Region Quadtrees

Tobias Nipkow
Technical University of Munich

March 17, 2025

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-dimenstional 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

2 Quad Tree Basics

theory *Quad-Base*
imports *HOL−Library.Tree*
begin

datatype $'a$ *qtree* = $L\ 'a \mid Q\ 'a\ qtree\ 'a\ qtree\ 'a\ qtree\ 'a\ qtree$

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*: $[\![\ i < 2{*}m;\ \neg\ i < m\ ]\!] \Longrightarrow i\ mod\ m = i - (m{::}nat)$
  ⟨*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* ∧ *compressed t1* ∧ *compressed t2* ∧ *compressed t3*)
   ∧ ¬ (∃ *x*. *t0* = *L x* ∧ *t1* = *t0* ∧ *t2* = *t0* ∧ *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 Quadtrees

type-synonym *qtb* = *bool qtree*

3.3.1 Abstraction of boolean quadtrees to sets of points

Superceded by the more general *get* abstraction.

type-synonym *points* = (*nat* × *nat*) *set*

abbreviation *sq* :: *nat* ⇒ *points* **where**
sq (*n*::*nat*) ≡ {*0*..<*2* ^ *n*} × {*0*..<*2* ^ *n*}

definition *shift* :: *nat* ⇒ *nat* ⇒ *nat* ∗ *nat* ⇒ *nat* ∗ *nat* **where**
shift di dj = (λ(*i*,*j*). (*i*+*di*, *j*+*dj*))

lemma *shift-pair*[*simp*]: *shift di dj* (*a*,*b*) = (*a*+*di*,*b*+*dj*)
⟨*proof*⟩

lemma *in-shift-image*: (*x*,*y*) ∈ *shift di dj* ' *M* ⟷ *di* ≤ *x* ∧ *dj* ≤ *y* ∧ (*x*−*di*,*y*−*dj*) ∈ *M*
⟨*proof*⟩

lemma *inj-shift*: *inj* (*shift i j*)
⟨*proof*⟩

lemma *shift-disj-shift*: ⟦ *s* ⊆ *sq n*; *s'* ⊆ *sq n*;
i ≥ *i'* + *2*^*n* ∨ *i'* ≥ *i* + *2*^*n* ∨ *j* ≥ *j'* + *2*^*n* ∨ *j'* ≥ *j* + *2*^*n* ⟧ ⟹
shift i j ' *s* ∩ *shift i' j'* ' *s'* = {}
⟨*proof*⟩

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* ⇒ *points* ⇒ *points* ⇒ *points* ⇒ *points* ⇒ *points* **where**
Qsq n A0 A1 A2 A3 =
shift 0 0 ' *A0* ∪ *shift 0* (*2*^*n*) ' *A1* ∪ *shift* (*2*^*n*) *0* ' *A2* ∪ *shift* (*2*^*n*) (*2*^*n*) ' *A3*

lemma *sq-Suc-Qsq*: {*0*..<*2* ∗ *2* ^ *n*} × {*0*..<*2* ∗ *2* ^ *n*} = *Qsq n* (*sq n*) (*sq n*) (*sq n*) (*sq n*)
⟨*proof*⟩

fun *points* :: *nat* ⇒ *qtb* ⇒ (*nat* ∗ *nat*) *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-subset*: *height t* ≤ *n* ⟹ *points n t* ⊆ *sq n*
⟨*proof*⟩

lemma *point-Suc-Qc*[*simp*]: *points* (*Suc n*) (*Qc t0 t1 t2 t3*) = *points* (*Suc n*) (*Q t0 t1 t2 t3*)
⟨*proof*⟩

lemma *get-points*: ⟦ *height t* ≤ *n*; (*i*,*j*) ∈ *sq n* ⟧ ⟹ *get n t i j* = ((*i*,*j*) ∈ *points n t*)

$\langle proof \rangle$

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*:
$Qsq\ n\ A0\ A1\ A2\ A3 \cup Qsq\ n\ B0\ B1\ B2\ B3 = Qsq\ n\ (A0 \cup B0)\ (A1 \cup B1)\ (A2 \cup B2)\ (A3 \cup B3)$
$\langle proof \rangle$

lemma *points-union*:
$max\ (height\ t1)\ (height\ t2) \le n \Longrightarrow points\ n\ (union\ t1\ t2) = points\ n\ t1 \cup points\ n\ t2$
$\langle proof \rangle$

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

lemma *height-union2*: $[\![\ height\ t1 \le n;\ height\ t2 \le n\ ]\!] \Longrightarrow height\ (union\ t1\ t2) \le n$
$\langle proof \rangle$

lemma *get-union*:
$max\ (height\ t1)\ (height\ t2) \le n \Longrightarrow get\ n\ (union\ t1\ t2)\ i\ j = (get\ n\ t1\ i\ j \vee get\ n\ t2\ i\ j)$
$\langle proof \rangle$

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

lemma *Qsq-inter*:
$[\![\ A0 \subseteq sq\ n;\ A1 \subseteq sq\ n;\ A2 \subseteq sq\ n;\ A3 \subseteq sq\ n;$
$B0 \subseteq sq\ n;\ B1 \subseteq sq\ n;\ B2 \subseteq sq\ n;\ B3 \subseteq sq\ n\ ]\!]$
$\Longrightarrow Qsq\ n\ A0\ A1\ A2\ A3 \cap Qsq\ n\ B0\ B1\ B2\ B3 = Qsq\ n\ (A0 \cap B0)\ (A1 \cap B1)\ (A2 \cap B2)\ (A3 \cap B3)$
⟨*proof*⟩

lemma *points-inter*: $n \geq max\ (height\ t1)\ (height\ t2) \Longrightarrow$
$points\ n\ (inter\ t1\ t2) = points\ n\ t1 \cap points\ n\ t2$
⟨*proof*⟩

lemma *height-inter*: $height\ (inter\ t1\ t2) \leq max\ (height\ t1)\ (height\ t2)$
⟨*proof*⟩

