langle \text{proof} \rangle$

5.4 Compression

Compressing Split:

fun *SplitC* :: 'a kdt ⇒ 'a kdt ⇒ 'a kdt **where**
SplitC (Box b1) (Box b2) = (if b1=b2 then Box b1 else Split (Box b1) (Box b2)) |
SplitC l r = Split l r

fun *compressed* :: 'a kdt ⇒ bool **where**
compressed (Box -) = True |
compressed (Split l r) = (compressed l ∧ compressed r ∧ ¬(∃ b. l = Box b ∧ r = Box b))

lemma *compressedI*: [*compressed* l; *compressed* r] ⇒ *compressed* (*SplitC* l r)
 ⟨*proof*⟩

lemma *subtree-SplitC*:
 1 ≤ length bs ⇒ subtree (*SplitC* l r) bs = subtree (Split l r) bs
 ⟨*proof*⟩

lemma *height-SplitC*: height(*SplitC* l r) ≤ Suc (max (height l) (height r))
 ⟨*proof*⟩

lemma *height-SplitC2*: [height l ≤ n; height r ≤ n] ⇒ height(*SplitC* l r) ≤ Suc n
 ⟨*proof*⟩

5.5 Extracting a point from a tree

Also the abstraction function.

fun *get* :: nat ⇒ 'a kdt ⇒ nat list ⇒ 'a **where**
get - (Box b) - = b |
get (Suc n) t ps = get n (subtree t (map (λi. i < 2ⁿ) ps)) (map (λi. i mod 2ⁿ) ps)

lemma *get-Suc*: *get* (Suc n) t ps =
 get n (subtree t (map (λi. i < 2ⁿ) ps)) (map (λi. i mod 2ⁿ) ps)
 ⟨*proof*⟩

lemma *points-get*: [height t ≤ k*n; ps ∈ nlists k {0..<2ⁿ}] ⇒
 get n t ps = (ps ∈ points k n t)
 ⟨*proof*⟩

5.6 Modifying a point in a tree

fun *modify* :: ('a kdt ⇒ 'a kdt) ⇒ bool list ⇒ 'a kdt ⇒ 'a kdt **where**
modify f [] t = f t |
modify f (b # bs) (Split l r) = (if b then SplitC l (modify f bs r) else SplitC (modify f bs l) r) |
modify f (b # bs) (Box a) =
 (let t = modify f bs (Box a) in if b then SplitC (Box a) t else SplitC t (Box a))

fun *put* :: nat list ⇒ 'a ⇒ nat ⇒ 'a kdt ⇒ 'a kdt **where**

$put\ ps\ a\ 0\ (Box\ -) = Box\ a\ |$
 $put\ ps\ a\ (Suc\ n)\ t = modify\ (put\ (map\ (\lambda i. i\ mod\ 2^{\wedge}n)\ ps)\ a\ n)\ (map\ (\lambda i. i < 2^{\wedge}n)\ ps)\ t$

lemma *height-modify*: $\llbracket \forall t. height\ t \leq nk \longrightarrow height\ (f\ t) \leq nk;$
 $height\ t \leq k + nk; length\ bs = k \rrbracket$
 $\implies height\ (modify\ f\ bs\ t) \leq k + nk$
 $\langle proof \rangle$

lemma *height-put*: $height\ t \leq n * length\ ps \implies height\ (put\ ps\ a\ n\ t) \leq n * length\ ps$
 $\langle proof \rangle$

lemma *subtree-modify*: $\llbracket length\ bs' = length\ bs \rrbracket$
 $\implies subtree\ (modify\ f\ bs\ t)\ bs' = (if\ bs' = bs\ then\ f\ (subtree\ t\ bs)\ else\ subtree\ t\ bs')$
 $\langle proof \rangle$

lemma *mod-eq1*: $\llbracket y < 2 * n; ya < 2 * n; \neg ya < n; \neg y < n; ya\ mod\ n = y\ mod\ n \rrbracket$
 $\implies ya = (y::nat)$
 $\langle proof \rangle$

