

Verification of the UpDown scheme

Johannes Hölzl

23. Februar 2021

Zusammenfassung

The UpDown scheme is a recursive scheme used to compute the stiffness matrix on a special form of sparse grids. Usually, when discretizing a Euclidean space of dimension d we need $O(n^d)$ points, for n points along each dimension. Sparse grids are a hierarchical representation where the number of points is reduced to $O(n \cdot \log(n)^d)$. One disadvantage of such sparse grids is that the algorithm now operate recursively in the dimensions and levels of the sparse grid.

The UpDown scheme allows us to compute the stiffness matrix on such a sparse grid. The stiffness matrix represents the influence of each representation function on the L^2 scalar product. For a detailed description see Pflüger's PhD thesis [2]. This formalization was developed as an interdisciplinary project (IDP) at the TU München [1].

Note: This development has two main theories. The correctness of the UpDown scheme, and a verification of an imperative version of it. Both theories can not be merged, as they use different orders on the product type.

Inhaltsverzeichnis

1	Grid Points	2
2	Sparse Grids	6
2.1	Vectors	6
2.2	Inductive enumeration of all grid points	6
2.3	Grid Restricted to a Level	18
2.4	Unbounded Sparse Grid	18
2.5	Sparse Grid	19
2.6	Compute Sparse Grid Points	20
2.7	Base Points	24
2.8	Lift Operation over all Grid Points	31
2.9	Parent Points	32
3	Hat Functions	38

4	UpDown Scheme	47
5	Up Part	48
6	Down part	54
7	UpDown	58
8	Imperative Version	67

1 Grid Points

```
theory Grid-Point
imports HOL-Analysis.Multivariate-Analysis
begin
```

```
type-synonym grid-point = (nat × int) list
```

```
definition lv :: grid-point ⇒ nat ⇒ nat
  where lv p d = fst (p ! d)
```

```
definition ix :: grid-point ⇒ nat ⇒ int
  where ix p d = snd (p ! d)
```

```
definition level :: grid-point ⇒ nat
  where level p = (∑ i < length p. lv p i)
```

```
lemma level-all-eq:
  assumes  $\bigwedge d. d < \text{length } p \implies \text{lv } p \ d = \text{lv } p' \ d$ 
  and  $\text{length } p = \text{length } p'$ 
  shows  $\text{level } p' = \text{level } p$ 
  unfolding level-def using assms by auto
```

```
datatype dir = left | right
```

```
fun sgn :: dir ⇒ int
where
  sgn left = -1
  | sgn right = 1
```

```
fun inv :: dir ⇒ dir
where
  inv left = right
  | inv right = left
```

```
lemma inv-inv[simp]: inv (inv dir) = dir
  by (cases dir) simp-all
```

```
lemma sgn-inv[simp]: sgn (inv dir) = - sgn dir
```

by (cases dir, auto)

definition child :: grid-point \Rightarrow dir \Rightarrow nat \Rightarrow grid-point
where child p dir d = p[d := (lv p d + 1, 2 * (ix p d) + sgn dir)]

lemma child-length[simp]: length (child p dir d) = length p
unfolding child-def by simp

lemma child-odd[simp]: d < length p \implies odd (ix (child p dir d) d)
unfolding child-def ix-def by (cases dir, auto)

lemma child-eq: p ! d = (l, i) \implies \exists j. child p dir d = p[d := (l + 1, j)]
by (auto simp add: child-def ix-def lv-def)

lemma child-other: d \neq d' \implies child p dir d ! d' = p ! d'
unfolding child-def lv-def ix-def by (cases d' < length p, auto)

lemma child-invariant: assumes d' < length p shows (child p dir d ! d' = p ! d')
= (d \neq d')

proof –

obtain l i where p ! d' = (l, i) using prod.exhaust .

with assms show ?thesis

unfolding child-def ix-def lv-def by auto

qed

lemma child-single-level: d < length p \implies lv (child p dir d) d > lv p d
unfolding lv-def child-def by simp

lemma child-lv: d < length p \implies lv (child p dir d) d = lv p d + 1
unfolding child-def lv-def by simp

lemma child-lv-other: assumes d' \neq d shows lv (child p dir d') d = lv p d
using child-other[OF assms] unfolding lv-def by simp

lemma child-ix-left: d < length p \implies ix (child p left d) d = 2 * ix p d - 1
unfolding child-def ix-def by simp

lemma child-ix-right: d < length p \implies ix (child p right d) d = 2 * ix p d + 1
unfolding child-def ix-def by simp

lemma child-ix: d < length p \implies ix (child p dir d) d = 2 * ix p d + sgn dir
unfolding child-def ix-def by simp

lemma child-level[simp]: assumes d < length p
shows level (child p dir d) = level p + 1

proof –

have inter: {0..<length p} \cap {d} = {d} using assms by auto

have level (child p dir d) =

$(\sum d' = 0..<\text{length } p. \text{ if } d' \in \{d\} \text{ then } lv \ p \ d + 1 \text{ else } lv \ p \ d')$
by (*auto intro!*: *sum.cong simp add: child-lv-other child-lv level-def*)
moreover have $\text{level } p + 1 =$
 $(\sum d' = 0..<\text{length } p. \text{ if } d' \in \{d\} \text{ then } lv \ p \ d \text{ else } lv \ p \ d') + 1$
by (*auto intro!*: *sum.cong simp add: child-lv-other child-lv level-def*)
ultimately show *?thesis*
unfolding *sum.If-cases[OF finite-atLeastLessThan]* *inter*
using *assms* **by** *auto*
qed

lemma *child-ex-neighbour*: $\exists b'. \text{child } b \ \text{dir } d = \text{child } b' \ (\text{inv } \text{dir}) \ d$
proof
show $\text{child } b \ \text{dir } d = \text{child } (b[d := (lv \ b \ d, ix \ b \ d + \text{sgn } \text{dir})]) \ (\text{inv } \text{dir}) \ d$
unfolding *child-def ix-def lv-def* **by** (*cases d < length b, auto simp add:*
algebra-simps)
qed

lemma *child-level-gt[simp]*: $\text{level } (\text{child } p \ \text{dir } d) \geq \text{level } p$
by (*cases d < length p, simp, simp add: child-def*)

lemma *child-estimate-child*:
assumes $d < \text{length } p$
and $l \leq lv \ p \ d$
and *i'-range*: $ix \ p \ d < (i + 1) * 2^{\wedge}(lv \ p \ d - l) \wedge$
 $ix \ p \ d > (i - 1) * 2^{\wedge}(lv \ p \ d - l)$
(is *?top p* \wedge *?bottom p***)**
and *is-child*: $p' = \text{child } p \ \text{dir } d$
shows $?top \ p' \wedge ?bottom \ p'$
proof
from *is-child* **and** $\langle d < \text{length } p \rangle$
have $lv \ p' \ d = lv \ p \ d + 1$ **by** (*auto simp add: child-def ix-def lv-def*)
hence $lv \ p' \ d - l = lv \ p \ d - l + 1$ **using** $\langle lv \ p \ d \geq l \rangle$ **by** *auto*
hence *pow-l''*: $(2^{\wedge}(lv \ p' \ d - l) :: \text{int}) = 2 * 2^{\wedge}(lv \ p \ d - l)$ **by** *auto*

show *?top p'*
proof –
from *is-child* **and** $\langle d < \text{length } p \rangle$
have $ix \ p' \ d \leq 2 * (ix \ p \ d) + 1$
by (*cases dir, auto simp add: child-def lv-def ix-def*)
also have $\dots < (i + 1) * (2 * 2^{\wedge}(lv \ p \ d - l))$ **using** *i'-range* **by** *auto*
finally show *?thesis* **using** *pow-l''* **by** *auto*
qed

show *?bottom p'*
proof –
have $(i - 1) * 2^{\wedge}(lv \ p' \ d - l) = 2 * ((i - 1) * 2^{\wedge}(lv \ p \ d - l))$
using *pow-l''* **by** *simp*
also have $\dots < 2 * ix \ p \ d - 1$ **using** *i'-range* **by** *auto*
finally show *?thesis* **using** *is-child* **and** $\langle d < \text{length } p \rangle$

```

    by (cases dir, auto simp add: child-def lv-def ix-def)
  qed
qed

lemma child-neighbour: assumes child p (inv dir) d = child ps dir d (is ?c-p =
?c-ps)
  shows ps = p[d := (lv p d, ix p d - sgn dir)] (is - = ?ps)
proof (rule nth-equalityI)
  have length ?c-ps = length ?c-p using ⟨?c-p = ?c-ps⟩ by simp
  hence len-eq: length ps = length p by simp
  thus length ps = length ?ps by simp

  show ps ! i = ?ps ! i if i < length ps for i
  proof -
    have i < length p
      using that len-eq by auto

    show ps ! i = ?ps ! i
    proof (cases d = i)
      case [simp]: True

        have ?c-p ! i = ?c-ps ! i using ⟨?c-p = ?c-ps⟩ by auto
        hence ix p i = ix ps d + sgn dir and lv p i = lv ps i
          by (auto simp add: child-def
            nth-list-update-eq[OF i < length p]
            nth-list-update-eq[OF i < length ps])
        thus ?thesis by (simp add: lv-def ix-def i < length p)
      next
        assume d ≠ i
        with child-other[OF this, of ps dir] child-other[OF this, of p inv dir]
        show ?thesis using assms by auto
    qed
  qed
qed

definition start :: nat ⇒ grid-point
where
  start dm = replicate dm (0, 1)

lemma start-lv[simp]: d < dm ⇒ lv (start dm) d = 0
  unfolding start-def lv-def by simp

lemma start-ix[simp]: d < dm ⇒ ix (start dm) d = 1
  unfolding start-def ix-def by simp

lemma start-length[simp]: length (start dm) = dm
  unfolding start-def by auto

lemma level-start-0[simp]: level (start dm) = 0

```

using *level-def* by *auto*

end

2 Sparse Grids

theory *Grid*
imports *Grid-Point*
begin

2.1 Vectors

type-synonym *vector* = *grid-point* \Rightarrow *real*

definition *null-vector* :: *vector*
where *null-vector* \equiv $\lambda p. 0$

definition *sum-vector* :: *vector* \Rightarrow *vector* \Rightarrow *vector*
where *sum-vector* $\alpha \beta \equiv \lambda p. \alpha p + \beta p$

2.2 Inductive enumeration of all grid points

inductive-set

grid :: *grid-point* \Rightarrow *nat set* \Rightarrow *grid-point set*
for *b* :: *grid-point* **and** *ds* :: *nat set*

where

Start[*intro!*]: $b \in \text{grid } b \text{ } ds$
| *Child*[*intro!*]: $\llbracket p \in \text{grid } b \text{ } ds ; d \in ds \rrbracket \Longrightarrow \text{child } p \text{ dir } d \in \text{grid } b \text{ } ds$

lemma *grid-length*[*simp*]: $p' \in \text{grid } p \text{ } ds \Longrightarrow \text{length } p' = \text{length } p$
by (*erule grid.induct, auto*)

lemma *grid-union-dims*: $\llbracket ds \subseteq ds' ; p \in \text{grid } b \text{ } ds \rrbracket \Longrightarrow p \in \text{grid } b \text{ } ds'$
by (*erule grid.induct, auto*)

lemma *grid-transitive*: $\llbracket a \in \text{grid } b \text{ } ds ; b \in \text{grid } c \text{ } ds' ; ds' \subseteq ds'' ; ds \subseteq ds'' \rrbracket \Longrightarrow a \in \text{grid } c \text{ } ds''$
by (*erule grid.induct, auto simp add: grid-union-dims*)

lemma *grid-child*[*intro?*]: **assumes** $d \in ds$ **and** *p-grid*: $p \in \text{grid } (\text{child } b \text{ dir } d) \text{ } ds$
shows $p \in \text{grid } b \text{ } ds$
using $\langle d \in ds \rangle$ *grid-transitive*[*OF p-grid*] **by** *auto*

lemma *grid-single-level*[*simp*]: **assumes** $p \in \text{grid } b \text{ } ds$ **and** $d < \text{length } b$
shows $lv \ b \ d \leq lv \ p \ d$
using *assms*

proof *induct*

case (*Child* $p' \ d' \ \text{dir}$)

thus ?*case* **by** (*cases* $d' = d$, *auto simp add: child-def ix-def lv-def*)

qed auto

lemma grid-child-level:

assumes $d < \text{length } b$

and $p\text{-grid}: p \in \text{grid } (child\ b\ dir\ d)\ ds$

shows $lv\ b\ d < lv\ p\ d$

proof -

have $lv\ b\ d < lv\ (child\ b\ dir\ d)\ d$ using $child\text{-}lv[OF\ \langle d < \text{length } b \rangle]$ by auto

also have $\dots \leq lv\ p\ d$ using $p\text{-grid}\ \text{assms}$ by (intro $grid\text{-}single\text{-}level$) auto

finally show ?thesis .

qed

lemma child-out: $\text{length } p < d \implies child\ p\ dir\ d = p$

unfolding $child\text{-}def$ by auto

lemma grid-dim-remove:

assumes $inset: p \in \text{grid } b\ (\{d\} \cup ds)$

and $eq: d < \text{length } b \implies p ! d = b ! d$

shows $p \in \text{grid } b\ ds$

using $inset\ eq$

proof induct

case (Child $p'\ d'\ dir$)

show ?case

proof (cases $d' \geq \text{length } p'$)

case True with $child\text{-}out[OF\ this]$

show ?thesis using Child by auto

next

case False hence $d' < \text{length } p'$ by simp

show ?thesis

proof (cases $d' = d$)

case True

hence $lv\ b\ d \leq lv\ p'\ d$ and $lv\ p'\ d < lv\ (child\ p'\ dir\ d)\ d$

using $child\text{-}single\text{-}level\ Child\ \langle d' < \text{length } p' \rangle$ by auto

hence False using Child and $\langle d' = d \rangle$ and $lv\text{-}def$ and $\langle \neg d' \geq \text{length } p' \rangle$ by

auto

thus ?thesis ..