lemma *height-inter2*: $[\![\ height\ t1 \leq n;\ height\ t2 \leq n\ ]\!] \Longrightarrow height\ (inter\ t1\ t2) \leq n$
⟨*proof*⟩

lemma *get-inter*:
$[\![\ height\ t1 \leq n;\ height\ t2 \leq n\ ]\!] \Longrightarrow get\ n\ (inter\ t1\ t2)\ i\ j = (get\ n\ t1\ i\ j \wedge get\ n\ t2\ i\ j)$
⟨*proof*⟩

lemma *compressed-inter*: $compressed\ t1 \Longrightarrow compressed\ t2 \Longrightarrow compressed(inter\ t1\ t2)$
⟨*proof*⟩

lemma *Qsq-diff*: $[\![\ B0 \subseteq sq\ n;\ B1 \subseteq sq\ n;\ B2 \subseteq sq\ n;\ B3 \subseteq sq\ n;\ A0 \subseteq sq\ n;\ A1 \subseteq sq\ n;\ A2 \subseteq sq\ n;\ A3 \subseteq sq\ n\ ]\!] \Longrightarrow$
$Qsq\ n\ B0\ B1\ B2\ B3 - Qsq\ n\ A0\ A1\ A2\ A3 = Qsq\ n\ (B0 - A0)\ (B1 - A1)\ (B2 - A2)\ (B3 - A3)$
⟨*proof*⟩

lemma *points-negate*: $n \geq height\ t \Longrightarrow points\ n\ (negate\ t) = sq\ n - points\ n\ t$
⟨*proof*⟩

lemma *negate-eq-L-iff*: $compressed\ t \Longrightarrow negate\ t = L\ x \longleftrightarrow t = L(\neg x)$
⟨*proof*⟩

lemma *compressed-negate*: $compressed\ t \Longrightarrow compressed(negate\ t)$
⟨*proof*⟩

lemma *points-diff*: $n \geq max\ (height\ t1)\ (height\ t2) \Longrightarrow$
$points\ n\ (diff\ t1\ t2) = points\ n\ t1 - points\ n\ t2$

⟨*proof*⟩

lemma *compressed-diff*: $compressed\ t1 \Longrightarrow compressed\ t2 \Longrightarrow compressed(diff\ t1\ t2)$
⟨*proof*⟩

3.4 Operation *put*

fun *put* :: $nat \Rightarrow nat \Rightarrow 'a \Rightarrow nat \Rightarrow 'a\ qtree \Rightarrow 'a\ qtree$ **where**
$put\ i\ j\ a\ 0\ (L\ \text{-}) = L\ a\ |$
$put\ i\ j\ a\ (Suc\ n)\ t = modify\ (put\ (i\ mod\ 2\hat{}n)\ (j\ mod\ 2\hat{}n)\ a\ n)\ (i < 2\hat{}n)\ (j < 2\hat{}n)$
$(case\ t\ of\ L\ b \Rightarrow (L\ b,\ L\ b,\ L\ b,\ L\ b)\ |\ Q\ t0\ t1\ t2\ t3 \Rightarrow (t0,t1,t2,t3))$

lemma *points-put*: $[\![\ height\ t \le n;\ (i,j) \in sq\ n\ ]\!] \Longrightarrow$
$points\ n\ (put\ i\ j\ b\ n\ t) = (if\ b\ then\ points\ n\ t \cup \{(i,j)\}\ else\ points\ n\ t - \{(i,j)\})$
⟨*proof*⟩

lemma *height-put*: $height\ t \le n \Longrightarrow height\ (put\ i\ j\ a\ n\ t) \le n$
⟨*proof*⟩

lemma *get-put*: $[\![\ height\ t \le n;\ (i,j) \in sq\ n;\ (i',j') \in sq\ n\ ]\!] \Longrightarrow$
$get\ n\ (put\ i\ j\ a\ n\ t)\ i'\ j' = (if\ i'{=}i \wedge j'{=}j\ then\ a\ else\ get\ n\ t\ i'\ j')$
⟨*proof*⟩

lemma *compressed-put*:
$[\![\ height\ t \le n;\ compressed\ t\ ]\!] \Longrightarrow compressed\ (put\ i\ j\ a\ n\ t)$
⟨*proof*⟩

3.5 Extract Square

fun *get-sq* :: $nat \Rightarrow 'a\ qtree \Rightarrow nat \Rightarrow nat \Rightarrow nat \Rightarrow 'a\ qtree$ **where**
$get\text{-}sq\ n\ (L\ b)\ m\ i\ j = L\ b\ |$
$get\text{-}sq\ n\ t\ 0\ i\ j = L\ (get\ n\ t\ i\ j)\ |$
$get\text{-}sq\ (Suc\ n)\ (Q\ t0\ t1\ t2\ t3)\ (Suc\ m)\ i\ j =$
$(if\ i\ mod\ 2\hat{}n + 2\hat{}(m+1) \le 2\hat{}n \wedge j\ mod\ 2\hat{}n + 2\hat{}(m+1) \le 2\hat{}n$
$then\ get\text{-}sq\ n\ (select\ (i < 2\hat{}n)\ (j < 2\hat{}n)\ t0\ t1\ t2\ t3)\ (m+1)\ (i\ mod\ 2\hat{}n)\ (j\ mod\ 2\hat{}n)$
$else\ qf\ Qc\ (get\text{-}sq\ (Suc\ n)\ (Q\ t0\ t1\ t2\ t3)\ m)\ i\ j\ (2\hat{}m))$

lemma *shift-shift*: $shift\ i\ j\ `\ (shift\ i'\ j'\ `\ s) = shift\ (i+i')\ (j+j')\ `\ s$
⟨*proof*⟩
lemma *shift-shift2*: $shift\ i\ j\ `\ (shift\ i'\ j'\ `\ s) = shift\ (i'+i)\ (j'+j)\ `\ s$
⟨*proof*⟩

lemma *shift-split*: $shift\ i\ j\ `\ s =$
$shift\ (i - i\ mod\ 2\hat{}n)\ (j - j\ mod\ 2\hat{}n)\ `\ (shift\ (i\ mod\ 2\hat{}n)\ (j\ mod\ 2\hat{}n)\ `\ s)$
⟨*proof*⟩

lemma *plus-pow-aux*: $(i{::}nat) + 2\hat{}m \le 2{*}2\hat{}n \Longrightarrow i < 2 * 2\ \hat{}\ n$