lemma *nlist-eq-mod*: $\llbracket ps \in nlists\ k\ \{0..<(2::nat) * 2^{\wedge}n\}; ps' \in nlists\ k\ \{0..<2 * 2^{\wedge}n\};$
 $map\ (\lambda i. i < 2^{\wedge}n)\ ps' = map\ (\lambda i. i < 2^{\wedge}n)\ ps; ps' \neq ps \rrbracket \implies$
 $map\ (\lambda i. i\ mod\ 2^{\wedge}n)\ ps' \neq map\ (\lambda i. i\ mod\ 2^{\wedge}n)\ ps$
 $\langle proof \rangle$

lemma *get-put*: $\llbracket height\ t \leq k*n; ps \in cube\ k\ n; ps' \in cube\ k\ n \rrbracket \implies$
 $get\ n\ (put\ ps\ a\ n\ t)\ ps' = (if\ ps' = ps\ then\ a\ else\ get\ n\ t\ ps')$
 $\langle proof \rangle$

lemma *compressed-modify*: $\llbracket compressed\ t; compressed\ (f\ (subtree\ t\ bs)) \rrbracket \implies$
 $compressed\ (modify\ f\ bs\ t)$
 $\langle proof \rangle$

lemma *compressed-subtree*: $compressed\ t \implies compressed\ (subtree\ t\ bs)$
 $\langle proof \rangle$

lemma *compressed-put*:
 $\llbracket height\ t \leq k*n; k = length\ ps; compressed\ t \rrbracket \implies compressed\ (put\ ps\ a\ n\ t)$
 $\langle proof \rangle$

5.7 Union

fun *union* :: $kdtb \Rightarrow kdtb \Rightarrow kdtb$ **where**
 $union\ (Box\ b)\ t = (if\ b\ then\ Box\ True\ else\ t)\ |$

$union\ t\ (Box\ b) = (if\ b\ then\ Box\ True\ else\ t) \mid$
 $union\ (Split\ l1\ r1)\ (Split\ l2\ r2) = SplitC\ (union\ l1\ l2)\ (union\ r1\ r2)$

lemma *union-Box2*: $union\ t\ (Box\ b) = (if\ b\ then\ Box\ True\ else\ t)$
 $\langle proof \rangle$

lemma *subtree-union*: $subtree\ (union\ t1\ t2)\ bs = union\ (subtree\ t1\ bs)\ (subtree\ t2\ bs)$
 $\langle proof \rangle$

lemma *points-union*:
 $\llbracket\ max\ (height\ t1)\ (height\ t2) \leq k * n \rrbracket \implies$
 $points\ k\ n\ (union\ t1\ t2) = points\ k\ n\ t1 \cup points\ k\ n\ t2$
 $\langle proof \rangle$

lemma *get-union*:
 $\llbracket\ max\ (height\ t1)\ (height\ t2) \leq length\ ps * n \rrbracket \implies$
 $get\ n\ (union\ t1\ t2)\ ps = (get\ n\ t1\ ps \vee get\ n\ t2\ ps)$
 $\langle proof \rangle$

lemma *height-union*: $height\ (union\ t1\ t2) \leq max\ (height\ t1)\ (height\ t2)$
 $\langle proof \rangle$

lemma *compressed-union*: $compressed\ t1 \implies compressed\ t2 \implies compressed\ (union\ t1\ t2)$
 $\langle proof \rangle$

end

6 K-dimensional Region Trees - Version 2

theory *KD-Region-Tree2*

imports

HOL-Library.NList

HOL-Library.Tree

begin

datatype $'a\ kdt = Box\ 'a \mid Split\ 'a\ kdt\ 'a\ kdt$

datatype-compat kdt

type-synonym $kdtb = bool\ kdt$

A kdt is most easily explained by showing how quad trees are represented: $Q\ t0\ t1\ t2\ t3$ becomes $Split\ (Split\ t0'\ t1')\ (Split\ t2'\ t3')$ where ti' is the representation of ti ; $L\ a$ becomes $Box\ a$. In general, each level of an abstract

k dimensional tree subdivides space into 2^k subregions. This subdivision is represented by a *kdt* of depth at most k . Further subdivisions of the subregions are seamlessly represented as the subtrees at depth k . *Box a* represents a subregion entirely filled with a 's. In contrast to quad trees, cubes can also occur half way down the subdivision. For example, $Q (L a) (L a) (L b) (L c)$ becomes $Split (Box a) (Split (Box b) (Box c))$.