next

case False

hence $d' \in ds$ using Child by auto

moreover have $d < \text{length } b \implies p' ! d = b ! d$

proof -

assume $d < \text{length } b$

hence $d < \text{length } p'$ using Child by auto

hence $child\ p'\ dir\ d' ! d = p' ! d$ using $child\text{-}invariant$ and False by auto

thus ?thesis using Child and $\langle d < \text{length } b \rangle$ by auto

qed

hence $p' \in \text{grid } b\ ds$ using Child by auto

ultimately show ?thesis using $grid.Child$ by auto

qed

qed
qed auto

lemma *gridgen-dim-restrict*:
 assumes *inset*: $p \in \text{grid } b \ (ds' \cup ds)$
 and *eq*: $\forall d \in ds'. d \geq \text{length } b$
 shows $p \in \text{grid } b \ ds$
 using *inset eq*
 proof induct
 case (*Child p' d dir*)
 thus ?*case*
 proof (cases $d \in ds$)
 case *False* thus ?*thesis* using *Child and child-def* by auto
 qed auto
 qed auto

lemma *grid-dim-remove-outer*: $\text{grid } b \ ds = \text{grid } b \ \{d \in ds. d < \text{length } b\}$
 proof
 have $\{d \in ds. d < \text{length } b\} \subseteq ds$ by auto
 from *grid-union-dims*[*OF this*]
 show $\text{grid } b \ \{d \in ds. d < \text{length } b\} \subseteq \text{grid } b \ ds$ by auto

 have $ds = (ds - \{d \in ds. d < \text{length } b\}) \cup \{d \in ds. d < \text{length } b\}$ by auto
 moreover
 have $\text{grid } b \ ((ds - \{d \in ds. d < \text{length } b\}) \cup \{d \in ds. d < \text{length } b\}) \subseteq \text{grid } b \ \{d \in ds. d < \text{length } b\}$
 proof
 fix *p*
 assume $p \in \text{grid } b \ (ds - \{d \in ds. d < \text{length } b\}) \cup \{d \in ds. d < \text{length } b\}$
 moreover have $\forall d \in (ds - \{d \in ds. d < \text{length } b\}). d \geq \text{length } b$ by auto
 ultimately show $p \in \text{grid } b \ \{d \in ds. d < \text{length } b\}$ by (*rule gridgen-dim-restrict*)
 qed
 ultimately show $\text{grid } b \ ds \subseteq \text{grid } b \ \{d \in ds. d < \text{length } b\}$ by auto
 qed

lemma *grid-level*[*intro*]: assumes $p \in \text{grid } b \ ds$ shows $\text{level } b \leq \text{level } p$
 proof –
 have $*$: $\text{length } p = \text{length } b$ using *grid-length* *assms* by auto
 { fix *i* assume $i \in \{0 .. < \text{length } p\}$
 hence $\text{lv } b \ i \leq \text{lv } p \ i$ using $\langle p \in \text{grid } b \ ds \rangle$ and *grid-single-level ** by auto
 } thus ?*thesis* unfolding *level-def ** by (*auto intro!*: *sum-mono*)
 qed

lemma *grid-empty-ds*[*simp*]: $\text{grid } b \ \{\} = \{ b \}$
 proof –
 have !! *z*. $z \in \text{grid } b \ \{\} \implies z = b$
 by (*erule grid.induct, auto*)
 thus ?*thesis* by auto
 qed

lemma *grid-Start*: assumes *inset*: $p \in \text{grid } b \ ds$ and *eq*: $\text{level } p = \text{level } b$ shows

$p = b$
using *inset eq*
proof induct
case (*Child p d dir*)
show *?case*
proof (*cases d < length b*)
case *True*
from *Child*
have $\text{level } p \geq \text{level } b$ **by** *auto*
moreover
have $\text{level } p \leq \text{level } (\text{child } p \text{ dir } d)$ **by** (*rule child-level-gt*)
hence $\text{level } p \leq \text{level } b$ **using** *Child* **by** *auto*
ultimately have $\text{level } p = \text{level } b$ **by** *auto*
hence $p = b$ **using** *Child(2)* **by** *auto*
with *Child(4)* **have** $\text{level } (\text{child } b \text{ dir } d) = \text{level } b$ **by** *auto*
moreover have $\text{level } (\text{child } b \text{ dir } d) \neq \text{level } b$ **using** *child-level* **and** $\langle d <$
length b **by** *auto*
ultimately show *?thesis* **by** *auto*
next
case *False*
with *Child* **have** $\text{length } p = \text{length } b$ **by** *auto*
with *False* **have** $\text{child } p \text{ dir } d = p$ **using** *child-def* **by** *auto*
moreover with *Child* **have** $\text{level } p = \text{level } b$ **by** *auto*
with *Child(2)* **have** $p = b$ **by** *auto*
ultimately show *?thesis* **by** *auto*
qed
qed auto
lemma grid-estimate:
assumes $d < \text{length } b$ **and** *p-grid: p ∈ grid b ds*
shows $\text{ix } p \text{ d} < (\text{ix } b \text{ d} + 1) * 2^{\lceil \text{lv } p \text{ d} - \text{lv } b \text{ d} \rceil} \wedge \text{ix } p \text{ d} > (\text{ix } b \text{ d} - 1) * 2^{\lfloor \text{lv } p \text{ d} - \text{lv } b \text{ d} \rfloor}$
using *p-grid*
proof induct
case (*Child p d' dir*)
show *?case*
proof (*cases d = d'*)
case *False* **with** *Child* **show** *?thesis unfolding child-def lv-def ix-def* **by** *auto*
next
case *True* **with** *child-estimate-child* **and** *Child* **and** $\langle d < \text{length } b \rangle$
show *?thesis using grid-single-level* **by** *auto*
qed
qed auto
lemma grid-odd: assumes $d < \text{length } b$ **and** *p-diff: p ! d ≠ b ! d* **and** *p-grid: p*
 $\in \text{grid } b \text{ ds}$
shows *odd (ix p d)*
using *p-grid* **and** *p-diff*
proof induct
case (*Child p d' dir*)
show *?case*

proof (cases $d = d'$)
case *True* **with** *child-odd* **and** $\langle d < \text{length } b \rangle$ **and** *Child* **show** *?thesis* **by** *auto*
next
case *False* **with** *Child* **and** $\langle d < \text{length } b \rangle$ **show** *?thesis* **using** *child-def* **and**
ix-def **and** *lv-def* **by** *auto*
qed
qed *auto*
lemma *grid-invariant*: **assumes** $d < \text{length } b$ **and** $d \notin ds$ **and** *p-grid*: $p \in \text{grid } b$
 ds
shows $p ! d = b ! d$
using *p-grid*
proof (*induct*)
case (*Child* $p \ d' \ \text{dir}$) **hence** $d' \neq d$ **using** $\langle d \notin ds \rangle$ **by** *auto*
thus *?case* **using** *child-def* **and** *Child* **by** *auto*
qed *auto*
lemma *grid-part*: **assumes** $d < \text{length } b$ **and** *p-valid*: $p \in \text{grid } b \ \{d\}$ **and** *p'-valid*:
 $p' \in \text{grid } b \ \{d\}$
and *level*: $lv \ p' \ d \geq lv \ p \ d$
and *right*: $ix \ p' \ d \leq (ix \ p \ d + 1) * 2^{\wedge}(lv \ p' \ d - lv \ p \ d)$ (**is** *?right* $p \ p' \ d$)
and *left*: $ix \ p' \ d \geq (ix \ p \ d - 1) * 2^{\wedge}(lv \ p' \ d - lv \ p \ d)$ (**is** *?left* $p \ p' \ d$)
shows $p' \in \text{grid } p \ \{d\}$
using *p'-valid* *left* *right* *level* **and** *p-valid*
proof (*induct*)
case (*Child* $p' \ d' \ \text{dir}$)
hence $d = d'$ **by** *auto*
let *?child* = *child* $p' \ \text{dir} \ d'$

show *?case*
proof (cases $lv \ p \ d = lv \ ?child \ d$)
case *False*
moreover **have** $lv \ ?child \ d = lv \ p' \ d + 1$ **using** *child-lv* **and** $\langle d < \text{length } b \rangle$
and *Child* **and** $\langle d = d' \rangle$ **by** *auto*
ultimately **have** $lv \ p \ d < lv \ p' \ d + 1$ **using** *Child* **by** *auto*
hence *lv*: $\text{Suc} \ (lv \ p' \ d) - lv \ p \ d = \text{Suc} \ (lv \ p' \ d - lv \ p \ d)$ **by** *auto*

have $?left \ p \ p' \ d \wedge ?right \ p \ p' \ d$
proof (cases *dir*)
case *left*
with *Child* **have** $2 * ix \ p' \ d - 1 \leq (ix \ p \ d + 1) * 2^{\wedge}(\text{Suc} \ (lv \ p' \ d) - lv \ p \ d)$
using $\langle d = d' \rangle$ **and** $\langle d < \text{length } b \rangle$ **by** (*auto simp add: child-def ix-def lv-def*)
also **have** $\dots = 2 * (ix \ p \ d + 1) * 2^{\wedge}(lv \ p' \ d - lv \ p \ d)$ **using** *lv* **by** *auto*
finally **have** $2 * ix \ p' \ d - 2 < 2 * (ix \ p \ d + 1) * 2^{\wedge}(lv \ p' \ d - lv \ p \ d)$ **by** *auto*
also **have** $\dots = 2 * ((ix \ p \ d + 1) * 2^{\wedge}(lv \ p' \ d - lv \ p \ d))$ **by** *auto*
finally **have** *left-r*: $ix \ p' \ d \leq (ix \ p \ d + 1) * 2^{\wedge}(lv \ p' \ d - lv \ p \ d)$ **by** *auto*

have $2 * ((ix \ p \ d - 1) * 2^{\wedge}(lv \ p' \ d - lv \ p \ d)) = 2 * (ix \ p \ d - 1) * 2^{\wedge}(lv \ p' \ d$
 $- lv \ p \ d)$ **by** *auto*
also **have** $\dots = (ix \ p \ d - 1) * 2^{\wedge}(\text{Suc} \ (lv \ p' \ d) - lv \ p \ d)$ **using** *lv* **by** *auto*
also **have** $\dots \leq 2 * ix \ p' \ d - 1$

using *left* **and** *Child* **and** $\langle d = d' \rangle$ **and** $\langle d < \text{length } b \rangle$ **by** (*auto simp add: child-def ix-def lv-def*)
finally have *right-r*: $((ix\ p\ d - 1) * 2^{\wedge}(lv\ p'\ d - lv\ p\ d)) \leq ix\ p'\ d$ **by** *auto*

show *?thesis* **using** *left-r* **and** *right-r* **by** *auto*

next
case *right*
with *Child* **have** $2 * ix\ p'\ d + 1 \leq (ix\ p\ d + 1) * 2^{\wedge}(Suc\ (lv\ p'\ d) - lv\ p\ d)$
using $\langle d = d' \rangle$ **and** $\langle d < \text{length } b \rangle$ **by** (*auto simp add: child-def ix-def lv-def*)
also have $\dots = 2 * (ix\ p\ d + 1) * 2^{\wedge}(lv\ p'\ d - lv\ p\ d)$ **using** *lv* **by** *auto*
finally have $2 * ix\ p'\ d < 2 * (ix\ p\ d + 1) * 2^{\wedge}(lv\ p'\ d - lv\ p\ d)$ **by** *auto*
also have $\dots = 2 * ((ix\ p\ d + 1) * 2^{\wedge}(lv\ p'\ d - lv\ p\ d))$ **by** *auto*
finally have *left-r*: $ix\ p'\ d \leq (ix\ p\ d + 1) * 2^{\wedge}(lv\ p'\ d - lv\ p\ d)$ **by** *auto*

have $2 * ((ix\ p\ d - 1) * 2^{\wedge}(lv\ p'\ d - lv\ p\ d)) = 2 * (ix\ p\ d - 1) * 2^{\wedge}(lv\ p'\ d - lv\ p\ d)$ **by** *auto*
also have $\dots = (ix\ p\ d - 1) * 2^{\wedge}(Suc\ (lv\ p'\ d) - lv\ p\ d)$ **using** *lv* **by** *auto*
also have $\dots \leq 2 * ix\ p'\ d + 1$
using *right* **and** *Child* **and** $\langle d = d' \rangle$ **and** $\langle d < \text{length } b \rangle$ **by** (*auto simp add: child-def ix-def lv-def*)
also have $\dots < 2 * (ix\ p'\ d + 1)$ **by** *auto*
finally have *right-r*: $((ix\ p\ d - 1) * 2^{\wedge}(lv\ p'\ d - lv\ p\ d)) \leq ix\ p'\ d$ **by** *auto*

show *?thesis* **using** *left-r* **and** *right-r* **by** *auto*

qed
with *Child* **and** *lv* **have** $p' \in \text{grid } p\ \{d\}$ **by** *auto*
thus *?thesis* **using** $\langle d = d' \rangle$ **by** *auto*

next
case *True*
moreover with *Child* **have** $?left\ p\ ?child\ d \wedge ?right\ p\ ?child\ d$ **by** *auto*
ultimately have *range*: $ix\ p\ d - 1 \leq ix\ ?child\ d \wedge ix\ ?child\ d \leq ix\ p\ d + 1$ **by** *auto*