$\langle proof \rangle$

lemma *Qsq-lem*: $[\![A0 \subseteq sq\ n;\ A1 \subseteq sq\ n;\ A2 \subseteq sq\ n;\ A3 \subseteq sq\ n;$
$i + 2 \mathbin{\hat{}} m \leq 2 \mathbin{\hat{}} Suc\ n;\ j + 2 \mathbin{\hat{}} m \leq 2 \mathbin{\hat{}} Suc\ n;$
$i\ mod\ 2 \mathbin{\hat{}} n + 2 \mathbin{\hat{}} m \leq 2 \mathbin{\hat{}} n;\ j\ mod\ 2 \mathbin{\hat{}} n + 2 \mathbin{\hat{}} m \leq 2 \mathbin{\hat{}} n\ ]\!] \Longrightarrow$
$Qsq\ n\ A0\ A1\ A2\ A3 \cap shift\ i\ j\ `\ sq\ m =$
$shift\ (i - i\ mod\ 2\hat{}n)\ (j - j\ mod\ 2\hat{}n)\ `\ select\ (i < 2 \mathbin{\hat{}} n)\ (j < 2 \mathbin{\hat{}} n)\ A0\ A1$
$A2\ A3 \cap shift\ i\ j\ `\ sq\ m$
$\langle proof \rangle$

lemma *f-select*: $f\ (select\ x\ y\ a\ b\ c\ d) = select\ x\ y\ (f\ a)\ (f\ b)\ (f\ c)\ (f\ d)$
$\langle proof \rangle$

lemma *height-get-sq*: $m \leq n \Longrightarrow height\ (get\text{-}sq\ n\ t\ m\ i\ j) \leq m$
$\langle proof \rangle$

lemma *shift-Qsq*: $shift\ i\ j\ `\ Qsq\ n\ A0\ A1\ A2\ A3 =$
$Qsq\ n\ (shift\ i\ j\ `\ A0)\ (shift\ i\ j\ `\ A1)\ (shift\ i\ j\ `\ A2)\ (shift\ i\ j\ `\ A3)$
$\langle proof \rangle$

lemma *points-get-sq*:
$[\![height\ t \leq n;\ i + 2\hat{}m \leq 2\hat{}n;\ j + 2\hat{}m \leq 2\hat{}n\ ]\!] \Longrightarrow$
$shift\ i\ j\ `\ points\ m\ (get\text{-}sq\ n\ t\ m\ i\ j) = points\ n\ t \cap (shift\ i\ j\ `\ sq\ m)$
$\langle proof \rangle$

lemma *get-get-sq*:
$[\![height\ t \leq n;\ i + 2\hat{}m \leq 2\hat{}n;\ j + 2\hat{}m \leq 2\hat{}n;\ i' < 2\hat{}m;\ j' < 2\hat{}m\ ]\!] \Longrightarrow$
$get\ m\ (get\text{-}sq\ n\ t\ m\ i\ j)\ i'\ j' = get\ n\ t\ (i{+}i')\ (j{+}j')$
$\langle proof \rangle$

lemma *compressed-get-sq*:
$[\![height\ t \leq n;\ compressed\ t\ ]\!] \Longrightarrow compressed\ (get\text{-}sq\ n\ t\ m\ i\ j)$
$\langle proof \rangle$

3.6 From Matrix to Quadtree

3.6.1 Matrix as list of lists

type-synonym $'a\ mx = 'a\ list\ list$

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

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

Decompose matrix into submatrices

definition *decomp* **where**
$decomp\ n\ mx = (let\ mx01 = take\ (2\hat{}n)\ mx;\ mx23 = drop\ (2\hat{}n)\ mx$
$in\ (map\ (take\ (2\hat{}n))\ mx01,\ map\ (drop\ (2\hat{}n))\ mx01,\ map\ (take\ (2\hat{}n))\ mx23,$
$map\ (drop\ (2\hat{}n))\ mx23))$

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

Quadtree of matrix:

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

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

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

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

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

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**
$mx\text{-}of\ n\ (L\ x) = replicate\ (2\hat{}n)\ (replicate\ (2\hat{}n)\ x)\ |$
$mx\text{-}of\ (Suc\ n)\ (Q\ t0\ t1\ t2\ t3) =$
$Qmx\ (mx\text{-}of\ n\ t0)\ (mx\text{-}of\ n\ t1)\ (mx\text{-}of\ n\ t2)\ (mx\text{-}of\ n\ t3)$

lemma *nth-Qmx-select*: $[\![\ sq\text{-}mx\ n\ mx0;\ sq\text{-}mx\ n\ mx1;\ sq\text{-}mx\ n\ mx2;\ sq\text{-}mx\ n\ mx3;\ i < 2{*}2\hat{}n;\ j < 2{*}2\hat{}n\ ]\!] \Longrightarrow$
$Qmx\ mx0\ mx1\ mx2\ mx3\ !\ i\ !\ j = select\ (i < 2\hat{}n)\ (j < 2\hat{}n)\ mx0\ mx1\ mx2\ mx3\ !\ (i\ mod\ 2\hat{}n)\ !\ (j\ mod\ 2\hat{}n)$
$\langle proof \rangle$