instantiation *kdt* :: (type)height
begin

fun *height-kdt* :: 'a *kdt* \Rightarrow nat **where**
height (*Box* -) = 0 |
height (*Split* l r) = max (*height* l) (*height* r) + 1

instance <proof>

end

lemma *height-0-iff*: *height* t = 0 \longleftrightarrow (\exists x. t = *Box* x)
 <proof>

definition *bits* :: nat \Rightarrow bool list set **where**
bits n \equiv *nlists* n UNIV

lemma *bits-Suc*[code]:
bits (*Suc* n) = (let B = *bits* n in (#) True ' B \cup (#) False ' B)
 <proof>

6.1 Subtree

fun *subtree* :: 'a *kdt* \Rightarrow bool list \Rightarrow 'a *kdt* **where**
subtree t [] = t |
subtree (*Box* x) - = *Box* x |
subtree (*Split* l r) (b#bs) = *subtree* (if b then r else l) bs

lemma *subtree-Box*[simp]: *subtree* (*Box* x) bs = *Box* x
 <proof>

lemma *height-subtree*: *height* (*subtree* t bs) \leq *height* t - *length* bs
 <proof>

lemma *height-subtree2*: \llbracket *height* t \leq k * (*Suc* n); *length* bs = k $\rrbracket \Longrightarrow$ *height* (*subtree* t bs) \leq k * n
 <proof>

lemma *subtree-Split-Box*: *length* bs \neq 0 \Longrightarrow *subtree* (*Split* (*Box* b) (*Box* b)) bs = *Box* b
 <proof>

6.2 Shifting a coordinate by a boolean vector

The ?

definition $mv :: \text{bool list} \Rightarrow \text{nat list} \Rightarrow \text{nat list}$ **where**
 $mv = \text{map2 } (\lambda b x. 2*x + (\text{if } b \text{ then } 0 \text{ else } 1))$

lemma $\text{map-}zip1$: $\llbracket \text{length } xs = \text{length } ys; \forall p \in \text{set}(\text{zip } xs \text{ } ys). f p = \text{fst } p \rrbracket \Longrightarrow$
 $\text{map } f (\text{zip } xs \text{ } ys) = xs$
 $\langle \text{proof} \rangle$

lemma $\text{map-}mv1$: $\llbracket \text{length } ps = \text{length } bs \rrbracket \Longrightarrow \text{map even } (mv \text{ } bs \text{ } ps) = bs$
 $\langle \text{proof} \rangle$

lemma $\text{map-}zip2$: $\llbracket \text{length } xs = \text{length } ys; \forall p \in \text{set}(\text{zip } xs \text{ } ys). f p = \text{snd } p \rrbracket \Longrightarrow$
 $\text{map } f (\text{zip } xs \text{ } ys) = ys$
 $\langle \text{proof} \rangle$

lemma $\text{map-}mv2$: $\llbracket \text{length } ps = \text{length } bs \rrbracket \Longrightarrow \text{map } (\lambda x. x \text{ div } 2) (mv \text{ } bs \text{ } ps) =$
 ps
 $\langle \text{proof} \rangle$

lemma $mv\text{-map-}map$: $mv (\text{map even } ps) (\text{map } (\lambda x. x \text{ div } 2) ps) = ps$
 $\langle \text{proof} \rangle$

lemma $mv\text{-in-}nlists$:
 $\llbracket p \in nlists \text{ } k \{0..<2 \wedge n\}; bs \in bits \text{ } k \rrbracket \Longrightarrow mv \text{ } bs \text{ } p \in nlists \text{ } k \{0..<2 * 2 \wedge n\}$
 $\langle \text{proof} \rangle$

lemma $in\text{-}nlists2D$: $xs \in nlists \text{ } k \{0..<2 * 2 \wedge n\} \Longrightarrow \exists bs \in bits \text{ } k. xs \in mv \text{ } bs \text{ '}$
 $nlists \text{ } k \{0..<2 \wedge n\}$
 $\langle \text{proof} \rangle$

lemma $nlists2\text{-}simp$: $nlists \text{ } k \{0..<2 * 2 \wedge n\} = (\bigcup bs \in bits \text{ } k. mv \text{ } bs \text{ ' } nlists \text{ } k$
 $\{0..<2 \wedge n\})$
 $\langle \text{proof} \rangle$