have $p!\ d \neq b!\ d$
proof (*rule ccontr*)
assume $\neg (p!\ d \neq b!\ d)$
with $\langle lv\ p\ d = lv\ ?child\ d \rangle$ **have** $lv\ b\ d = lv\ ?child\ d$ **by** (*auto simp add: lv-def*)
hence $lv\ b\ d = lv\ p'\ d + 1$ **using** $\langle d = d' \rangle$ **and** *Child* **and** $\langle d < \text{length } b \rangle$ **and** *child-lv* **by** *auto*
moreover have $lv\ b\ d \leq lv\ p'\ d$ **using** $\langle d = d' \rangle$ **and** *Child* **and** $\langle d < \text{length } b \rangle$ **and** *grid-single-level* **by** *auto*
ultimately show *False* **by** *auto*

qed
hence *odd* $(ix\ p\ d)$ **using** *grid-odd* **and** $\langle p \in \text{grid } b\ \{d\} \rangle$ **and** $\langle d < \text{length } b \rangle$ **by** *auto*
hence $\neg \text{odd}\ (ix\ p\ d + 1)$ **and** $\neg \text{odd}\ (ix\ p\ d - 1)$ **by** *auto*

have $d < \text{length } p'$ **using** $\langle p' \in \text{grid } b\ \{d\} \rangle$ **and** $\langle d < \text{length } b \rangle$ **by** *auto*

hence odd-child: odd (ix ?child d) using child-odd and $\langle d = d' \rangle$ by auto

have $ix\ p\ d - 1 \neq ix\ ?child\ d$
proof (rule ccontr)
assume $\neg (ix\ p\ d - 1 \neq ix\ ?child\ d)$
hence odd (ix p d - 1) using odd-child by auto
thus False using $\langle \neg\ odd\ (ix\ p\ d - 1) \rangle$ by auto
qed

moreover
have $ix\ p\ d + 1 \neq ix\ ?child\ d$
proof (rule ccontr)
assume $\neg (ix\ p\ d + 1 \neq ix\ ?child\ d)$
hence odd (ix p d + 1) using odd-child by auto
thus False using $\langle \neg\ odd\ (ix\ p\ d + 1) \rangle$ by auto
qed

ultimately have $ix\ p\ d = ix\ ?child\ d$ using range by auto
with True have d-eq: $p ! d = (?child) ! d$ by (auto simp add: prod-eqI ix-def lv-def)

have $length\ p = length\ ?child$ using $\langle p \in grid\ b\ \{d\} \rangle$ and $\langle p' \in grid\ b\ \{d\} \rangle$ by auto

moreover have $p ! d'' = ?child ! d''$ if $d'' < length\ p$ for d''
proof -
have $d'' < length\ b$ using that $\langle p \in grid\ b\ \{d\} \rangle$ by auto
show $p ! d'' = ?child ! d''$
proof (cases $d = d''$)
case True with d-eq show ?thesis by auto
next
case False hence $d'' \notin \{d\}$ by auto
from $\langle d'' < length\ b \rangle$ and this and $\langle p \in grid\ b\ \{d\} \rangle$
have $p ! d'' = b ! d''$ by (rule grid-invariant)
also have $\dots = p' ! d''$ using $\langle d'' < length\ b \rangle$ and $\langle d'' \notin \{d\} \rangle$ and $\langle p' \in grid\ b\ \{d\} \rangle$
by (rule grid-invariant[symmetric])
also have $\dots = ?child ! d''$
proof -
have $d'' < length\ p'$ using $\langle d'' < length\ b \rangle$ and $\langle p' \in grid\ b\ \{d\} \rangle$ by auto
hence $?child ! d'' = p' ! d''$ using child-invariant and $\langle d \neq d'' \rangle$ and $\langle d = d' \rangle$ by auto
thus ?thesis by auto
qed
finally show ?thesis .
qed

qed
ultimately have $p = ?child$ by (rule nth-equalityI)
thus $?child \in grid\ p\ \{d\}$ by auto
qed

next
case Start

moreover hence $lv\ b\ d \leq lv\ p\ d$ using *grid-single-level* and $\langle d < length\ b \rangle$ by *auto*

ultimately have $lv\ b\ d = lv\ p\ d$ by *auto*

have $level\ p = level\ b$

proof –

{ fix d'

assume $d' < length\ b$

have $lv\ b\ d' = lv\ p\ d'$

proof (cases $d = d'$)

case *True* with $\langle lv\ b\ d = lv\ p\ d \rangle$ show *?thesis* by *auto*

next

case *False* hence $d' \notin \{d\}$ by *auto*

from $\langle d' < length\ b \rangle$ and *this* and $\langle p \in grid\ b\ \{d\} \rangle$

have $p \neq d' = b \neq d'$ by (rule *grid-invariant*)

thus *?thesis* by (*auto simp add: lv-def*)

qed }

moreover have $length\ b = length\ p$ using $\langle p \in grid\ b\ \{d\} \rangle$ by *auto*

ultimately show *?thesis* by (rule *level-all-eq*)

qed

hence $p = b$ using *grid-Start* and $\langle p \in grid\ b\ \{d\} \rangle$ by *auto*

thus *?case* by *auto*

qed

lemma *grid-disjunct*: assumes $d < length\ p$

shows $grid\ (child\ p\ left\ d)\ ds \cap grid\ (child\ p\ right\ d)\ ds = \{\}$

(is $grid\ ?l\ ds \cap grid\ ?r\ ds = \{\}$)

proof (intro *set-eqI iffI*)

fix x

assume $x \in grid\ ?l\ ds \cap grid\ ?r\ ds$

hence $ix\ x\ d < (ix\ ?l\ d + 1) * 2^{(lv\ x\ d - lv\ ?l\ d)}$

and $ix\ x\ d > (ix\ ?r\ d - 1) * 2^{(lv\ x\ d - lv\ ?r\ d)}$

using *grid-estimate* $\langle d < length\ p \rangle$ by *auto*

thus $x \in \{\}$ using $\langle d < length\ p \rangle$ and *child-lv* and *child-ix* by *auto*

qed *auto*

lemma *grid-level-eq*: assumes *eq*: $\forall d \in ds. lv\ p\ d = lv\ b\ d$ and *grid*: $p \in grid\ b\ ds$

shows $level\ p = level\ b$

proof (rule *level-all-eq*)

{ fix i assume $i < length\ b$

show $lv\ b\ i = lv\ p\ i$

proof (cases $i \in ds$)

case *True* with *eq* show *?thesis* by *auto*

next case *False* with $\langle i < length\ b \rangle$ and *grid* show *?thesis*

using *lv-def ix-def grid-invariant* by *auto*

qed }

show $length\ b = length\ p$ using *grid* by *auto*

qed

lemma *grid-partition*:

$grid\ p\ \{d\} = \{p\} \cup grid\ (child\ p\ left\ d)\ \{d\} \cup grid\ (child\ p\ right\ d)\ \{d\}$

(**is** $- = - \cup grid\ ?l\ \{d\} \cup grid\ ?r\ \{d\}$)

proof –

have $!!\ x.\ \llbracket\ x \in grid\ p\ \{d\} ; x \neq p ; x \notin grid\ ?r\ \{d\} \rrbracket \implies x \in grid\ ?l\ \{d\}$

proof (*cases* $d < length\ p$)

case *True*

fix x

let $?nr-r\ p = ix\ x\ d > (ix\ p\ d + 1) * 2^\wedge (lv\ x\ d - lv\ p\ d)$

let $?nr-l\ p = (ix\ p\ d - 1) * 2^\wedge (lv\ x\ d - lv\ p\ d) > ix\ x\ d$

have *ix-r-eq*: $ix\ ?r\ d = 2 * ix\ p\ d + 1$ **using** $\langle d < length\ p \rangle$ **and** *child-ix* **by** *auto*

have *lv-r-eq*: $lv\ ?r\ d = lv\ p\ d + 1$ **using** $\langle d < length\ p \rangle$ **and** *child-lv* **by** *auto*

have *ix-l-eq*: $ix\ ?l\ d = 2 * ix\ p\ d - 1$ **using** $\langle d < length\ p \rangle$ **and** *child-ix* **by** *auto*

have *lv-l-eq*: $lv\ ?l\ d = lv\ p\ d + 1$ **using** $\langle d < length\ p \rangle$ **and** *child-lv* **by** *auto*

assume $x \in grid\ p\ \{d\}$ **and** $x \neq p$ **and** $x \notin grid\ ?r\ \{d\}$

hence $lv\ p\ d \leq lv\ x\ d$ **using** *grid-single-level* **and** $\langle d < length\ p \rangle$ **by** *auto*

moreover **have** $lv\ p\ d \neq lv\ x\ d$

proof (*rule* *ccontr*)

assume $\neg lv\ p\ d \neq lv\ x\ d$

hence *level* $x = level\ p$ **using** $\langle x \in grid\ p\ \{d\} \rangle$ **and** *grid-level-eq*[*where* $ds=\{d\}$] **by** *auto*

hence $x = p$ **using** *grid-Start* **and** $\langle x \in grid\ p\ \{d\} \rangle$ **by** *auto*

thus *False* **using** $\langle x \neq p \rangle$ **by** *auto*

qed

ultimately **have** $lv\ p\ d < lv\ x\ d$ **by** *auto*

hence $lv\ ?r\ d \leq lv\ x\ d$ **and** $?r \in grid\ p\ \{d\}$ **using** *child-lv* **and** $\langle d < length\ p \rangle$ **by** *auto*

with $\langle d < length\ p \rangle$ **and** $\langle x \in grid\ p\ \{d\} \rangle$

have *r-range*: $\neg ?nr-r\ ?r \wedge \neg ?nr-l\ ?r \implies x \in grid\ ?r\ \{d\}$

using *grid-part*[*where* $p=?r$ **and** $p'=x$ **and** $b=p$ **and** $d=d$] **by** *auto*

have $x \notin grid\ ?r\ \{d\} \implies ?nr-l\ ?r \vee ?nr-r\ ?r$ **by** (*rule* *ccontr*, *auto* *simp* *add*: *r-range*)

hence $?nr-l\ ?r \vee ?nr-r\ ?r$ **using** $\langle x \notin grid\ ?r\ \{d\} \rangle$ **by** *auto*

have *gt0*: $lv\ x\ d - lv\ p\ d > 0$ **using** $\langle lv\ p\ d < lv\ x\ d \rangle$ **by** *auto*

have *ix-shift*: $ix\ ?r\ d = ix\ ?l\ d + 2$ **and** *lv-lr*: $lv\ ?r\ d = lv\ ?l\ d$ **and** *right1*: $!!\ x$
 $::\ int.\ x + 2 - 1 = x + 1$

using $\langle d < length\ p \rangle$ **and** *child-ix* **and** *child-lv* **by** *auto*

have $lv\ x\ d - lv\ p\ d = Suc\ (lv\ x\ d - (lv\ p\ d + 1))$

using *gt0* **by** *auto*

hence *lv-shift*: $!!\ y :: int.\ y * 2^\wedge (lv\ x\ d - lv\ p\ d) = y * 2 * 2^\wedge (lv\ x\ d - (lv$

$p\ d + 1)$
by auto

have $ix\ x\ d < (ix\ p\ d + 1) * 2^{\wedge}(lv\ x\ d - lv\ p\ d)$
using $\langle x \in grid\ p\ \{d\}\rangle$ *grid-estimate* **and** $\langle d < length\ p\rangle$ **by auto**
also have $\dots = (ix\ ?r\ d + 1) * 2^{\wedge}(lv\ x\ d - lv\ ?r\ d)$
using $\langle lv\ p\ d < lv\ x\ d\rangle$ **and** *ix-r-eq* **and** *lv-r-eq* *lv-shift*[**where** $y=ix\ p\ d + 1$]
by auto
finally have $?nr-l\ ?r$ **using** $\langle ?nr-l\ ?r \vee ?nr-r\ ?r\rangle$ **by auto**
hence *r-bound*: $(ix\ ?l\ d + 1) * 2^{\wedge}(lv\ x\ d - lv\ ?l\ d) > ix\ x\ d$
unfolding *ix-shift* *lv-lr* **using** *right1* **by auto**

have $(ix\ ?l\ d - 1) * 2^{\wedge}(lv\ x\ d - lv\ ?l\ d) = (ix\ p\ d - 1) * 2 * 2^{\wedge}(lv\ x\ d - (lv\ p\ d + 1))$
unfolding *ix-l-eq* *lv-l-eq* **by auto**
also have $\dots = (ix\ p\ d - 1) * 2^{\wedge}(lv\ x\ d - lv\ p\ d)$
using *lv-shift*[**where** $y=ix\ p\ d - 1$] **by auto**
also have $\dots < ix\ x\ d$
using $\langle x \in grid\ p\ \{d\}\rangle$ *grid-estimate* **and** $\langle d < length\ p\rangle$ **by auto**
finally have *l-bound*: $(ix\ ?l\ d - 1) * 2^{\wedge}(lv\ x\ d - lv\ ?l\ d) < ix\ x\ d$.