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

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

lemma *mx-of-get*: $[\![$ *height* $t \le n$; $(i,j) \in sq\ n$ $]\!] \Longrightarrow mx\text{-}of\ n\ t\ !\ i\ !\ j = get\ n\ t\ i\ j$
$\langle proof \rangle$

end

4 Block Matrices via Quad Trees

theory *Quad-Matrix*
imports
Complex-Main
Quad-Base
begin

There are two possible representations of marices as quadtrees. In this file we use the standard quadtree with two constructors L and Q. $L\ x$ represents the x-diagonal ma 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 1x1 ma x. The advantage is that there are fewer cases in function definitions because one cannot add/multiply L and Q: the have different dimensions. However, $L\ 0$ is special: it still represents the 0 ma 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 ma are 0. This is maintained by addition; multiplication and diagonal need an explicit argument n to maintain it.

definition *mk-sq* :: $nat \Rightarrow ma \Rightarrow ma$ **where**
$mk\text{-}sq\ n\ a = (\lambda i\ j.\ if\ i < 2\hat{}n \wedge j < 2\hat{}n\ then\ a\ i\ j\ else\ 0)$

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

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

definition *diag-ma* :: $nat \Rightarrow real \Rightarrow ma$ **where**
$diag\text{-}ma\ n\ x = mk\text{-}sq\ n\ (\lambda i\ j.\ if\ i{=}j\ then\ x\ else\ 0)$

definition *add-ma* :: $ma \Rightarrow ma \Rightarrow ma$ **where**
$add\text{-}ma\ a\ b = (\lambda i\ j.\ a\ i\ j + b\ i\ j)$

definition *mult-ma* :: *nat* ⇒ *ma* ⇒ *ma* ⇒ *ma* **where**
mult-ma n a b = $(\lambda i\ j.\ \sum k{=}0..{<}2\hat{\ }n.\ a\ i\ k * b\ k\ j)$

4.2 Matrix Lemmas

lemma *add-ma-diag-ma*[*simp*]: *add-ma* (*diag-ma n x*) (*diag-ma n y*) = *diag-ma n* (*x*+*y*)
⟨*proof*⟩

lemma *add-ma-diag-ma-0*[*simp*]: *add-ma* (*diag-ma n 0*) *a* = *a*
⟨*proof*⟩

lemma *add-ma-diag-ma-02*[*simp*]: *add-ma a* (*diag-ma n 0*) = *a*
⟨*proof*⟩

lemma *mult-ma-diag-ma-0*[*simp*]: *mult-ma n* (*diag-ma n 0*) *a* = *diag-ma n 0*
⟨*proof*⟩

lemma *mult-ma-diag-ma-02*[*simp*]: *mult-ma n a* (*diag-ma n 0*) = *diag-ma n 0*
⟨*proof*⟩

lemma *mult-ma-diag-ma-diag-ma*[*simp*]: *mult-ma n* (*diag-ma n x*) (*diag-ma n y*) = *diag-ma n* (*x*∗*y*)
⟨*proof*⟩

4.3 Real Quad Trees and Abstraction to Matrices

type-synonym *qtr* = *real qtree*

fun *compressed* :: *qtr* ⇒ *bool* **where**
compressed (*L x*) = *True* |
compressed (*Q* (*L x0*) (*L x1*) (*L x2*) (*L x3*)) = (¬ (*x1*=*0* ∧ *x2*=*0* ∧ *x0*=*x3*)) |
compressed (*Q t0 t1 t2 t3*) = (*compressed t0* ∧ *compressed t1* ∧ *compressed t2* ∧ *compressed t3*)

lemma *compressed-Q*:
compressed (*Q t0 t1 t2 t3*) ⟹ (*compressed t0* ∧ *compressed t1* ∧ *compressed t2* ∧ *compressed t3*)
⟨*proof*⟩

definition *Qma* :: *nat* ⇒ *ma* ⇒ *ma* ⇒ *ma* ⇒ *ma* ⇒ *ma* **where**
Qma n a b c d =
(λ*i j*. *if i* < 2^*n then if j* < 2^*n then a i j else b i* (*j* − 2^*n*) *else*
if j < 2^*n then c* (*i* − 2^*n*) *j else d* (*i* − 2^*n*) (*j* − 2^*n*))

lemma *add-ma-Qma*:
add-ma (*Qma n a b c d*) (*Qma n a′ b′ c′ d′*) =
Qma n (*add-ma a a′*) (*add-ma b b′*) (*add-ma c c′*) (*add-ma d d′*)
⟨*proof*⟩

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

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

lemma *diag-ma-Suc*: *diag-ma* (*Suc n*) *x* = *Qma n* (*diag-ma n x*) (*diag-ma n 0*) (*diag-ma n 0*) (*diag-ma n x*)
⟨*proof*⟩

Abstraction function:

fun *ma* :: *nat* ⇒ *qtr* ⇒ *ma* **where**
ma n (*L x*) = *diag-ma n x* |
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* ⇒ *qtr* ⇒ *qtr* ⇒ *qtr* ⇒ *qtr* **where**
Qc (*L x0*) (*L x1*) (*L x2*) (*L x3*) =
(*if x1=0* ∧ *x2=0* ∧ *x0=x3 then L x0 else Q* (*L x0*) (*L x1*) (*L x2*) (*L x3*)) |
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* ∧ *compressed t1* ∧ *compressed t2* ∧ *compressed t3*)
⟨*proof*⟩

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

fun *add* :: *qtr* ⇒ *qtr* ⇒ *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*) |
add (*L x*) (*L y*) = *L*(*x*+*y*) |
add (*L x*) (*Q t0 t1 t2 t3*) = *Qc* (*add* (*L x*) *t0*) *t1 t2* (*add* (*L x*) *t3*) |
add (*Q t0 t1 t2 t3*) (*L x*) = *Qc* (*add t0* (*L x*)) *t1 t2* (*add t3* (*L x*))

fun *mult* :: *qtr* ⇒ *qtr* ⇒ *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*)) |