6.3 Points in a tree

fun $cube :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat list set}$ **where**
 $cube \text{ } k \text{ } n = nlists \text{ } k \{0..<2 \wedge n\}$

fun $points :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{kdtb} \Rightarrow \text{nat list set}$ **where**
 $points \text{ } k \text{ } n (\text{Box } b) = (\text{if } b \text{ then } cube \text{ } k \text{ } n \text{ else } \{\}) \mid$
 $points \text{ } k (\text{Suc } n) t = (\bigcup bs \in bits \text{ } k. mv \text{ } bs \text{ ' } points \text{ } k \text{ } n (\text{subtree } t \text{ } bs))$

lemma $points\text{-}Suc$: $points \text{ } k (\text{Suc } n) t = (\bigcup bs \in bits \text{ } k. mv \text{ } bs \text{ ' } points \text{ } k \text{ } n (\text{subtree}$
 $t \text{ } bs))$
 $\langle \text{proof} \rangle$

lemma *points-subset*: $height\ t \leq k*n \implies points\ k\ n\ t \subseteq nlists\ k\ \{0..<2^{\wedge}n\}$
 <proof>

6.4 Compression

Compressing Split:

fun *SplitC* :: 'a kdt \Rightarrow 'a kdt \Rightarrow 'a kdt **where**
SplitC (Box b1) (Box b2) = (if b1=b2 then Box b1 else Split (Box b1) (Box b2)) |
SplitC t1 t2 = Split t1 t2

fun *compressed* :: 'a kdt \Rightarrow bool **where**
compressed (Box -) = True |
compressed (Split l r) = (compressed l \wedge compressed r \wedge $\neg(\exists b. l = Box\ b \wedge r = Box\ b)$)

lemma *compressedI*: $\llbracket compressed\ t1; compressed\ t2 \rrbracket \implies compressed\ (SplitC\ t1\ t2)$
 <proof>

lemma *subtree-SplitC*:
 $1 \leq length\ bs \implies subtree\ (SplitC\ l\ r)\ bs = subtree\ (Split\ l\ r)\ bs$
 <proof>

6.5 Union

fun *union* :: kdtb \Rightarrow kdtb \Rightarrow kdtb **where**
union (Box b) t = (if b then Box True else t) |
union t (Box b) = (if b then Box True else t) |
union (Split l1 r1) (Split l2 r2) = SplitC (union l1 l2) (union r1 r2)

lemma *union-Box2*: $union\ t\ (Box\ b) = (if\ b\ then\ Box\ True\ else\ t)$
 <proof>

lemma *in-mv-image*: $\llbracket ps \in nlists\ k\ \{0..<2*2^{\wedge}n\}; Ps \subseteq nlists\ k\ \{0..<2^{\wedge}n\}; bs \in bits\ k \rrbracket \implies$
 $ps \in mv\ bs\ 'Ps \iff map\ (\lambda x. x\ div\ 2)\ ps \in Ps \wedge (bs = map\ even\ ps)$
 <proof>

lemma *subtree-union*: $subtree\ (union\ t1\ t2)\ bs = union\ (subtree\ t1\ bs)\ (subtree\ t2\ bs)$
 <proof>

lemma *points-union*:
 $\llbracket max\ (height\ t1)\ (height\ t2) \leq k*n \rrbracket \implies$
 $points\ k\ n\ (union\ t1\ t2) = points\ k\ n\ t1 \cup points\ k\ n\ t2$
 <proof>

lemma *compressed-union*: $compressed\ t1 \implies compressed\ t2 \implies compressed(union\ t1\ t2)$
 ⟨*proof*⟩

6.6 Extracting a point from a tree

lemma *size-subtree*: $bs \neq [] \implies (\forall b. t \neq Box\ b) \implies size\ (subtree\ t\ bs) < size\ t$
 ⟨*proof*⟩