from *l-bound* *r-bound* $\langle d < length\ p\rangle$ **and** $\langle x \in grid\ p\ \{d\}\rangle$ $\langle lv\ ?r\ d \leq lv\ x\ d\rangle$
and *lv-lr*
show $x \in grid\ ?l\ \{d\}$ **using** *grid-part*[**where** $p=?l$ **and** $p'=x$ **and** $d=d$] **by auto**
qed (*auto simp add: child-def*)
thus *?thesis* **by** (*auto intro: grid-child*)
qed

lemma *grid-change-dim*: **assumes** *grid*: $p \in grid\ b\ ds$
shows $p[d := X] \in grid\ (b[d := X])\ ds$
using *grid*
proof *induct*
case (*Child* $p\ d'\ dir$)
show *?case*
proof (*cases* $d \neq d'$)
case *True*
have $(child\ p\ dir\ d')[d := X] = child\ (p[d := X])\ dir\ d'$
unfolding *child-def* **and** *ix-def* **and** *lv-def*
unfolding *list-update-swap*[*OF* $\langle d \neq d'\rangle$] **and** *nth-list-update-neq*[*OF* $\langle d \neq d'\rangle$]
..
thus *?thesis* **using** *Child* **by auto**
next
case *False* **hence** $d = d'$ **by auto**
with *Child* **show** *?thesis* **unfolding** *child-def* $\langle d = d'\rangle$ *list-update-overwrite* **by auto**
qed
qed *auto*

lemma *grid-change-dim-child*: **assumes** *grid*: $p \in grid\ b\ ds$ **and** $d \notin ds$
shows $child\ p\ dir\ d \in grid\ (child\ b\ dir\ d)\ ds$

proof (*cases* $d < \text{length } b$)
case *True* **thus** *?thesis* **using** *grid-change-dim*[*OF* *grid*]
unfolding *child-def lv-def ix-def grid-invariant*[*OF* *True* $\langle d \notin ds \rangle$ *grid*] **by** *auto*
next
case *False* **hence** $\text{length } b \leq d$ **and** $\text{length } p \leq d$ **using** *grid* **by** *auto*
thus *?thesis* **unfolding** *child-def* **using** *list-update-beyond* *assms* **by** *auto*
qed
lemma *grid-split*: **assumes** *grid*: $p \in \text{grid } b \ (ds' \cup ds)$ **shows** $\exists x \in \text{grid } b \ ds. p \in \text{grid } x \ ds'$
using *grid*
proof *induct*
case (*Child* $p \ d \ \text{dir}$)
show *?case*
proof (*cases* $d \in ds'$)
case *True* **with** *Child* **show** *?thesis* **by** *auto*
next
case *False*
hence $d \in ds$ **using** *Child* **by** *auto*
obtain x **where** $x \in \text{grid } b \ ds$ **and** $p \in \text{grid } x \ ds'$ **using** *Child* **by** *auto*
hence *child* $x \ \text{dir } d \in \text{grid } b \ ds$ **using** $\langle d \in ds \rangle$ **by** *auto*
moreover **have** *child* $p \ \text{dir } d \in \text{grid } (\text{child } x \ \text{dir } d) \ ds'$
using $\langle p \in \text{grid } x \ ds' \rangle$ *False* **and** *grid-change-dim-child* **by** *auto*
ultimately **show** *?thesis* **by** *auto*
qed
qed *auto*
lemma *grid-union-eq*: $(\bigcup p \in \text{grid } b \ ds. \text{grid } p \ ds') = \text{grid } b \ (ds' \cup ds)$
using *grid-split* **and** *grid-transitive*[**where** $ds'' = ds' \cup ds$ **and** $ds = ds'$ **and** $ds' = ds$, *OF* - - *Un-upper2 Un-upper1*] **by** *auto*
lemma *grid-onedim-split*:
 $\text{grid } b \ (ds \cup \{d\}) = \text{grid } b \ ds \cup \text{grid } (\text{child } b \ \text{left } d) \ (ds \cup \{d\}) \cup \text{grid } (\text{child } b \ \text{right } d) \ (ds \cup \{d\})$
 $(\text{is } - = ?g \cup ?l \ (ds \cup \{d\}) \cup ?r \ (ds \cup \{d\}))$
proof -
have $?g \cup ?l \ (ds \cup \{d\}) \cup ?r \ (ds \cup \{d\}) = ?g \cup (\bigcup p \in ?l \ \{d\}. \text{grid } p \ ds) \cup (\bigcup p \in ?r \ \{d\}. \text{grid } p \ ds)$
unfolding *grid-union-eq* ..
also **have** $\dots = (\bigcup p \in (\{b\} \cup ?l \ \{d\} \cup ?r \ \{d\}). \text{grid } p \ ds)$ **by** *auto*
also **have** $\dots = (\bigcup p \in \text{grid } b \ \{d\}. \text{grid } p \ ds)$ **unfolding** *grid-partition*[**where** $p=b$] ..
finally **show** *?thesis* **unfolding** *grid-union-eq* **by** *auto*
qed
lemma *grid-child-without-parent*: **assumes** *grid*: $p \in \text{grid } (\text{child } b \ \text{dir } d) \ ds$ (**is** $p \in \text{grid } ?c \ ds$) **and** $d < \text{length } b$
shows $p \neq b$
proof -
have $\text{level } ?c \leq \text{level } p$ **using** *grid* **by** (*rule* *grid-level*)
hence $\text{level } b < \text{level } p$ **using** *child-level* **and** $\langle d < \text{length } b \rangle$ **by** *auto*
thus *?thesis* **by** *auto*
qed

lemma *grid-disjunct'*:
assumes $p \in \text{grid } b \text{ } ds$ **and** $p' \in \text{grid } b \text{ } ds$ **and** $x \in \text{grid } p \text{ } ds'$ **and** $p \neq p'$ **and**
 $ds \cap ds' = \{\}$
shows $x \notin \text{grid } p' \text{ } ds'$
proof (*rule ccontr*)
assume $\neg x \notin \text{grid } p' \text{ } ds'$ **hence** $x \in \text{grid } p' \text{ } ds'$ **by** *auto*
have $l: \text{length } b = \text{length } p$ **and** $l': \text{length } b = \text{length } p'$ **using** $\langle p \in \text{grid } b \text{ } ds \rangle$
and $\langle p' \in \text{grid } b \text{ } ds \rangle$ **by** *auto*
hence $\text{length } p' = \text{length } p$ **by** *auto*
moreover have $\forall d < \text{length } p'. p' ! d = p ! d$
proof (*rule allI, rule impI*)
fix d **assume** $dl': d < \text{length } p'$ **hence** $d < \text{length } b$ **using** l' **by** *auto*
hence $dl: d < \text{length } p$ **using** l **by** *auto*
show $p' ! d = p ! d$
proof (*cases d \in ds'*)
case *True* **with** $\langle ds \cap ds' = \{\} \rangle$ **have** $d \notin ds$ **by** *auto*
hence $p' ! d = b ! d$ **and** $p ! d = b ! d$
using $\langle d < \text{length } b \rangle$ $\langle p' \in \text{grid } b \text{ } ds \rangle$ **and** $\langle p \in \text{grid } b \text{ } ds \rangle$ **and** *grid-invariant*
by *auto*
thus *?thesis* **by** *auto*
next
case *False*
show *?thesis*
using *grid-invariant*[*OF dl' False (x \in grid p' ds')*]
and *grid-invariant*[*OF dl False (x \in grid p ds')*] **by** *auto*
qed
qed
ultimately have $p' = p$ **by** (*metis nth-equalityI*)
thus *False* **using** $\langle p \neq p' \rangle$ **by** *auto*
qed

lemma *grid-split1*: **assumes** *grid*: $p \in \text{grid } b \text{ } (ds' \cup ds)$ **and** $ds \cap ds' = \{\}$
shows $\exists! x \in \text{grid } b \text{ } ds. p \in \text{grid } x \text{ } ds'$
proof (*rule ex-ex1I*)
obtain x **where** $x \in \text{grid } b \text{ } ds$ **and** $p \in \text{grid } x \text{ } ds'$ **using** *grid-split*[*OF grid*] **by**
auto
thus $\exists x. x \in \text{grid } b \text{ } ds \wedge p \in \text{grid } x \text{ } ds'$ **by** *auto*
next
fix $x \ y$
assume $x \in \text{grid } b \text{ } ds \wedge p \in \text{grid } x \text{ } ds'$ **and** $y \in \text{grid } b \text{ } ds \wedge p \in \text{grid } y \text{ } ds'$
hence $x \in \text{grid } b \text{ } ds$ **and** $p \in \text{grid } x \text{ } ds'$ **and** $y \in \text{grid } b \text{ } ds$ **and** $p \in \text{grid } y \text{ } ds'$ **by**
auto
show $x = y$
proof (*rule ccontr*)
assume $x \neq y$
from *grid-disjunct'*[*OF (x \in grid b ds) (y \in grid b ds) (p \in grid x ds')*] **this** $\langle ds$
 $\cap ds' = \{\} \rangle$
show *False* **using** $\langle p \in \text{grid } y \text{ } ds' \rangle$ **by** *auto*
qed
qed

2.3 Grid Restricted to a Level

definition *lgrid* :: *grid-point* \Rightarrow *nat set* \Rightarrow *nat* \Rightarrow *grid-point set*
where *lgrid* *b ds lm* = { *p* \in *grid* *b ds*. *level* *p* < *lm* }

lemma *lgridI*[*intro*]:
 [*p* \in *grid* *b ds* ; *level* *p* < *lm*] \Longrightarrow *p* \in *lgrid* *b ds lm*
unfolding *lgrid-def* **by** *simp*

lemma *lgridE*[*elim*]:
assumes *p* \in *lgrid* *b ds lm*
assumes [*p* \in *grid* *b ds* ; *level* *p* < *lm*] \Longrightarrow *P*
shows *P*
using *assms* **unfolding** *lgrid-def* **by** *auto*

lemma *lgridI-child*[*intro*]:
d \in *ds* \Longrightarrow *p* \in *lgrid* (*child* *b dir d*) *ds lm* \Longrightarrow *p* \in *lgrid* *b ds lm*
by (*auto intro: grid-child*)

lemma *lgrid-empty*[*simp*]: *lgrid* *p ds* (*level* *p*) = {}
proof (*rule equals0I*)
fix *p'* **assume** *p'* \in *lgrid* *p ds* (*level* *p*)
hence *level* *p'* < *level* *p* **and** *level* *p* \leq *level* *p'* **by** *auto*
thus *False* **by** *auto*
qed

lemma *lgrid-empty'*: **assumes** *lm* \leq *level* *p* **shows** *lgrid* *p ds lm* = {}
proof (*rule equals0I*)
fix *p'* **assume** *p'* \in *lgrid* *p ds lm*
hence *level* *p'* < *lm* **and** *level* *p* \leq *level* *p'* **by** *auto*
thus *False* **using** (<*lm* \leq *level* *p*) **by** *auto*
qed

lemma *grid-not-child*:
assumes [*simp*]: *d* < *length* *p*
shows *p* \notin *grid* (*child* *p dir d*) *ds*
proof (*rule ccontr*)
assume \neg ?*thesis*
have *level* *p* < *level* (*child* *p dir d*) **by** *auto*
with *grid-level*[*OF* (< \neg ?*thesis*)[*unfolded not-not*]]
show *False* **by** *auto*
qed

2.4 Unbounded Sparse Grid

definition *sparsegrid'* :: *nat* \Rightarrow *grid-point set*
where
sparsegrid' *dm* = *grid* (*start* *dm*) { 0 ..< *dm* }

lemma *grid-subset-alldim*:

assumes $p: p \in \text{grid } b \text{ } ds$
defines $dm \equiv \text{length } b$
shows $p \in \text{grid } b \{0..<dm\}$
proof –
have $ds \cap \{dm..\} \cup ds \cap \{0..<dm\} = ds$ **by** *auto*
from *gridgen-dim-restrict*[**where** $ds=ds \cap \{0..<dm\}$ **and** $ds'=ds \cap \{dm..\}$] *this*
have $ds \cap \{0..<dm\} \subseteq \{0..<dm\}$
and $p \in \text{grid } b (ds \cap \{0..<dm\})$ **using** p **unfolding** *dm-def* **by** *auto*
thus *?thesis* **by** (rule *grid-union-dims*)
qed

lemma *sparsegrid'-length*[*simp*]:
 $b \in \text{sparsegrid}' \text{ } dm \implies \text{length } b = dm$ **unfolding** *sparsegrid'-def* **by** *auto*

lemma *sparsegrid'I*[*intro*]:
assumes $b: b \in \text{sparsegrid}' \text{ } dm$ **and** $p: p \in \text{grid } b \text{ } ds$
shows $p \in \text{sparsegrid}' \text{ } dm$
using *sparsegrid'-length*[*OF* b] b
grid-transitive[*OF* *grid-subset-alldim*[*OF* p], **where** $c=\text{start } dm$ **and** $ds''=\{0..<dm\}$]
unfolding *sparsegrid'-def* **by** *auto*

lemma *sparsegrid'-start*:
assumes $b \in \text{grid } (\text{start } dm) \text{ } ds$
shows $b \in \text{sparsegrid}' \text{ } dm$
unfolding *sparsegrid'-def*
using *grid-subset-alldim*[*OF* *assms*] **by** *simp*

2.5 Sparse Grid

definition *sparsegrid* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{grid-point set}$
where

$\text{sparsegrid } dm \text{ } lm = \text{lgrid } (\text{start } dm) \{ 0 ..< dm \} \text{ } lm$

lemma *sparsegrid-length*: $p \in \text{sparsegrid } dm \text{ } lm \implies \text{length } p = dm$
by (*auto simp: sparsegrid-def*)

lemma *sparsegrid-subset*[*intro*]: $p \in \text{sparsegrid } dm \text{ } lm \implies p \in \text{sparsegrid}' \text{ } dm$
unfolding *sparsegrid-def* *sparsegrid'-def* *lgrid-def* **by** *auto*

lemma *sparsegridI*[*intro*]:
assumes $p \in \text{sparsegrid}' \text{ } dm$ **and** $\text{level } p < lm$
shows $p \in \text{sparsegrid } dm \text{ } lm$
using *assms* **unfolding** *sparsegrid'-def* *sparsegrid-def* *lgrid-def* **by** *auto*

lemma *sparsegrid-start*:
assumes $b \in \text{lgrid } (\text{start } dm) \text{ } ds \text{ } lm$
shows $b \in \text{sparsegrid } dm \text{ } lm$

proof
have $b \in \text{grid } (\text{start } dm) \text{ } ds$ **using** *assms* **by** *auto*

thus $b \in \text{sparsegrid}' \text{ dm}$ **by** (rule *sparsegrid'-start*)
qed (*insert assms, auto*)

lemma *sparsegridE[elim]*:
assumes $p \in \text{sparsegrid} \text{ dm } lm$
shows $p \in \text{sparsegrid}' \text{ dm}$ **and** $\text{level } p < lm$
using *assms unfolding sparsegrid'-def sparsegrid-def lgrid-def* **by** *auto*