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

Initialization of *qtr* from ma

fun $qtr :: nat \Rightarrow ma \Rightarrow qtr$ **where**
$qtr\ 0\ a = L(a\ 0\ 0)\ |$
$qtr\ (Suc\ n)\ a =$
$\ (let\ t0 = qtr\ n\ a;\ t1 = qtr\ n\ (\lambda i\ j.\ a\ i\ (j+2\hat{}n));$
$\quad t2 = qtr\ n\ (\lambda i\ j.\ a\ (i+2\hat{}n)\ j);\ t3 = qtr\ n\ (\lambda i\ j.\ a\ (i+2\hat{}n)\ (j+2\hat{}n))$
$\ \ in\ Q\ t0\ t1\ t2\ t3)$

4.5 Correctness of Quad Tree Implementations

4.5.1 *add*

lemma *ma-add*: $[\![\ height\ s \leq n;\ height\ t \leq n\ ]\!] \Longrightarrow$
$\ ma\ n\ (add\ s\ t) = add\text{-}ma\ (ma\ n\ s)\ (ma\ n\ t)$
⟨*proof*⟩

lemma *height-add*: $height\ (add\ s\ t) \leq max\ (height\ s)\ (height\ t)$
⟨*proof*⟩

lemma *compressed-add*: $[\![\ compressed\ s;\ compressed\ t\ ]\!] \Longrightarrow compressed\ (add\ s\ t)$
⟨*proof*⟩

lemma *Max4*: $Max\{n0,n1,n2,n3\} = max\ n0\ (max\ n1\ (max\ n2\ n3))$⟨*proof*⟩

lemma *height-mult*: $height\ (mult\ s\ t) \leq max\ (height\ s)\ (height\ t)$
⟨*proof*⟩

4.5.2 *mult*

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

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

$(add\text{-}ma\ (mult\text{-}ma\ n\ c\ b')\ (mult\text{-}ma\ n\ d\ d')))$
⟨*proof*⟩

lemma *ma-mult*: $[\![\ height\ s \le n;\ height\ t \le n\ ]\!] \Longrightarrow$
$ma\ n\ (mult\ s\ t) = mult\text{-}ma\ n\ (ma\ n\ s)\ (ma\ n\ t)$
⟨*proof*⟩

lemma *compressed-mult*: $[\![\ compressed\ s;\ compressed\ t\ ]\!] \Longrightarrow compressed\ (mult\ s\ t)$
⟨*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 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 ⟨*proof*⟩

end

lemma *height-0-iff*: *height t* = *0* ⟷ (∃ *x*. *t* = *Box x*)
⟨*proof*⟩

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

lemma *bits-0*[*code*]: *bits 0* = {[]}
⟨*proof*⟩

lemma *bits-Suc*[*code*]:
bits (*Suc n*) = (*let B* = *bits n in* (#) *True* ‘ *B* ∪ (#) *False* ‘ *B*)
⟨*proof*⟩

5.1 Subtree

fun *subtree* :: *'a kdt* ⇒ *bool list* ⇒ *'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*) ≤ *height t* − *length bs*
⟨*proof*⟩

lemma *height-subtree2*: ⟦ *height t* ≤ *k* ∗ (*Suc n*); *length bs* = *k*⟧ ⟹ *height* (*subtree t bs*) ≤ *k* ∗ *n*
⟨*proof*⟩

lemma *subtree-Split-Box*: *length bs* ≠ *0* ⟹ *subtree* (*Split* (*Box b*) (*Box b*)) *bs* = *Box b*
⟨*proof*⟩

5.2 Shifting a coordinate by a boolean vector

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

lemma *map-zip1*: ⟦ *length xs* = *length ys*; ∀ *p* ∈ *set*(*zip xs ys*). *f p* = *fst p* ⟧ ⟹ *map f* (*zip xs ys*) = *xs*
⟨*proof*⟩

lemma *map-mv1*: $[\![$ *ps* $\in$ *nlists* (*length bs*) $\{0..<n\}$; *length ps* = *length bs* $]\!]$
$\Longrightarrow$ *map* ($\lambda i.\ i < n$) (*mv* (*n*) *bs ps*) = *bs*
⟨*proof*⟩

lemma *map-zip2*: $[\![$ *length xs* = *length ys*; $\forall$ *p* $\in$ *set*(*zip xs ys*). *f p* = *snd p* $]\!]$ $\Longrightarrow$
map f (*zip xs ys*) = *ys*
⟨*proof*⟩

lemma *map-mv2*: $[\![$ *ps* $\in$ *nlists* (*length bs*) $\{0..<2\hat{}n\}$ $]\!]$ $\Longrightarrow$ *map* ($\lambda x.\ x$ *mod* $2\hat{}n$)
(*mv* ($2\hat{}n$) *bs ps*) = *ps*
⟨*proof*⟩

lemma *mv-map-map*: *set ps* $\subseteq$ $\{0..<2 * n\}$ $\Longrightarrow$ *mv* (*n*) (*map* ($\lambda x.\ x < n$) *ps*)
(*map* ($\lambda x.\ x$ *mod n*) *ps*) = *ps*
⟨*proof*⟩

lemma *mv-in-nlists*:
$[\![$ *p* $\in$ *nlists k* $\{0..<2\ \hat{}\ n\}$; *bs* $\in$ *bits k* $]\!]$ $\Longrightarrow$ *mv* ($2\hat{}n$) *bs p* $\in$ *nlists k* $\{0..<2 *$
$2\ \hat{}\ n\}$
⟨*proof*⟩

lemma *in-nlists2D*: *xs* $\in$ *nlists k* $\{0..<2 * 2\hat{}n\}$ $\Longrightarrow$ $\exists$ *bs*$\in$ *bits k*. *xs* $\in$ *mv* ($2\hat{}n$)
bs ‘ *nlists k* $\{0..<2\hat{}n\}$
⟨*proof*⟩

lemma *nlists2-simp*: *nlists k* $\{0..<2 * 2\ \hat{}\ n\}$ = ($\bigcup$ *bs*$\in$ *bits k*. *mv* ($2\hat{}n$) *bs* ‘ *nlists*
k $\{0..<2\ \hat{}\ n\}$)
⟨*proof*⟩

lemma *in-mv-image*: $[\![$ *ps* $\in$ *nlists k* $\{0..<2*2\hat{}n\}$; *Ps* $\subseteq$ *nlists k* $\{0..<2\hat{}n\}$; *bs* $\in$
bits k $]\!]$ $\Longrightarrow$
ps $\in$ *mv* ($2\hat{}n$) *bs* ‘ *Ps* $\longleftrightarrow$ *map* ($\lambda x.\ x$ *mod* $2\hat{}n$) *ps* $\in$ *Ps* $\wedge$ (*bs* = *map* ($\lambda i.\ i <$
$2\hat{}n$) *ps*)
⟨*proof*⟩