For termination of *get*:

corollary *size-subtree-Split*[*termination-simp*]:
 $bs \neq [] \implies size\ (subtree\ (Split\ l\ r)\ bs) < Suc\ (size\ l + size\ r)$
 ⟨*proof*⟩

fun *get* :: 'a kdt \Rightarrow nat list \Rightarrow 'a **where**
 $get\ (Box\ b) = b$ |
 $get\ t\ ps = (if\ ps = []\ then\ undefined\ else\ get\ (subtree\ t\ (map\ even\ ps))\ (map\ (\lambda i. i\ div\ 2)\ ps))$

lemma *points-get*: $\llbracket height\ t \leq k * n; ps \in nlists\ k\ \{0..<2^n\} \rrbracket \implies$
 $get\ t\ ps = (ps \in points\ k\ n\ t)$
 ⟨*proof*⟩

end

7 K-dimensional Region Trees - Nested Trees

theory *KD-Region-Nested*
imports *HOL-Library.NList*
begin

fun *cube* :: nat \Rightarrow nat \Rightarrow nat list set **where**
 $cube\ k\ n = nlists\ k\ \{0..<2^n\}$

datatype 'a tree1 = Lf 'a | Br 'a tree1 'a tree1
datatype 'a kdt = Cube 'a | Dims 'a kdt tree1

datatype-compat tree1
datatype-compat kdt

type-synonym kdtb = bool kdt

lemma *set-tree1-finite-ne*: $finite\ (set-tree1\ t) \wedge set-tree1\ t \neq \{\}$
 ⟨*proof*⟩

lemma *kdt-tree1-term*[*termination-simp*]: $x \in set-tree1\ t \implies size-kdt\ f\ x < Suc\ (size-tree1\ (size-kdt\ f)\ t)$
 ⟨*proof*⟩

```

fun h-tree1 :: 'a tree1 ⇒ nat where
  h-tree1 (Lf -) = 0 |
  h-tree1 (Br l r) = max (h-tree1 l) (h-tree1 r) + 1

function (sequential) h-kdt :: 'a kdt ⇒ nat where
  h-kdt (Cube -) = 0 |
  h-kdt (Dims t) = Max (h-kdt ` (set-tree1 t)) + 1
  ⟨proof⟩

termination
  ⟨proof⟩

function (sequential) inv-kdt :: nat ⇒ 'a kdt ⇒ bool where
  inv-kdt k (Cube b) = True |
  inv-kdt k (Dims t) = (h-tree1 t ≤ k ∧ (∀ kt ∈ set-tree1 t. inv-kdt k kt))
  ⟨proof⟩

termination
  ⟨proof⟩

definition bits :: ⟨nat ⇒ bool list set⟩
  where ⟨bits n = nlists n UNIV⟩

lemma bits-0-eq [simp]:
  ⟨bits 0 = {[]}⟩
  ⟨proof⟩

lemma bits-Suc-eq [simp]:
  ⟨bits (Suc n) = (#) False ` nlists n UNIV ∪ (#) True ` nlists n UNIV⟩
  ⟨proof⟩

lemma bits-code [code]:
  ⟨bits n = nlists n {False, True}⟩
  ⟨proof⟩

fun leaf :: 'a tree1 ⇒ bool list ⇒ 'a where
  leaf (Lf x) - = x |
  leaf (Br l r) (b#bs) = leaf (if b then r else l) bs |
  leaf (Br l r) [] = leaf l []

definition mv :: bool list ⇒ nat list ⇒ nat list where
  mv = map2 (λb x. 2*x + (if b then 0 else 1))

fun points :: nat ⇒ nat ⇒ kdtb ⇒ nat list set where
  points k n (Cube b) = (if b then cube k n else {}) |
  points k (Suc n) (Dims t) = (∪ bs ∈ bits k. mv bs ` points k n (leaf t bs))

lemma bits-nonempty: bits n ≠ {}
  ⟨proof⟩