2.6 Compute Sparse Grid Points

fun *gridgen* :: *grid-point* \Rightarrow *nat set* \Rightarrow *nat* \Rightarrow *grid-point list*
where

gridgen $p \text{ ds } 0 = []$
| *gridgen* $p \text{ ds } (\text{Suc } l) = (\text{let}$
 $\text{sub} = \lambda d. \text{gridgen } (\text{child } p \text{ left } d) \{ d' \in \text{ds} . d' \leq d \} l @$
 $\text{gridgen } (\text{child } p \text{ right } d) \{ d' \in \text{ds} . d' \leq d \} l$
 $\text{in } p \# \text{concat } (\text{map } \text{sub } [d \leftarrow [0 ..< \text{length } p]. d \in \text{ds}])$)

lemma *gridgen-lgrid-eq*: $\text{set } (\text{gridgen } p \text{ ds } l) = \text{lgrid } p \text{ ds } (\text{level } p + l)$

proof (*induct l arbitrary: p ds*)

case (*Suc l*)

let $?sub \text{ dir } d = \text{set } (\text{gridgen } (\text{child } p \text{ dir } d) \{ d' \in \text{ds} . d' \leq d \} l)$

let $?sub \text{ dir } d = \text{lgrid } (\text{child } p \text{ dir } d) \{ d' \in \text{ds} . d' \leq d \} (\text{level } p + \text{Suc } l)$

let $?union \text{ F } dm = \{p\} \cup (\bigcup d \in \{ d \in \text{ds} . d < dm \}. \text{F left } d \cup \text{F right } d)$

have *hyp*: $!! \text{dir } d. d < \text{length } p \implies ?sub \text{ dir } d = ?sub \text{ dir } d$

using *Suc.hyps using child-level* **by** *auto*

{ fix dm **assume** $dm \leq \text{length } p$

hence $?union ?sub dm = \text{lgrid } p \{ d \in \text{ds} . d < dm \} (\text{level } p + \text{Suc } l)$

proof (*induct dm*)

case (*Suc dm*)

hence $dm \leq \text{length } p$ **by** *auto*

let $?l = \text{child } p \text{ left } dm$ **and** $?r = \text{child } p \text{ right } dm$

have *p-lgrid*: $p \in \text{lgrid } p \{ d \in \text{ds} . d < dm \} (\text{level } p + \text{Suc } l)$ **by** *auto*

show *?case*

proof (*cases dm \in ds*)

case *True*

let $?ds = \{ d \in \text{ds} . d < dm \} \cup \{ dm \}$

have *ds-eq*: $\{ d' \in \text{ds} . d' \leq dm \} = ?ds$ **using** *True* **by** *auto*

have *ds-eq'*: $\{ d \in \text{ds} . d < \text{Suc } dm \} = \{ d \in \text{ds} . d < dm \} \cup \{ dm \}$ **using**

True **by** *auto*

have $?union ?sub (\text{Suc } dm) = ?union ?sub dm \cup (\{p\} \cup ?sub \text{ left } dm \cup ?sub \text{ right } dm)$

unfolding *ds-eq'* **by** *auto*

also have $\dots = \text{lgrid } p \{d \in ds. d < dm\} (\text{level } p + \text{Suc } l) \cup ?\text{sub left } dm$
 $\cup ?\text{sub right } dm$
unfolding $\text{Suc.hyps}[OF \langle dm \leq \text{length } p \rangle]$ **using** $p\text{-lgrid}$ **by** auto
also have $\dots = \{p' \in \text{grid } p \{d \in ds. d < dm\} \cup (\text{grid } ?l \ ?ds) \cup (\text{grid } ?r$
 $?\text{ds})$.
 $\text{level } p' < \text{level } p + \text{Suc } l$ **unfolding** lgrid-def ds-eq **by** auto
also have $\dots = \text{lgrid } p \{d \in ds. d < \text{Suc } dm\} (\text{level } p + \text{Suc } l)$
unfolding $\text{lgrid-def ds-eq}'$ **unfolding** $\text{grid-onedim-split}[\text{where } b=p]$ **..**
finally show $?thesis$.
next
case False **hence** $\{d \in ds. d < \text{Suc } dm\} = \{d \in ds. d < dm \vee d = dm\}$
by auto
hence $\text{ds-eq}: \{d \in ds. d < \text{Suc } dm\} = \{d \in ds. d < dm\}$ **using** $\langle dm \notin ds \rangle$
by auto
show $?thesis$ **unfolding** $\text{ds-eq Suc.hyps}[OF \langle dm \leq \text{length } p \rangle]$ **..**
qed
next case 0 **thus** $?case$ **unfolding** lgrid-def **by** auto
qed }
hence $?union ?\text{sub} (\text{length } p) = \text{lgrid } p \{d \in ds. d < \text{length } p\} (\text{level } p + \text{Suc } l)$
by auto
hence $\text{union-lgrid-eq}: ?union ?\text{sub} (\text{length } p) = \text{lgrid } p \text{ ds } (\text{level } p + \text{Suc } l)$
unfolding lgrid-def **using** $\text{grid-dim-remove-outer}$ **by** auto

have $\text{set } (\text{gridgen } p \text{ ds } (\text{Suc } l)) = ?union ?\text{subg} (\text{length } p)$
unfolding gridgen.simps **and** Let-def **by** auto
hence $\text{set } (\text{gridgen } p \text{ ds } (\text{Suc } l)) = ?union ?\text{sub} (\text{length } p)$
using hyp **by** auto
also have $\dots = \text{lgrid } p \text{ ds } (\text{level } p + \text{Suc } l)$
using union-lgrid-eq .
finally show $?case$.
qed auto

lemma $\text{gridgen-distinct}: \text{distinct } (\text{gridgen } p \text{ ds } l)$
proof $(\text{induct } l \text{ arbitrary}: p \text{ ds})$
case $(\text{Suc } l)$
let $?\text{ds} = [d \leftarrow [0..<\text{length } p]. d \in ds]$
let $?\text{left } d = \text{gridgen } (\text{child } p \text{ left } d) \{d' \in ds. d' \leq d\} l$
and $?\text{right } d = \text{gridgen } (\text{child } p \text{ right } d) \{d' \in ds. d' \leq d\} l$
let $?\text{sub } d = ?\text{left } d @ ?\text{right } d$

have $\text{distinct } (\text{concat } (\text{map } ?\text{sub } ?\text{ds}))$
proof $(\text{cases } l)$
case $(\text{Suc } l')$

have $\text{inj-on}: \text{inj-on } ?\text{sub} (\text{set } ?\text{ds})$
proof $(\text{rule } \text{inj-onI}, \text{rule } \text{ccontr})$
fix $d \ d'$ **assume** $d \in \text{set } ?\text{ds}$ **and** $d' \in \text{set } ?\text{ds}$
hence $d < \text{length } p$ **and** $d \in \text{set } ?\text{ds}$ **and** $d' < \text{length } p$ **by** auto
assume $*$: $?\text{sub } d = ?\text{sub } d'$

```

have in-d: child p left d ∈ set (?sub d)
  using ⟨d ∈ set ?ds⟩ Suc
  by (auto simp add: gridgen-lgrid-eq lgrid-def grid-Start)

have in-d': child p left d' ∈ set (?sub d')
  using ⟨d ∈ set ?ds⟩ Suc
  by (auto simp add: gridgen-lgrid-eq lgrid-def grid-Start)

{ fix p' d assume d ∈ set ?ds and p' ∈ set (?sub d)
  hence lv p d < lv p' d
    using grid-child-level
    by (auto simp add: gridgen-lgrid-eq lgrid-def grid-child-level) }
note level-less = this

assume d ≠ d'
show False
proof (cases d' < d)
  case True
    with in-d' ⟨?sub d = ?sub d'⟩ level-less[OF ⟨d ∈ set ?ds]
    have lv p d < lv (child p left d') d by simp
    thus False unfolding lv-def
      using child-invariant[OF ⟨d < length p⟩, of left d'] ⟨d ≠ d'⟩
      by auto
  next
    case False hence d < d' using ⟨d ≠ d'⟩ by auto
    with in-d ⟨?sub d = ?sub d'⟩ level-less[OF ⟨d' ∈ set ?ds]
    have lv p d' < lv (child p left d) d' by simp
    thus False unfolding lv-def
      using child-invariant[OF ⟨d' < length p⟩, of left d] ⟨d ≠ d'⟩
      by auto
  qed
qed

show ?thesis
proof (rule distinct-concat)
  show distinct (map ?sub ?ds)
    unfolding distinct-map using inj-on by simp
  next
    fix ys assume ys ∈ set (map ?sub ?ds)
    then obtain d where d ∈ ds and d < length p
    and *: ys = ?sub d by auto

    show distinct ys unfolding *
      using grid-disjunct[OF ⟨d < length p⟩, of {d' ∈ ds. d' ≤ d}]
      gridgen-lgrid-eq lgrid-def ⟨distinct (?left d)⟩ ⟨distinct (?right d)⟩
      by auto
  next
    fix ys zs
    assume ys ∈ set (map ?sub ?ds)

```

then obtain d where $ys: ys = ?sub\ d$ and $d \in set\ ?ds$ by auto
hence $d < length\ p$ by auto

assume $zs \in set\ (map\ ?sub\ ?ds)$
then obtain d' where $zs: zs = ?sub\ d'$ and $d' \in set\ ?ds$ by auto
hence $d' < length\ p$ by auto

assume $ys \neq zs$
hence $d' \neq d$ unfolding $ys\ zs$ by auto

show $set\ ys \cap set\ zs = \{\}$
proof (rule ccontr)
assume $\neg\ ?thesis$
then obtain p' where $p' \in set\ (?sub\ d)$ and $p' \in set\ (?sub\ d')$
unfolding $ys\ zs$ by auto

hence $lv\ p\ d < lv\ p'\ d\ lv\ p\ d' < lv\ p'\ d'$
using $grid-child-level\ \langle d \in set\ ?ds \rangle\ \langle d' \in set\ ?ds \rangle$
by (auto simp add: gridgen-lgrid-eq lgrid-def grid-child-level)

show $False$
proof (cases $d < d'$)
case True
from $\langle p' \in set\ (?sub\ d) \rangle$
have $p ! d' = p' ! d'$
using $grid-invariant[of\ d'\ child\ p\ right\ d\ \{d' \in ds.\ d' \leq d\}]$
using $grid-invariant[of\ d'\ child\ p\ left\ d\ \{d' \in ds.\ d' \leq d\}]$
using $child-invariant[of\ d' - -\ d]\ \langle d < d' \rangle\ \langle d' < length\ p \rangle$
using $gridgen-lgrid-eq\ lgrid-def$ by auto
thus $False$ using $\langle lv\ p\ d' < lv\ p'\ d' \rangle$ unfolding $lv-def$ by auto
next
case $False$ hence $d' < d$ using $\langle d' \neq d \rangle$ by simp
from $\langle p' \in set\ (?sub\ d') \rangle$
have $p ! d = p' ! d$
using $grid-invariant[of\ d\ child\ p\ right\ d'\ \{d \in ds.\ d \leq d'\}]$
using $grid-invariant[of\ d\ child\ p\ left\ d'\ \{d \in ds.\ d \leq d'\}]$
using $child-invariant[of\ d - -\ d']\ \langle d' < d \rangle\ \langle d < length\ p \rangle$
using $gridgen-lgrid-eq\ lgrid-def$ by auto
thus $False$ using $\langle lv\ p\ d < lv\ p'\ d \rangle$ unfolding $lv-def$ by auto
qed
qed
qed

qed (simp add: map-replicate-const)
moreover
have $p \notin set\ (concat\ (map\ ?sub\ ?ds))$
using $gridgen-lgrid-eq\ lgrid-def\ grid-not-child[of\ -\ p]$ by simp
ultimately show $?case$
unfolding $gridgen.simps\ Let-def\ distinct.simps$ by simp
qed auto

lemma *lgrid-finite*: *finite* (*lgrid* *b ds lm*)
proof (*cases level b ≤ lm*)
 case *True* **from** *iffD1[OF le-iff-add True]*
 obtain *l* **where** *l: lm = level b + l* **by** *auto*
 show *?thesis unfolding l gridgen-lgrid-eq[symmetric]* **by** *auto*
next
 case *False* **hence** *!! x. x ∈ grid b ds ⇒ (¬ level x < lm)*
 proof –
 fix *x* **assume** *x ∈ grid b ds*
 from *grid-level[OF this]* **show** *¬ level x < lm* **using** *False* **by** *auto*
 qed
 hence *lgrid b ds lm = {}* **unfolding** *lgrid-def* **by** *auto*
 thus *?thesis* **by** *auto*
qed

lemma *lgrid-sum*:
 fixes *F :: grid-point ⇒ real*
 assumes *d < length b* **and** *level b < lm*
 shows $(\sum p \in \text{lgrid } b \{d\} \text{ lm. } F p) =$
 $(\sum p \in \text{lgrid } (\text{child } b \text{ left } d) \{d\} \text{ lm. } F p) + (\sum p \in \text{lgrid } (\text{child } b \text{ right } d) \{d\} \text{ lm. } F p) + F b$
 (is $(\sum p \in ?\text{grid } b. F p) = (\sum p \in ?\text{grid } ?l . F p) + (?sum (?grid ?r)) + F b$
proof –
 have *!! dir. b ∉ ?grid (child b dir d)*
 using *grid-child-without-parent[where ds={d}]* **and** *d < length b* **and** *lgrid-def*
by *auto*
 hence *b-distinct: b ∉ (?grid ?l ∪ ?grid ?r)* **by** *auto*

 have *?grid ?l ∩ ?grid ?r = {}*
 unfolding *lgrid-def* **using** *grid-disjunct* **and** *d < length b* **by** *auto*
 from *lgrid-finite lgrid-finite* **and** *this*
 have *child-eq: ?sum ((?grid ?l) ∪ (?grid ?r)) = ?sum (?grid ?l) + ?sum (?grid ?r)*
 by (*rule sum.union-disjoint*)

 have *?grid b = {b} ∪ (?grid ?l) ∪ (?grid ?r)* **unfolding** *lgrid-def* *grid-partition[where p=b]* **using** *assms* **by** *auto*
 hence *?sum (?grid b) = F b + ?sum ((?grid ?l) ∪ (?grid ?r))* **using** *b-distinct*
and *lgrid-finite* **by** *auto*
 thus *?thesis* **using** *child-eq* **by** *auto*
qed