5.3 Points in a tree

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

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

lemma *points-Suc*: *points k* (*Suc n*) *t* = ($\bigcup$ *bs* $\in$ *bits k*. *mv* ($2\hat{}n$) *bs* ‘ *points k n*
(*subtree t bs*))
⟨*proof*⟩

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

5.4 Compression

Compressing Split:

fun *SplitC* :: $'a\ kdt \Rightarrow 'a\ kdt \Rightarrow 'a\ kdt$ **where**
$SplitC\ (Box\ b1)\ (Box\ b2) = (if\ b1{=}b2\ then\ Box\ b1\ else\ Split\ (Box\ b1)\ (Box\ b2))$ |
$SplitC\ l\ r = Split\ l\ r$

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

lemma *compressedI*: $[\![\ compressed\ l;\ compressed\ r\ ]\!] \Longrightarrow compressed\ (SplitC\ l\ r)$
$\langle proof \rangle$

lemma *subtree-SplitC*:
$1 \leq length\ bs \Longrightarrow subtree\ (SplitC\ l\ r)\ bs = subtree\ (Split\ l\ r)\ bs$
$\langle proof \rangle$

lemma *height-SplitC*: $height(SplitC\ l\ r) \leq Suc\ (max\ (height\ l)\ (height\ r))$
$\langle proof \rangle$

lemma *height-SplitC2*: $[\![\ height\ l \leq n;\ height\ r \leq n\ ]\!] \Longrightarrow height(SplitC\ l\ r) \leq Suc\ n$
$\langle proof \rangle$

5.5 Extracting a point from a tree

Also the abstraction function.

fun *get* :: $nat \Rightarrow 'a\ kdt \Rightarrow nat\ list \Rightarrow 'a$ **where**
$get\ \text{-}\ (Box\ b)\ \text{-} = b$ |
$get\ (Suc\ n)\ t\ ps = get\ n\ (subtree\ t\ (map\ (\lambda i.\ i < 2\hat{}n)\ ps))\ (map\ (\lambda i.\ i\ mod\ 2\hat{}n)\ ps)$

lemma *get-Suc*: $get\ (Suc\ n)\ t\ ps =$
$get\ n\ (subtree\ t\ (map\ (\lambda i.\ i < 2\ \hat{}\ n)\ ps))\ (map\ (\lambda i.\ i\ mod\ 2\ \hat{}\ n)\ ps)$
$\langle proof \rangle$

lemma *points-get*: $[\![\ height\ t \leq k*n;\ ps \in nlists\ k\ \{0..<2\hat{}n\}\ ]\!] \Longrightarrow$
$get\ n\ t\ ps = (ps \in points\ k\ n\ t)$
$\langle proof \rangle$

5.6 Modifying a point in a tree

fun *modify* :: $('a\ kdt \Rightarrow 'a\ kdt) \Rightarrow bool\ list \Rightarrow 'a\ kdt \Rightarrow 'a\ kdt$ **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 $put :: nat\ list \Rightarrow 'a \Rightarrow nat \Rightarrow 'a\ kdt \Rightarrow 'a\ kdt$ **where**
$put\ ps\ a\ 0\ (Box\ \text{-}) = Box\ a\ |$
$put\ ps\ a\ (Suc\ n)\ t = modify\ (put\ (map\ (\lambda i.\ i\ mod\ 2\hat{\ }n)\ ps)\ a\ n)\ (map\ (\lambda i.\ i < 2\hat{\ }n)\ ps)\ t$

lemma *height-modify*: $[\![\ \forall t.\ height\ t \le nk \longrightarrow height\ (f\ t) \le nk;$
$height\ t \le k + nk;\ length\ bs = k]\!]$
$\Longrightarrow height\ (modify\ f\ bs\ t) \le k + nk$
$\langle proof \rangle$

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

lemma *subtree-modify*: $[\![\ length\ bs' = length\ bs\ ]\!]$
$\Longrightarrow subtree\ (modify\ f\ bs\ t)\ bs' = (if\ bs' = bs\ then\ f(subtree\ t\ bs)\ else\ subtree\ t\ bs')$
$\langle proof \rangle$

lemma *mod-eq1*: $[\![\ y < 2 * n;\ ya < 2 * n;\ \neg\ ya < n;\ \neg\ y < n;\ ya\ mod\ n = y\ mod\ n]\!]$
$\Longrightarrow ya = (y::nat)$
$\langle proof \rangle$

lemma *nlist-eq-mod*: $[\![\ ps \in nlists\ k\ \{0..<(2::nat) * 2\ \hat{}\ n\};\ ps' \in nlists\ k\ \{0..<2 * 2\ \hat{}\ n\};$
$map\ (\lambda i.\ i < 2\ \hat{}\ n)\ ps' = map\ (\lambda i.\ i < 2\ \hat{}\ n)\ ps;\ ps' \ne ps\ ]\!] \Longrightarrow$
$map\ (\lambda i.\ i\ mod\ 2\ \hat{}\ n)\ ps' \ne map\ (\lambda i.\ i\ mod\ 2\ \hat{}\ n)\ ps$
$\langle proof \rangle$

lemma *get-put*: $[\![\ height\ t \le k{*}n;\ ps \in cube\ k\ n;\ ps' \in cube\ k\ n\ ]\!] \Longrightarrow$
$get\ n\ (put\ ps\ a\ n\ t)\ ps' = (if\ ps' = ps\ then\ a\ else\ get\ n\ t\ ps')$
$\langle proof \rangle$

lemma *compressed-modify*: $[\![\ compressed\ t;\ compressed\ (f\ (subtree\ t\ bs))\ ]\!] \Longrightarrow compressed\ (modify\ f\ bs\ t)$
$\langle proof \rangle$