```

lemma *finite-bits*: *finite* (bits *n*)
 ⟨*proof*⟩

lemma *mv-in-nlists*:
 $\llbracket p \in \text{nlists } k \{0..<2^{\wedge} n\}; bs \in \text{bits } k \rrbracket \implies mv \text{ } bs \text{ } p \in \text{nlists } k \{0..<2 * 2^{\wedge} n\}$
 ⟨*proof*⟩

lemma *leaf-append*: $length \text{ } bs \geq h\text{-tree1 } t \implies leaf \text{ } t \text{ } (bs@bs^{\wedge}) = leaf \text{ } t \text{ } bs$
 ⟨*proof*⟩

lemma *leaf-take*: $length \text{ } bs \geq h\text{-tree1 } t \implies leaf \text{ } t \text{ } (bs) = leaf \text{ } t \text{ } (take \text{ } (h\text{-tree1 } t) \text{ } bs)$
 ⟨*proof*⟩

lemma *Union-bits-le*:
 $h\text{-tree1 } t \leq n \implies (\bigcup bs \in \text{bits } n. \{leaf \text{ } t \text{ } bs\}) = (\bigcup bs \in \text{bits } (h\text{-tree1 } t). \{leaf \text{ } t \text{ } bs\})$
 ⟨*proof*⟩

lemma *image-leaf-bits-greater-eq*:
 $\langle leaf \text{ } t \text{ } \text{' } bits \text{ } n = leaf \text{ } t \text{ } \text{' } bits \text{ } (h\text{-tree1 } t) \rangle \text{ if } \langle h\text{-tree1 } t \leq n \rangle$
 ⟨*proof*⟩

lemma *set-tree1-leafs*:
 $\langle set\text{-tree1 } t = leaf \text{ } t \text{ } \text{' } bits \text{ } (h\text{-tree1 } t) \rangle$
 ⟨*proof*⟩

lemma *points-subset*: $inv\text{-kdt } k \text{ } t \implies h\text{-kdt } t \leq n \implies points \text{ } k \text{ } n \text{ } t \subseteq \text{nlists } k \{0..<2^{\wedge} n\}$
 ⟨*proof*⟩

fun *comb1* :: ('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow 'a tree1 \Rightarrow 'a tree1 \Rightarrow 'a tree1 **where**
 $comb1 \text{ } f \text{ } (Lf \text{ } x1) \text{ } (Lf \text{ } x2) = Lf \text{ } (f \text{ } x1 \text{ } x2) \mid$
 $comb1 \text{ } f \text{ } (Br \text{ } l1 \text{ } r1) \text{ } (Br \text{ } l2 \text{ } r2) = Br \text{ } (comb1 \text{ } f \text{ } l1 \text{ } l2) \text{ } (comb1 \text{ } f \text{ } r1 \text{ } r2) \mid$
 $comb1 \text{ } f \text{ } (Br \text{ } l1 \text{ } r1) \text{ } (Lf \text{ } x) = Br \text{ } (comb1 \text{ } f \text{ } l1 \text{ } (Lf \text{ } x)) \text{ } (comb1 \text{ } f \text{ } r1 \text{ } (Lf \text{ } x)) \mid$
 $comb1 \text{ } f \text{ } (Lf \text{ } x) \text{ } (Br \text{ } l2 \text{ } r2) = Br \text{ } (comb1 \text{ } f \text{ } (Lf \text{ } x) \text{ } l2) \text{ } (comb1 \text{ } f \text{ } (Lf \text{ } x) \text{ } r2)$

The last two equations cover cases that do not arise but are needed to prove that *comb1* only applies *f* to elements of the two trees, which implies this congruence lemma:

lemma *comb1-cong*[*fundef-cong*]:
 $\llbracket s1 = t1; s2 = t2; \bigwedge x y. x \in set\text{-tree1 } t1 \implies y \in set\text{-tree1 } t2 \implies f \text{ } x \text{ } y = g \text{ } x \text{ } y \rrbracket \implies comb1 \text{ } f \text{ } s1 \text{ } s2 = comb1 \text{ } g \text{ } t1 \text{ } t2$
 ⟨*proof*⟩

This congruence lemma in turn implies that *union* terminates because the recursive calls of *union* via *comb1* only involve elements from the two trees, which are smaller.

function (*sequential*) *union* :: *kdtb* \Rightarrow *kdtb* \Rightarrow *kdtb* **where**
 $union \text{ } (Cube \text{ } b) \text{ } t = (if \text{ } b \text{ then } Cube \text{ } True \text{ else } t) \mid$
 $union \text{ } t \text{ } (Cube \text{ } b) = (if \text{ } b \text{ then } Cube \text{ } True \text{ else } t) \mid$