2.7 Base Points

definition *base* :: *nat set ⇒ grid-point ⇒ grid-point*
where *base ds p = (THE b. b ∈ grid (start (length p)) ({0 ..< length p} - ds) ∧ p ∈ grid b ds)*

lemma baseE: **assumes** $p\text{-grid}$: $p \in \text{sparsegrid}'\ dm$
shows $\text{base } ds\ p \in \text{grid } (\text{start } dm) (\{0..<dm\} - ds)$
and $p \in \text{grid } (\text{base } ds\ p)\ ds$
proof –
from $p\text{-grid}[\text{unfolded } \text{sparsegrid}'\text{-def}]$
have $*$: $\exists! x \in \text{grid } (\text{start } dm) (\{0..<dm\} - ds)$. $p \in \text{grid } x\ ds$
by ($\text{intro } \text{grid-split1}$) ($\text{auto } \text{intro}: \text{grid-union-dims}$)
then obtain x **where** $x\text{-eq}$: $x \in \text{grid } (\text{start } dm) (\{0..<dm\} - ds) \wedge p \in \text{grid } x\ ds$
by auto
with $*$ **have** $\text{base } ds\ p = x$ **unfolding** base-def **by** auto
thus $\text{base } ds\ p \in \text{grid } (\text{start } dm) (\{0..<dm\} - ds)$ **and** $p \in \text{grid } (\text{base } ds\ p)\ ds$
using $x\text{-eq}$ **by** auto
qed

lemma baseI: **assumes** $x\text{-grid}$: $x \in \text{grid } (\text{start } dm) (\{0..<dm\} - ds)$ **and** $p\text{-xgrid}$:
 $p \in \text{grid } x\ ds$
shows $\text{base } ds\ p = x$
proof –
have $p \in \text{grid } (\text{start } dm) (ds \cup (\{0..<dm\} - ds))$
using $\text{grid-transitive}[OF\ p\text{-xgrid } x\text{-grid}, \text{where } ds''=ds \cup (\{0..<dm\} - ds)]$ **by**
 auto
moreover **have** $ds \cap (\{0..<dm\} - ds) = \{\}$ **by** auto
ultimately **have** $\exists! x \in \text{grid } (\text{start } dm) (\{0..<dm\} - ds)$. $p \in \text{grid } x\ ds$
using $\text{grid-split1}[\text{where } p=p \text{ and } b=\text{start } dm \text{ and } ds'=ds \text{ and } ds=\{0..<dm\} - ds]$ **by** auto
thus $\text{base } ds\ p = x$ **using** $x\text{-grid } p\text{-xgrid}$ **unfolding** base-def **by** auto
qed

lemma base-empty: **assumes** $p\text{-grid}$: $p \in \text{sparsegrid}'\ dm$ **shows** $\text{base } \{\}\ p = p$
using grid-empty-ds **and** $p\text{-grid}$ **and** $\text{grid-split1}[\text{where } ds=\{0..<dm\} \text{ and } ds'=\{\}]$
unfolding $\text{base-def } \text{sparsegrid}'\text{-def}$ **by** auto

lemma base-start-eq: **assumes** $p\text{-spg}$: $p \in \text{sparsegrid } dm\ lm$
shows $\text{start } dm = \text{base } \{0..<dm\}\ p$
proof –
from $p\text{-spg}$
have $\text{start } dm \in \text{grid } (\text{start } dm) (\{0..<dm\} - \{0..<dm\})$
and $p \in \text{grid } (\text{start } dm) \{0..<dm\}$ **using** $\text{sparsegrid}'\text{-def}$ **by** auto
from $\text{baseI}[OF\ \text{this}(1)\ \text{this}(2)]$ **show** $?thesis$ **by** auto
qed

lemma base-in-grid: **assumes** $p\text{-grid}$: $p \in \text{sparsegrid}'\ dm$ **shows** $\text{base } ds\ p \in \text{grid } (\text{start } dm) \{0..<dm\}$
proof –
let $?ds = ds \cup \{0..<dm\}$
have $ds\text{-eq}$: $\{d \in ?ds. d < \text{length } (\text{start } dm)\} = \{0..<dm\}$
unfolding start-def **by** auto
have $\text{base } ds\ p \in \text{grid } (\text{start } dm) ?ds$

using *grid-union-dims*[*OF* - *baseE*(1)[*OF* *p-grid*, **where** $ds=ds$], **where** $ds'=?ds$
by *auto*
thus *?thesis* **using** *grid-dim-remove-outer*[**where** $b=start\ dm$ **and** $ds=?ds$]
unfolding *ds-eq* **by** *auto*
qed

lemma *base-grid*: **assumes** *p-grid*: $p \in sparsegrid'\ dm$ **shows** $grid\ (base\ ds\ p)\ ds \subseteq sparsegrid'\ dm$

proof

fix x **assume** *xgrid*: $x \in grid\ (base\ ds\ p)\ ds$
have *ds-eq*: $\{d \in \{0..<dm\} \cup ds.\ d < length\ (start\ dm)\} = \{0..<dm\}$ **by** *auto*
from *grid-transitive*[*OF* *xgrid* *base-in-grid*[*OF* *p-grid*], **where** $ds''=\{0..<dm\} \cup ds$]
show $x \in sparsegrid'\ dm$ **unfolding** *sparsegrid'-def*

using *grid-dim-remove-outer*[**where** $b=start\ dm$ **and** $ds=\{0..<dm\} \cup ds$]
unfolding *ds-eq* **unfolding** *Un-ac*(3)[*of* $\{0..<dm\}$]
by *auto*

qed

lemma *base-length*[*simp*]: **assumes** *p-grid*: $p \in sparsegrid'\ dm$ **shows** $length\ (base\ ds\ p) = dm$

proof –

from *baseE*[*OF* *p-grid*] **have** $base\ ds\ p \in grid\ (start\ dm)\ (\{0..<dm\} - ds)$ **by** *auto*

thus *?thesis* **by** *auto*

qed

lemma *base-in*[*simp*]: **assumes** $d < dm$ **and** $d \in ds$ **and** *p-grid*: $p \in sparsegrid'\ dm$ **shows** $base\ ds\ p ! d = start\ dm ! d$

proof –

have *ds*: $d \notin \{0..<dm\} - ds$ **using** $\langle d \in ds \rangle$ **by** *auto*

have $d < length\ (start\ dm)$ **using** $\langle d < dm \rangle$ **by** *auto*

with *grid-invariant*[*OF* *this* *ds*] *baseE*(1)[*OF* *p-grid*] **show** *?thesis* **by** *auto*

qed

lemma *base-out*[*simp*]: **assumes** $d < dm$ **and** $d \notin ds$ **and** *p-grid*: $p \in sparsegrid'\ dm$ **shows** $base\ ds\ p ! d = p ! d$

proof –

have $d < length\ (base\ ds\ p)$ **using** *base-length*[*OF* *p-grid*] $\langle d < dm \rangle$ **by** *auto*

with *grid-invariant*[*OF* *this* $\langle d \notin ds \rangle$] *baseE*(2)[*OF* *p-grid*] **show** *?thesis* **by** *auto*

qed

lemma *base-base*: **assumes** *p-grid*: $p \in sparsegrid'\ dm$ **shows** $base\ ds\ (base\ ds'\ p) = base\ (ds \cup ds')\ p$

proof (*rule nth-equalityI*)

have *b-spg*: $base\ ds'\ p \in sparsegrid'\ dm$ **unfolding** *sparsegrid'-def*

using *grid-union-dims*[*OF* *Diff-subset*[**where** $A=\{0..<dm\}$ **and** $B=ds'$] *baseE*(1)[*OF* *p-grid*]] .

from *base-length*[*OF* *b-spg*] *base-length*[*OF* *p-grid*] **show** $length\ (base\ ds\ (base\ ds'\ p)) = length\ (base\ (ds \cup ds')\ p)$ **by** *auto*

show $base\ ds\ (base\ ds'\ p) ! i = base\ (ds \cup ds')\ p ! i$ **if** $i < length\ (base\ ds\ (base\ ds'\ p))$ **for** i

proof –
have $i < dm$ **using** *that base-length[OF b-spg]* **by auto**
show $\text{base } ds \ (\text{base } ds' \ p) \ ! \ i = \text{base } (ds \cup \ ds') \ p \ ! \ i$
proof (*cases* $i \in ds \cup \ ds'$)
case *True*
show *?thesis*
proof (*cases* $i \in ds$)
case *True* **from** $\text{base-in}[OF \ \langle i < dm \rangle \ \langle i \in ds \cup \ ds' \rangle \ p\text{-grid}] \ \text{base-in}[OF \ \langle i < dm \rangle \ \text{this } b\text{-spg}]$ **show** *?thesis* **by auto**
next
case *False* **hence** $i \in ds'$ **using** $\langle i \in ds \cup \ ds' \rangle$ **by auto**
from $\text{base-in}[OF \ \langle i < dm \rangle \ \langle i \in ds \cup \ ds' \rangle \ p\text{-grid}] \ \text{base-out}[OF \ \langle i < dm \rangle \ \langle i \notin ds \rangle \ b\text{-spg}] \ \text{base-in}[OF \ \langle i < dm \rangle \ \langle i \in ds' \rangle \ p\text{-grid}]$ **show** *?thesis* **by auto**
qed
next
case *False* **hence** $i \notin ds$ **and** $i \notin ds'$ **by auto**
from $\text{base-out}[OF \ \langle i < dm \rangle \ \langle i \notin ds \cup \ ds' \rangle \ p\text{-grid}] \ \text{base-out}[OF \ \langle i < dm \rangle \ \langle i \notin ds \rangle \ b\text{-spg}] \ \text{base-out}[OF \ \langle i < dm \rangle \ \langle i \notin ds' \rangle \ p\text{-grid}]$ **show** *?thesis* **by auto**
qed
qed
qed
lemma *grid-base-out*: **assumes** $d < dm$ **and** $d \notin ds$ **and** $p\text{-grid}: b \in \text{sparsegrid}'$ dm **and** $p \in \text{grid} \ (\text{base } ds \ b) \ ds$
shows $p \ ! \ d = b \ ! \ d$
proof –
have $\text{base } ds \ b \ ! \ d = b \ ! \ d$ **using** *assms* **by auto**
moreover **have** $d < \text{length} \ (\text{base } ds \ b)$ **using** *assms* **by auto**
from *grid-invariant[OF this]*
have $p \ ! \ d = \text{base } ds \ b \ ! \ d$ **using** *assms* **by auto**
ultimately show *?thesis* **by auto**
qed

lemma *grid-grid-inj-on*: **assumes** $ds \cap \ ds' = \{\}$ **shows** *inj-on snd* $(\bigcup_{p' \in \text{grid}} b \ ds. \bigcup_{p'' \in \text{grid}} p' \ ds'. \ \{(p', \ p'')\})$
proof (*rule inj-onI*)
fix $x \ y$
assume $x \in (\bigcup_{p' \in \text{grid}} b \ ds. \bigcup_{p'' \in \text{grid}} p' \ ds'. \ \{(p', \ p'')\})$
hence $\text{snd } x \in \text{grid} \ (\text{fst } x) \ ds'$ **and** $\text{fst } x \in \text{grid} \ b \ ds$ **by auto**

assume $y \in (\bigcup_{p' \in \text{grid}} b \ ds. \bigcup_{p'' \in \text{grid}} p' \ ds'. \ \{(p', \ p'')\})$
hence $\text{snd } y \in \text{grid} \ (\text{fst } y) \ ds'$ **and** $\text{fst } y \in \text{grid} \ b \ ds$ **by auto**

assume $\text{snd } x = \text{snd } y$
have $\text{fst } x = \text{fst } y$
proof (*rule ccontr*)
assume $\text{fst } x \neq \text{fst } y$
from *grid-disjunct'[OF \ \langle \text{fst } x \in \text{grid} \ b \ ds \rangle \ \langle \text{fst } y \in \text{grid} \ b \ ds \rangle \ \langle \text{snd } x \in \text{grid} \ (\text{fst } x) \ ds' \rangle \ \text{this} \ \langle ds \cap \ ds' = \{\} \rangle]*
show *False* **using** $\langle \text{snd } y \in \text{grid} \ (\text{fst } y) \ ds' \rangle$ **unfolding** $\langle \text{snd } x = \text{snd } y \rangle$ **by auto**

qed
show $x = y$ **using** *prod-eqI*[*OF* $\langle \text{fst } x = \text{fst } y \rangle \langle \text{snd } x = \text{snd } y \rangle$].
qed