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

lemma *compressed-put*:
$[\![\ height\ t \le k{*}n;\ k = length\ ps;\ compressed\ t\ ]\!] \Longrightarrow compressed\ (put\ ps\ a\ n\ t)$

⟨*proof*⟩

5.7 Union

fun *union* :: *kdtb* ⇒ *kdtb* ⇒ *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 *subtree-union*: *subtree* (*union t1 t2*) *bs* = *union* (*subtree t1 bs*) (*subtree t2 bs*)
⟨*proof*⟩

lemma *points-union*:
⟦ *max* (*height t1*) (*height t2*) ≤ *k*∗*n* ⟧ ⟹
points k n (*union t1 t2*) = *points k n t1* ∪ *points k n t2*
⟨*proof*⟩

lemma *get-union*:
⟦ *max* (*height t1*) (*height t2*) ≤ *length ps* ∗ *n* ⟧ ⟹
get n (*union t1 t2*) *ps* = (*get n t1 ps* ∨ *get n t2 ps*)
⟨*proof*⟩

lemma *height-union*: *height* (*union t1 t2*) ≤ *max* (*height t1*) (*height t2*)
⟨*proof*⟩

lemma *compressed-union*: *compressed t1* ⟹ *compressed t2* ⟹ *compressed*(*union t1 t2*)
⟨*proof*⟩

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* | *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 $2\hat{}k$ 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 $height\text{-}kdt :: 'a\ kdt \Rightarrow nat$ **where**
$height\ (Box\ \text{-}) = 0\ |$
$height\ (Split\ l\ r) = max\ (height\ l)\ (height\ r) + 1$

instance $\langle proof \rangle$

end

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

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

lemma $bits\text{-}Suc[code]$:
$bits\ (Suc\ n) = (let\ B = bits\ n\ in\ (\#)\ True\ `\ B \cup (\#)\ False\ `\ B)$
$\langle proof \rangle$

6.1 Subtree

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

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

lemma $height\text{-}subtree$: $height\ (subtree\ t\ bs) \le height\ t - length\ bs$
$\langle proof \rangle$

lemma $height\text{-}subtree2$: $[\![\ height\ t \le k * (Suc\ n);\ length\ bs = k]\!] \Longrightarrow height\ (subtree\ t\ bs) \le k * n$

$\langle proof \rangle$

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

6.2 Shifting a coordinate by a boolean vector

The ?

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

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

lemma *map-mv1*: $[\![\ length\ ps = length\ bs\ ]\!] \Longrightarrow map\ even\ (mv\ bs\ ps) = bs$
$\langle proof \rangle$

lemma *map-zip2*: $[\![\ length\ xs = length\ ys;\ \forall\, p \in set(zip\ xs\ ys).\ f\ p = snd\ p\ ]\!] \Longrightarrow map\ f\ (zip\ xs\ ys) = ys$
$\langle proof \rangle$

lemma *map-mv2*: $[\![\ length\ ps = length\ bs\ ]\!] \Longrightarrow map\ (\lambda x.\ x\ div\ 2)\ (mv\ bs\ ps) = ps$
$\langle proof \rangle$

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

lemma *mv-in-nlists*:
$[\![\ p \in nlists\ k\ \{0..{<}2\ \widehat{}\ n\};\ bs \in bits\ k\ ]\!] \Longrightarrow mv\ bs\ p \in nlists\ k\ \{0..{<}2 * 2\ \widehat{}\ n\}$
$\langle proof \rangle$

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

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

6.3 Points in a tree

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

fun $points :: nat \Rightarrow nat \Rightarrow kdtb \Rightarrow nat\ list\ set$ **where**

points k n (*Box b*) = (*if b then cube k n else* {}) |
points k (*Suc n*) *t* = ($\bigcup$ *bs* ∈ *bits k*. *mv bs* ‘ *points k n* (*subtree t bs*))

lemma *points-Suc*: *points k* (*Suc n*) *t* = ($\bigcup$ *bs* ∈ *bits k*. *mv bs* ‘ *points k n* (*subtree t bs*))
⟨*proof*⟩

lemma *points-subset*: *height t* ≤ *k*∗*n* ⟹ *points k n t* ⊆ *nlists k* {*0*..<*2*^*n*}
⟨*proof*⟩

6.4 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 t1 t2 = *Split t1 t2*

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 t1*; *compressed t2* ⟧ ⟹ *compressed* (*SplitC t1 t2*)
⟨*proof*⟩

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

6.5 Union

fun *union* :: *kdtb* ⇒ *kdtb* ⇒ *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*: ⟦ *ps* ∈ *nlists k* {*0*..<*2*∗*2*^*n*}; *Ps* ⊆ *nlists k* {*0*..<*2*^*n*}; *bs* ∈ *bits k* ⟧ ⟹
ps ∈ *mv bs* ‘ *Ps* ⟷ *map* (λ*x*. *x div 2*) *ps* ∈ *Ps* ∧ (*bs* = *map even ps*)
⟨*proof*⟩

lemma *subtree-union*: *subtree* (*union t1 t2*) *bs* = *union* (*subtree t1 bs*) (*subtree t2 bs*)
⟨*proof*⟩

lemma *points-union*:
$[\![$ *max* (*height t1*) (*height t2*) $\leq k{*}n$ $]\!] \Longrightarrow$
points k n (*union t1 t2*) = *points k n t1* $\cup$ *points k n t2*
⟨*proof*⟩