$union (Dims t1) (Dims t2) = Dims (comb1 union t1 t2)$
 ⟨proof⟩

termination
 ⟨proof⟩

lemma leaf-comb1:
 $\llbracket length\ bs \geq \max (h-tree1\ t1)\ (h-tree1\ t2) \rrbracket \implies$
 $leaf\ (comb1\ f\ t1\ t2)\ bs = f\ (leaf\ t1\ bs)\ (leaf\ t2\ bs)$
 ⟨proof⟩

lemma leaf-in-set-tree1: $\llbracket length\ bs \geq h-tree1\ t \rrbracket \implies leaf\ t\ bs \in set-tree1\ t$
 ⟨proof⟩

lemma leaf-in-set-tree2: $\llbracket x \in nlists\ k\ UNIV; h-tree1\ t1 \leq k \rrbracket \implies leaf\ t1\ x \in set-tree1\ t1$
 ⟨proof⟩

lemma points-union:
 $\llbracket inv-kdt\ k\ t1; inv-kdt\ k\ t2; n \geq \max (h-kdt\ t1)\ (h-kdt\ t2) \rrbracket \implies$
 $points\ k\ n\ (union\ t1\ t2) = points\ k\ n\ t1 \cup points\ k\ n\ t2$
 ⟨proof⟩

lemma size-leaf[termination-simp]: $size\ (leaf\ t\ (map\ f\ ps)) < Suc\ (size-tree1\ size\ t)$
 ⟨proof⟩

fun get :: 'a kdt \Rightarrow nat list \Rightarrow 'a where
 $get\ (Cube\ b) - = b \mid$
 $get\ (Dims\ t)\ ps = get\ (leaf\ t\ (map\ even\ ps))\ (map\ (\lambda x. x\ div\ 2)\ ps)$

lemma map-zip1: $\llbracket length\ xs = length\ ys; \forall p \in set(zip\ xs\ ys). f\ p = fst\ p \rrbracket \implies$
 $map\ f\ (zip\ xs\ ys) = xs$
 ⟨proof⟩

lemma map-mv1: $\llbracket length\ ps = length\ bs \rrbracket \implies map\ even\ (mv\ bs\ ps) = bs$
 ⟨proof⟩

lemma map-zip2: $\llbracket length\ xs = length\ ys; \forall p \in set(zip\ xs\ ys). f\ p = snd\ p \rrbracket \implies$
 $map\ f\ (zip\ xs\ ys) = ys$
 ⟨proof⟩

lemma map-mv2: $\llbracket length\ ps = length\ bs \rrbracket \implies map\ (\lambda x. x\ div\ 2)\ (mv\ bs\ ps) = ps$
 ⟨proof⟩

lemma mv-map-map: $mv\ (map\ even\ ps)\ (map\ (\lambda x. x\ div\ 2)\ ps) = ps$
 ⟨proof⟩

lemma in-mv-image: $\llbracket ps \in nlists\ k\ \{0..<2*2^{\wedge}n\}; Ps \subseteq nlists\ k\ \{0..<2^{\wedge}n\}; bs \in$

$\text{bits } k \text{]} \implies$
 $ps \in mv \text{ } bs \text{ ' } Ps \iff map (\lambda x. x \text{ div } 2) ps \in Ps \wedge (bs = map \text{ even } ps)$
 $\langle proof \rangle$

lemma *get-points*: $\llbracket inv\text{-kdt } k \ t; h\text{-kdt } t \leq n; ps \in nlists \ k \ \{0..<2^n\} \rrbracket \implies$
 $get \ t \ ps = (ps \in points \ k \ n \ t)$
 $\langle proof \rangle$

fun *modify* :: $('a \Rightarrow 'a) \Rightarrow bool \ list \Rightarrow 'a \ tree1 \Rightarrow 'a \ tree1$ **where**
 $modify \ f \ [] \ (Lf \ x) = Lf \ (f \ x) \mid$
 $modify \ f \ (b\#bs) \ (Lf \ x) = (if \ b \ then \ Br \ (Lf \ x) \ (modify \ f \ bs \ (Lf \ x)) \ else \ Br \ (modify$
 $f \ bs \ (Lf \ x)) \ (Lf \ x)) \mid$
 $modify \ f \ (b\#bs) \ (Br \ l \ r) = (if \ b \ then \ Br \ l \ (modify \ f \ bs \ r) \ else \ Br \ (modify$
 $f \ bs \ l) \ r)$