lemma *grid-level-d*: **assumes** $d < \text{length } b$ **and** *p-grid*: $p \in \text{grid } b \{d\}$ **and** $p \neq b$ **shows** $\text{lv } p \ d > \text{lv } b \ d$

proof –
from *p-grid*[*unfolded grid-partition*[**where** $p=b$]]
show *?thesis* **using** *grid-child-level* **using** *assms* **by** *auto*
qed

lemma *grid-base-base*: **assumes** $b \in \text{sparsegrid}' \ dm$
shows $\text{base } ds' \ b \in \text{grid} (\text{base } ds (\text{base } ds' \ b)) (ds \cup ds')$

proof –
from *base-grid*[*OF* $\langle b \in \text{sparsegrid}' \ dm \rangle$] **have** $\text{base } ds' \ b \in \text{sparsegrid}' \ dm$ **by** *auto*
from *grid-union-dims*[*OF* - *baseE*(2)] [*OF* *this*], *of* $ds \ ds \cup ds'$] **show** *?thesis* **by** *auto*
qed

lemma *grid-base-union*: **assumes** *b-spg*: $b \in \text{sparsegrid}' \ dm$ **and** *p-grid*: $p \in \text{grid} (\text{base } ds \ b) \ ds$ **and** *x-grid*: $x \in \text{grid} (\text{base } ds' \ p) \ ds'$
shows $x \in \text{grid} (\text{base} (ds \cup ds') \ b) (ds \cup ds')$

proof –
have *ds-union*: $ds \cup ds' = ds' \cup (ds \cup ds')$ **by** *auto*

from *base-grid*[*OF* *b-spg*] *p-grid* **have** *p-spg*: $p \in \text{sparsegrid}' \ dm$ **by** *auto*
with *assms* **and** *grid-base-base* **have** *base-b'*: $\text{base } ds' \ p \in \text{grid} (\text{base } ds (\text{base } ds' \ p)) (ds \cup ds')$ **by** *auto*

moreover **have** $\text{base } ds' (\text{base } ds \ b) = \text{base } ds' (\text{base } ds \ p)$ (**is** $?b = ?p$)

proof (*rule nth-equalityI*)

have *bb-spg*: $\text{base } ds \ b \in \text{sparsegrid}' \ dm$ **using** *base-grid*[*OF* *b-spg*] *grid.Start* **by** *auto*

hence $dm = \text{length} (\text{base } ds \ b)$ **by** *auto*

have *bp-spg*: $\text{base } ds \ p \in \text{sparsegrid}' \ dm$ **using** *base-grid*[*OF* *p-spg*] *grid.Start* **by** *auto*

show $\text{length } ?b = \text{length } ?p$ **using** *base-length*[*OF* *bp-spg*] *base-length*[*OF* *bb-spg*] **by** *auto*

show $?b \ ! \ i = ?p \ ! \ i$ **if** $i < \text{length } ?b$ **for** i

proof –

have $i < dm$ **and** $i < \text{length} (\text{base } ds \ b)$ **using** *that* *base-length*[*OF* *bb-spg*] $\langle dm = \text{length} (\text{base } ds \ b) \rangle$ **by** *auto*

show $?b \ ! \ i = ?p \ ! \ i$

proof (*cases* $i \in ds \cup ds'$)

case *True*

hence $!! \ x. \text{base } ds \ x \in \text{sparsegrid}' \ dm \implies x \in \text{sparsegrid}' \ dm \implies \text{base } ds' (\text{base } ds \ x) \ ! \ i = (\text{start } dm) \ ! \ i$

proof – **fix** x **assume** $x \in \text{sparsegrid}' \ dm$ **and** *xb-spg*: $\text{base } ds \ x \in$

```

sparsegrid' dm
  show base ds' (base ds x) ! i = (start dm) ! i
  proof (cases i ∈ ds')
    case True from base-in[OF ⟨i < dm⟩ this xb-spg] show ?thesis .
  next
    case False hence i ∈ ds using ⟨i ∈ ds ∪ ds'⟩ by auto
    from base-out[OF ⟨i < dm⟩ False xb-spg] base-in[OF ⟨i < dm⟩ this x-spg]
show ?thesis by auto
  qed
  qed
  from this[OF bp-spg p-spg] this[OF bb-spg b-spg] show ?thesis by auto
next
  case False hence i ∉ ds and i ∉ ds' by auto
  from grid-invariant[OF ⟨i < length (base ds b)⟩ ⟨i ∉ ds⟩ p-grid]
    base-out[OF ⟨i < dm⟩ ⟨i ∉ ds'⟩ bp-spg] base-out[OF ⟨i < dm⟩ ⟨i ∉ ds⟩ p-spg]
base-out[OF ⟨i < dm⟩ ⟨i ∉ ds'⟩ bb-spg]
  show ?thesis by auto
  qed
  qed
  qed
  ultimately have base ds' p ∈ grid (base (ds ∪ ds') b) (ds ∪ ds')
    by (simp only: base-base[OF p-spg] base-base[OF b-spg] Un-ac(3))
  from grid-transitive[OF x-grid this] show ?thesis using ds-union by auto
qed
lemma grid-base-dim-add: assumes ds' ⊆ ds and b-spg: b ∈ sparsegrid' dm and
p-grid: p ∈ grid (base ds' b) ds'
  shows p ∈ grid (base ds b) ds
proof -
  have ds-eq: ds' ∪ ds = ds using assms by auto

  have p ∈ sparsegrid' dm using base-grid[OF b-spg] p-grid by auto
  hence p ∈ grid (base ds p) ds using baseE by auto
  from grid-base-union[OF b-spg p-grid this]
  show ?thesis using ds-eq by auto
qed
lemma grid-replace-dim: assumes d < length b' and d < length b and p-grid: p
∈ grid b ds and p'-grid: p' ∈ grid b' ds
  shows p[d := p' ! d] ∈ grid (b[d := b' ! d]) ds (is - ∈ grid ?b ds)
  using p'-grid and p-grid
proof induct
  case (Child p'' d' dir)
  hence p''-grid: p''[d := p'' ! d] ∈ grid ?b ds and d < length p'' using assms by
auto
  have d < length p using p-grid assms by auto
  thus ?case
proof (cases d' = d)
  case True
  from grid.Child[OF p''-grid ⟨d' ∈ ds⟩]
  show ?thesis unfolding child-def ix-def lv-def list-update-overwrite ⟨d' = d⟩

```

$nth\text{-list-update-eq}[OF \langle d < length\ p'' \rangle] \text{ } nth\text{-list-update-eq}[OF \langle d < length\ p \rangle] .$
next
case *False*
show *?thesis unfolding child-def nth-list-update-neq[OF False] using Child by auto*
qed
qed (*rule grid-change-dim*)
lemma *grid-shift-base*:
assumes *ds-dj*: $ds \cap ds' = \{\}$ **and** *b-spg*: $b \in \text{sparsegrid}'\ dm$ **and** *p-grid*: $p \in \text{grid}(\text{base}(ds' \cup ds)\ b)\ (ds' \cup ds)$
shows $\text{base}\ ds'\ p \in \text{grid}(\text{base}(ds \cup ds')\ b)\ ds$
proof –
from *grid-split[OF p-grid]*
obtain *x* **where** *x-grid*: $x \in \text{grid}(\text{base}(ds' \cup ds)\ b)\ ds$ **and** *p-xgrid*: $p \in \text{grid}\ x\ ds'$ **by** *auto*
from *grid-union-dims[OF - this(1)]*
have *x-spg*: $x \in \text{sparsegrid}'\ dm$ **using** *base-grid[OF b-spg]* **by** *auto*

have *b-len*: $\text{length}(\text{base}(ds' \cup ds)\ b) = dm$ **using** *base-length[OF b-spg]* **by** *auto*

define *d'* **where** $d' = dm$
moreover **have** $d' \leq dm \implies x \in \text{grid}(\text{start}\ dm)\ (\{0..<dm\} - \{d \in ds'.\ d < d'\})$
proof (*induct d'*)
case (*Suc d'*)
with *b-len* **have** *d'-b*: $d' < \text{length}(\text{base}(ds' \cup ds)\ b)$ **by** *auto*
show *?case*
proof (*cases d' \in ds'*)
case *True* **hence** $d' \notin ds$ **and** $d' \in ds' \cup ds$ **using** *ds-dj* **by** *auto*
have $\{0..<dm\} - \{d \in ds'.\ d < d'\} = (\{0..<dm\} - \{d \in ds'.\ d < d'\}) - \{d'\} \cup \{d'\}$ **using** *Suc d' \leq dm* **by** *auto*
also **have** $\dots = (\{0..<dm\} - \{d \in ds'.\ d < \text{Suc}\ d'\}) \cup \{d'\}$ **by** *auto*
finally **have** *x-g*: $x \in \text{grid}(\text{start}\ dm)\ (\{d'\} \cup (\{0..<dm\} - \{d \in ds'.\ d < \text{Suc}\ d'\}))$ **using** *Suc* **by** *auto*
from *grid-invariant[OF d'-b \langle d' \notin ds \rangle x-grid] base-in[OF - \langle d' \in ds' \cup ds \rangle b-spg] \langle \text{Suc}\ d' \leq dm \rangle*
have $x ! d' = \text{start}\ dm ! d'$ **by** *auto*
from *grid-dim-remove[OF x-g this]* **show** *?thesis* .
next
case *False*
hence $\{d \in ds'.\ d < \text{Suc}\ d'\} = \{d \in ds'.\ d < d' \vee d = d'\}$ **by** *auto*
also **have** $\dots = \{d \in ds'.\ d < d'\}$ **using** *False* **by** *auto*
finally **show** *?thesis* **using** *Suc* **by** *auto*
qed
next
case *0* **show** *?case* **using** *x-spg[unfolded sparsegrid'-def]* **by** *auto*
qed
moreover **have** $\{0..<dm\} - ds' = \{0..<dm\} - \{d \in ds'.\ d < dm\}$ **by** *auto*
ultimately **have** $x \in \text{grid}(\text{start}\ dm)\ (\{0..<dm\} - ds')$ **by** *auto*

from *baseI*[*OF this p-xgrid*] and *x-grid*
 show *?thesis* by (*auto simp: Un-ac(3)*)
 qed

2.8 Lift Operation over all Grid Points

definition *lift* :: (*nat* \Rightarrow *nat* \Rightarrow *grid-point* \Rightarrow *vector* \Rightarrow *vector*) \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *vector* \Rightarrow *vector*
where *lift f dm lm d* = *foldr* (λ *p. f d (lm - level p) p*) (*gridgen (start dm) ({ 0 ..< dm } - { d }) lm*)

lemma *lift*:

assumes $d < dm$ and $p \in \text{sparsegrid } dm \text{ } lm$
and *Fintro*: $\bigwedge l b p \alpha. \llbracket b \in \text{lgrid } (start \ dm) \ (\{0..<dm\} - \{d\}) \text{ } lm ;$
 $l + \text{level } b = lm ; p \in \text{sparsegrid } dm \text{ } lm \rrbracket$
 $\implies F \ d \ l \ b \ \alpha \ p = (\text{if } b = \text{base } \{d\} \ p$
 $\text{then } (\sum p' \in \text{lgrid } b \ \{d\} \text{ } lm. S \ (\alpha \ p') \ p \ p')$
 $\text{else } \alpha \ p)$
shows *lift F dm lm d* α $p = (\sum p' \in \text{lgrid } (\text{base } \{d\} \ p) \ \{d\} \text{ } lm. S \ (\alpha \ p') \ p \ p')$
 (*is ?lift = ?S p* α)

proof –

let *?gridgen* = *gridgen (start dm) ({0..<dm} - {d}) lm*
let *?f p* = *F d (lm - level p) p*

{ **fix** *bs* β *b*

assume $set \ bs \subseteq set \ ?gridgen$ and *distinct bs* and $p \in \text{sparsegrid } dm \text{ } lm$

hence *foldr ?f bs* β $p = (\text{if } \text{base } \{d\} \ p \in set \ bs \text{ then } ?S \ p \ \beta \text{ else } \beta \ p)$

proof (*induct bs arbitrary: p*)

case (*Cons b bs*)

hence $b \in \text{lgrid } (start \ dm) \ (\{0..<dm\} - \{d\}) \text{ } lm$

and $(lm - \text{level } b) + \text{level } b = lm$

and *b-grid*: $b \in \text{grid } (start \ dm) \ (\{0..<dm\} - \{d\})$

using *lgrid-def gridgen-lgrid-eq* by *auto*

note $F = \text{Fintro}[OF \ \text{this}(1,2)] \ \langle p \in \text{sparsegrid } dm \text{ } lm \rangle$

have $b \notin set \ bs$ using $\langle \text{distinct } (b \# bs) \rangle$ by *auto*

show *?case*

proof (*cases base {d} p* $\in set \ (b \# bs)$)

case *True* **note** *base-in-set = this*

show *?thesis*

proof (*cases b = base {d} p*)

case *True*

moreover

{ **fix** $p' \in \text{lgrid } b \ \{d\} \text{ } lm$

hence $p' \in \text{grid } b \ \{d\}$ and $p' < lm$ **unfolding** *lgrid-def* by *auto*

from *grid-transitive*[*OF this(1) b-grid, of {0..<dm}*] $\langle d < dm \rangle$

baseI[*OF b-grid* $\langle p' \in \text{grid } b \ \{d\} \rangle$] $\langle b \notin set \ bs \rangle$

```

      Cons.prems Cons.hyps[of p] this(2)
    have foldr ?f bs β p' = β p' unfolding sparsegrid-def lgrid-def by auto }
  ultimately show ?thesis
    using F base-in-set by auto
next
  case False
  with base-in-set have base {d} p ∈ set bs by auto
  with Cons.hyps[of p] Cons.prems
  have foldr ?f bs β p = ?S p β by auto
  thus ?thesis using F base-in-set False by auto
qed
next
  case False
  hence b ≠ base {d} p by auto
  from False Cons.hyps[of p] Cons.prems
  have foldr ?f bs β p = β p by auto
  thus ?thesis using False F ⟨b ≠ base {d} p⟩ by auto
qed
qed auto
}
moreover have base {d} p ∈ set ?gridgen
proof -
  have p ∈ grid (base {d} p) {d}
    using ⟨p ∈ sparsegrid dm lm⟩[THEN sparsegrid-subset] by (rule baseE)
  from grid-level[OF this] baseE(1)[OF sparsegrid-subset[OF ⟨p ∈ sparsegrid dm
lm⟩]]
  show ?thesis using ⟨p ∈ sparsegrid dm lm⟩
    unfolding gridgen-lgrid-eq sparsegrid'-def lgrid-def sparsegrid-def
    by auto
qed
ultimately show ?thesis unfolding lift-def
  using gridgen-distinct ⟨p ∈ sparsegrid dm lm⟩ by auto
qed