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

6.6 Extracting a point from a tree

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

For termination of *get*:

corollary *size-subtree-Split*[*termination-simp*]:
$bs \neq [] \Longrightarrow size\ (subtree\ (Split\ l\ r)\ bs) < Suc\ (size\ l + size\ r)$
⟨*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*: $[\![$ *height t* $\leq k{*}n$; *ps* $\in$ *nlists k* {*0..<2^n*} $]\!] \Longrightarrow$
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* $t \Longrightarrow$ *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*[*code*]: *bits 0* = {[]}
⟨*proof*⟩

lemma *bits-Suc*[*code*]: *bits* (*Suc n*) = (*let B* = *bits n in* (#) *True* ‘ *B* ∪ (#) *False* ‘ *B*)
⟨*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* (λ*b 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*: $\mathit{finite}\ (\mathit{bits}\ n)$
⟨*proof*⟩

lemma *mv-in-nlists*:
$[\![\ p \in \mathit{nlists}\ k\ \{0..<2\ \widehat{}\ n\};\ bs \in \mathit{bits}\ k\ ]\!] \Longrightarrow \mathit{mv}\ bs\ p \in \mathit{nlists}\ k\ \{0..<2 * 2\ \widehat{}\ n\}$
⟨*proof*⟩

lemma *leaf-append*: $\mathit{length}\ bs \geq \mathit{h\text{-}tree1}\ t \Longrightarrow \mathit{leaf}\ t\ (bs@bs') = \mathit{leaf}\ t\ bs$
⟨*proof*⟩

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

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

lemma *set-tree1-leafs*:
$\mathit{set\text{-}tree1}\ t = (\bigcup bs \in \mathit{bits}\ (\mathit{h\text{-}tree1}\ t).\ \{\mathit{leaf}\ t\ bs\})$
⟨*proof*⟩

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

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

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*]:
$[\![s1 = t1;\ s2 = t2;\ \bigwedge x\ y.\ x \in \mathit{set\text{-}tree1}\ t1 \Longrightarrow y \in \mathit{set\text{-}tree1}\ t2 \Longrightarrow f\ x\ y = g\ x\ y]\!] \Longrightarrow \mathit{comb1}\ f\ s1\ s2 = \mathit{comb1}\ g\ t1\ t2$
⟨*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* :: $\mathit{kdtb} \Rightarrow \mathit{kdtb} \Rightarrow \mathit{kdtb}$ **where**
$\mathit{union}\ (\mathit{Cube}\ b)\ t = (\mathit{if}\ b\ \mathit{then}\ \mathit{Cube}\ \mathit{True}\ \mathit{else}\ t)\ |$
$\mathit{union}\ t\ (\mathit{Cube}\ b) = (\mathit{if}\ b\ \mathit{then}\ \mathit{Cube}\ \mathit{True}\ \mathit{else}\ t)\ |$
$\mathit{union}\ (\mathit{Dims}\ t1)\ (\mathit{Dims}\ t2) = \mathit{Dims}\ (\mathit{comb1}\ \mathit{union}\ t1\ t2)$
⟨*proof*⟩
termination
⟨*proof*⟩

lemma *leaf-comb1*:
$\llbracket$ *length bs* $\geq$ *max* (*h-tree1 t1*) (*h-tree1 t2*) $\rrbracket \Longrightarrow$
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* $\rrbracket \Longrightarrow$ *leaf t bs* $\in$ *set-tree1 t*
⟨*proof*⟩

lemma *leaf-in-set-tree2*: $\llbracket$*x* $\in$ *nlists k UNIV*; *h-tree1 t1* $\leq$ *k*$\rrbracket \Longrightarrow$ *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*) $\rrbracket \Longrightarrow$
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* |
get (*Dims t*) *ps* = *get* (*leaf t* (*map even ps*)) (*map* (λ*x*. *x div 2*) *ps*)

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

lemma *map-mv1*: $\llbracket$ *length ps* = *length bs* $\rrbracket \Longrightarrow$ *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* $\rrbracket \Longrightarrow$ *map f* (*zip xs ys*) = *ys*
⟨*proof*⟩

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

lemma *mv-map-map*: *mv* (*map even ps*) (*map* (λ*x*. *x div 2*) *ps*) = *ps*
⟨*proof*⟩

lemma *in-mv-image*: $\llbracket$ *ps* $\in$ *nlists k* {*0..<2*2^n*}; *Ps* $\subseteq$ *nlists k* {*0..<2^n*}; *bs* $\in$ *bits k* $\rrbracket \Longrightarrow$
ps $\in$ *mv bs* ‘ *Ps* $\longleftrightarrow$ *map* (λ*x*. *x div 2*) *ps* $\in$ *Ps* $\wedge$ (*bs* = *map even ps*)
⟨*proof*⟩

lemma *get-points*: $\llbracket$ *inv-kdt k t*; *h-kdt t* $\leq$ *n*; *ps* $\in$ *nlists k* $\{0..<2\hat{\ }n\}$ $\rrbracket \Longrightarrow$
  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*) |
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*)) |
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'* |
put b' (*Suc n*) *ps t* =
  Dims (*modify* (*put b' n* (*map* (λi. *i div 2*) *ps*)) (*map even ps*)
      (*case t of Cube b* $\Rightarrow$ *Lf* (*Cube b*) | *Dims t* $\Rightarrow$ *t*))

lemma *leaf-modify*: $\llbracket$ *h-tree1 t* $\leq$ *length bs*; *length bs'* = *length bs* $\rrbracket \Longrightarrow$
  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* $\{0..<2 * 2 \hat{\ } n\}$ $\Longrightarrow \exists$ *bs*$\in$*nlists k UNIV*. *xs* $\in$ *mv bs* ` *nlists k* $\{0..<2 \hat{\ } n\}$
$\langle$*proof*$\rangle$

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

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

lemma *put-points*: $\llbracket$ *inv-kdt k t*; *h-kdt t* $\leq$ *n*; *ps* $\in$ *nlists k* $\{0..<2\hat{\ }n\}$ $\rrbracket \Longrightarrow$
 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. Quadtrees 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):509517, 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):209226, 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.