fun *put* :: $'a \Rightarrow nat \Rightarrow nat \ list \Rightarrow 'a \ kdt \Rightarrow 'a \ kdt$ **where**
 $put \ b' \ 0 \ ps \ (Cube \ -) = Cube \ b' \mid$
 $put \ b' \ (Suc \ n) \ ps \ t =$
 $Dims \ (modify \ (put \ b' \ n \ (map \ (\lambda i. i \text{ div } 2) \ ps)) \ (map \ even \ ps))$
 $(case \ t \ of \ Cube \ b \Rightarrow Lf \ (Cube \ b) \mid Dims \ t \Rightarrow t)$

lemma *leaf-modify*: $\llbracket h\text{-tree1 } t \leq length \ bs; length \ bs' = length \ bs \rrbracket \implies$
 $leaf \ (modify \ f \ bs \ t) \ bs' = (if \ bs' = bs \ then \ f(leaf \ t \ bs) \ else \ leaf \ t \ bs')$
 $\langle proof \rangle$

lemma *in-nlists2D*: $xs \in nlists \ k \ \{0..<2 * 2^n\} \implies \exists bs \in nlists \ k \ UNIV. xs \in$
 $mv \ bs \text{ ' } nlists \ k \ \{0..<2^n\}$
 $\langle proof \rangle$

lemma *nlists2-simp*: $nlists \ k \ \{0..<2 * 2^n\} = (\bigcup bs \in nlists \ k \ UNIV. mv \ bs \text{ ' }$
 $nlists \ k \ \{0..<2^n\})$
 $\langle proof \rangle$

lemma *mv-diff*:
 $\llbracket length \ qs = length \ bs; \forall as \in A. length \ as = length \ bs \rrbracket \implies mv \ bs \text{ ' } (A - \{qs\})$
 $= mv \ bs \text{ ' } A - \{mv \ bs \ qs\}$
 $\langle proof \rangle$

lemma *put-points*: $\llbracket inv\text{-kdt } k \ t; h\text{-kdt } t \leq n; ps \in nlists \ k \ \{0..<2^n\} \rrbracket \implies$
 $points \ k \ n \ (put \ b \ n \ ps \ t) = (if \ b \ then \ points \ k \ n \ t \cup \{ps\} \ else \ points \ k \ n \ t - \{ps\})$
 $\langle proof \rangle$

end

References

- [1] S. Aluru. Quadrees and octrees. In D. P. Mehta and S. Sahni, editors, *Handbook of Data Structures and Applications*. Chapman and Hall/CRC, 2nd edition, 2017.
- [2] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509-517, 1975.
- [3] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209-226, 1977.
- [4] M. Rau. Multidimensional binary search trees. *Archive of Formal Proofs*, May 2019. https://isa-afp.org/entries/KD_Tree.html, Formal proof development.
- [5] H. Samet. The quadtree and related hierarchical data structures. *ACM Comput. Surv.*, 16(2):187-260, 1984.
- [6] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.
- [7] D. S. Wise. Representing matrices as quadtrees for parallel processors: extended abstract. *SIGSAM Bull.*, 18(3):24-25, 1984.
- [8] D. S. Wise. Representing matrices as quadtrees for parallel processors. *Inf. Process. Lett.*, 20(4):195-199, 1985.
- [9] D. S. Wise. Parallel decomposition of matrix inversion using quadtrees. In *International Conference on Parallel Processing, ICPP'86*, pages 92-99. IEEE Computer Society Press, 1986.
- [10] D. S. Wise. Matrix algebra and applicative programming. In G. Kahn, editor, *Functional Programming Languages and Computer Architecture*, volume 274, pages 134-153, 1987.
- [11] D. S. Wise. Matrix algorithms using quadtrees (invited talk). In G. Hains and L. M. R. Mullin, editors, *ATABLE-92, Intl. Workshop on Arrays, Functional Languages and Parallel Systems*, 1992.