```

2.9 Parent Points

definition *parents* :: *nat ⇒ grid-point ⇒ grid-point ⇒ grid-point set*
where *parents d b p = { x ∈ grid b {d}. p ∈ grid x {d} }*

lemma *parents-split*: **assumes** *p-grid*: *p ∈ grid (child b dir d) {d}*
shows *parents d b p = { b } ∪ parents d (child b dir d) p*

proof (*intro set-eqI iffI*)

let ?*chd* = *child b dir d* **and** ?*chid* = *child b (inv dir) d*

fix *x* **assume** *x ∈ parents d b p*

hence *x ∈ grid b {d}* **and** *p ∈ grid x {d}* **unfolding** *parents-def* **by** *auto*

hence *x-split*: *x ∈ {b} ∪ grid ?chd {d} ∪ grid ?chid {d}* **using** *grid-onedim-split* [**where**
ds={} **and** *b=b*] **and** *grid-empty-ds*

by (*cases dir, auto*)

thus *x ∈ {b} ∪ parents d (child b dir d) p*


```

proof (cases x = b)
  case False
  have d < length b
  proof (rule ccontr)
    assume ¬ d < length b hence empty: {d' ∈ {d}. d' < length b} = {} by auto
    have x = b using ⟨x ∈ grid b {d}⟩
      unfolding grid-dim-remove-outer[where ds={d} and b=b] empty
      using grid-empty-ds by auto
    thus False using ⟨¬ x = b⟩ by auto
  qed
  have x ∉ grid ?chid {d}
  proof (rule ccontr)
    assume ¬ x ∉ grid ?chid {d}
    hence p ∈ grid ?chid {d} using grid-transitive[OF ⟨p ∈ grid x {d}⟩, where
ds'={d}]
      by auto
    hence p ∉ grid ?chd {d} using grid-disjunct[OF ⟨d < length b⟩] by (cases
dir, auto)
    thus False using ⟨p ∈ grid ?chd {d}⟩ ..
  qed
  with False and x-split
  have x ∈ grid ?chd {d} by auto
  thus ?thesis unfolding parents-def using ⟨p ∈ grid x {d}⟩ by auto
qed auto
next
let ?chd = child b dir d and ?chid = child b (inv dir) d
fix x assume x-in: x ∈ {b} ∪ parents d ?chd p
thus x ∈ parents d b p
proof (cases x = b)
  case False
    hence x ∈ parents d ?chd p using x-in by auto
    thus ?thesis unfolding parents-def using grid-child[where b=b] by auto
  next
    from p-grid have p ∈ grid b {d} using grid-child[where b=b] by auto
    case True thus ?thesis unfolding parents-def using ⟨p ∈ grid b {d}⟩ by auto
  qed
qed

lemma parents-no-parent: assumes d < length b shows b ∉ parents d (child b dir
d) p (is - ∉ parents - ?ch -)
proof
  assume b ∈ parents d ?ch p hence b ∈ grid ?ch {d} unfolding parents-def by
auto
  from grid-level[OF this]
  have level b + 1 ≤ level b unfolding child-level[OF ⟨d < length b⟩] .
  thus False by auto
qed

lemma parents-subset-lgrid: parents d b p ⊆ lgrid b {d} (level p + 1)

```

proof

fix x **assume** $x \in \text{parents } d \ b \ p$
hence $x \in \text{grid } b \ \{d\}$ **and** $p \in \text{grid } x \ \{d\}$ **unfolding** *parents-def* **by** *auto*
moreover **hence** $\text{level } x \leq \text{level } p$ **using** *grid-level* **by** *auto*
hence $\text{level } x < \text{level } p + 1$ **by** *auto*
ultimately show $x \in \text{lgrid } b \ \{d\}$ ($\text{level } p + 1$) **unfolding** *lgrid-def* **by** *auto*
qed

lemma *parents-finite*: *finite (parents d b p)*
using *finite-subset[OF parents-subset-lgrid lgrid-finite]* .

lemma *parent-sum*: **assumes** *p-grid*: $p \in \text{grid } (\text{child } b \ \text{dir } d) \ \{d\}$ **and** $d < \text{length } b$
shows $(\sum x \in \text{parents } d \ b \ p. F \ x) = F \ b + (\sum x \in \text{parents } d \ (\text{child } b \ \text{dir } d) \ p. F \ x)$
unfolding *parents-split[OF p-grid]* **using** *parents-no-parent[OF <d < length b]*,
where $\text{dir} = \text{dir}$ **and** $p = p$ **using** *parents-finite*
by *auto*

lemma *parents-single*: $\text{parents } d \ b \ b = \{ b \}$

proof

have $\text{parents } d \ b \ b \subseteq \text{lgrid } b \ \{d\}$ ($\text{level } b + (\text{Suc } 0)$) **using** *parents-subset-lgrid*
by *auto*
also **have** $\dots = \{b\}$ **unfolding** *gridgen-lgrid-eq[symmetric]* *gridgen.simps* *Let-def*
by *auto*
finally show $\text{parents } d \ b \ b \subseteq \{ b \}$.

next

have $b \in \text{parents } d \ b \ b$ **unfolding** *parents-def* **by** *auto*
thus $\{ b \} \subseteq \text{parents } d \ b \ b$ **by** *auto*

qed

lemma *grid-single-dimensional-specification*:

assumes $d < \text{length } b$
and *odd i*
and $lv \ b \ d + l' = l$
and $i < (ix \ b \ d + 1) * 2^{l'}$
and $i > (ix \ b \ d - 1) * 2^{l'}$
shows $b[d := (l, i)] \in \text{grid } b \ \{d\}$
using *assms* **proof** (*induct l' arbitrary: b*)
case 0
hence $i = ix \ b \ d$ **and** $l = lv \ b \ d$ **by** *auto*
thus *?case* **unfolding** *ix-def* *lv-def* **by** *auto*
next
case (*Suc l'*)

have $d \in \{d\}$ **by** *auto*

show *?case*

proof (*rule linorder-cases*)

```

assume  $i = ix\ b\ d * 2^{\wedge}(Suc\ l')$ 
hence even i by auto
thus ?thesis using  $\langle odd\ i \rangle$  by blast
next
assume  $*$ :  $i < ix\ b\ d * 2^{\wedge}(Suc\ l')$ 

let  $?b = child\ b\ left\ d$ 

have  $d < length\ ?b$  using Suc by auto
moreover note  $\langle odd\ i \rangle$ 
moreover have  $lv\ ?b\ d + l' = l$ 
  and  $i < (ix\ ?b\ d + 1) * 2^{\wedge}l'$ 
  and  $(ix\ ?b\ d - 1) * 2^{\wedge}l' < i$ 
  unfolding child-ix-left[OF Suc.prems(1)]
  using Suc.prems * child-lv by (auto simp add: field-simps)
ultimately have  $?b[d := (l, i)] \in grid\ ?b\ \{d\}$ 
  by (rule Suc.hyps)
thus ?thesis
  by (auto intro!: grid-child[OF  $\langle d \in \{d\} \rangle$ , of - b left]
    simp add: child-def)
next
assume  $*$ :  $ix\ b\ d * 2^{\wedge}(Suc\ l') < i$ 

```

```

let  $?b = child\ b\ right\ d$ 

```

```

have  $d < length\ ?b$  using Suc by auto
moreover note  $\langle odd\ i \rangle$ 
moreover have  $lv\ ?b\ d + l' = l$ 
  and  $i < (ix\ ?b\ d + 1) * 2^{\wedge}l'$ 
  and  $(ix\ ?b\ d - 1) * 2^{\wedge}l' < i$ 
  unfolding child-ix-right[OF Suc.prems(1)]
  using Suc.prems * child-lv by (auto simp add: field-simps)
ultimately have  $?b[d := (l, i)] \in grid\ ?b\ \{d\}$ 
  by (rule Suc.hyps)
thus ?thesis
  by (auto intro!: grid-child[OF  $\langle d \in \{d\} \rangle$ , of - b right]
    simp add: child-def)

```

qed

qed

lemma *grid-multi-dimensional-specification*:

```

assumes  $dm \leq length\ b$  and  $length\ p = length\ b$ 
and  $\bigwedge d. d < dm \implies$ 
   $odd\ (ix\ p\ d) \wedge$ 
   $lv\ b\ d \leq lv\ p\ d \wedge$ 
   $ix\ p\ d < (ix\ b\ d + 1) * 2^{\wedge}(lv\ p\ d - lv\ b\ d) \wedge$ 
   $ix\ p\ d > (ix\ b\ d - 1) * 2^{\wedge}(lv\ p\ d - lv\ b\ d)$ 
  (is  $\bigwedge d. d < dm \implies ?bounded\ p\ d$ )
and  $\bigwedge d. \llbracket dm \leq d ; d < length\ b \rrbracket \implies p\ !\ d = b\ !\ d$ 

```

```

shows  $p \in \text{grid } b \{0..<dm\}$ 
using assms proof (induct dm arbitrary: p)
  case 0
  hence  $p = b$  by (auto intro!: nth-equalityI)
  thus ?case by auto
next
case (Suc dm)
hence  $dm \leq \text{length } b$ 
  and  $dm < \text{length } p$  by auto

let ?p =  $p[dm := b ! dm]$ 

note  $\langle dm \leq \text{length } b \rangle$ 
moreover have  $\text{length } ?p = \text{length } b$  using  $\langle \text{length } p = \text{length } b \rangle$  by simp
moreover
{
  fix d assume  $d < dm$ 
  hence  $*$ :  $d < \text{Suc } dm$  and  $dm \neq d$  by auto
  have  $?p ! d = p ! d$ 
    by (rule nth-list-update-neq[OF  $\langle dm \neq d \rangle$ ])
  hence ?bounded  $?p d$ 
    using Suc.prems(3)[OF  $*$ ] lv-def ix-def
    by simp
}
moreover
{
  fix d assume  $dm \leq d$  and  $d < \text{length } b$ 
  have  $?p ! d = b ! d$ 
  proof (cases d = dm)
    case True thus ?thesis using  $\langle d < \text{length } b \rangle$   $\langle \text{length } p = \text{length } b \rangle$  by auto
  next
    case False
    hence  $\text{Suc } dm \leq d$  using  $\langle dm \leq d \rangle$  by auto
    thus ?thesis using Suc.prems(4)  $\langle d < \text{length } b \rangle$  by auto
  qed
}
ultimately
have  $*$ :  $?p \in \text{grid } b \{0..<dm\}$ 
  by (auto intro!: Suc.hyps)

have  $lv \ b \ dm \leq lv \ p \ dm$  using Suc.prems(3)[OF lessI] by simp

have [simp]:  $lv \ ?p \ dm = lv \ b \ dm$  using lv-def  $\langle dm < \text{length } p \rangle$  by auto
have [simp]:  $ix \ ?p \ dm = ix \ b \ dm$  using ix-def  $\langle dm < \text{length } p \rangle$  by auto
have [simp]:  $p[dm := (lv \ p \ dm, ix \ p \ dm)] = p$ 
  using lv-def ix-def  $\langle dm < \text{length } p \rangle$  by auto
have  $dm < \text{length } ?p$  and
  [simp]:  $lv \ b \ dm + (lv \ p \ dm - lv \ b \ dm) = lv \ p \ dm$ 
  using  $\langle dm < \text{length } p \rangle$   $\langle lv \ b \ dm \leq lv \ p \ dm \rangle$  by auto

```

```

from grid-single-dimensional-specification[OF this(1)],
  where  $l=lv\ p\ dm$  and  $i=ix\ p\ dm$  and  $l'=lv\ p\ dm - lv\ b\ dm$ , simplified]
have  $p \in \text{grid } ?p\ \{dm\}$ 
  using Suc.premis(3)[OF lessI] by blast
from grid-transitive[OF this *]
show ?case by auto
qed

```

lemma *sparsegrid*:

```

sparsegrid  $dm\ lm = \{p.$ 
   $\text{length } p = dm \wedge \text{level } p < lm \wedge$ 
   $(\forall d < dm. \text{odd } (ix\ p\ d) \wedge 0 < ix\ p\ d \wedge ix\ p\ d < 2^{(lv\ p\ d + 1)})\}$ 
  (is - = ?set)
proof (rule equalityI[OF subsetI subsetI])
  fix  $p$ 
  assume  $*$ :  $p \in \text{sparsegrid } dm\ lm$ 
  hence  $\text{length } p = dm$  and  $\text{level } p < lm$  unfolding sparsegrid-def by auto
  moreover
  { fix  $d$  assume  $d < dm$ 
    hence  $**$ :  $p \in \text{grid } (\text{start } dm)\ \{0..<dm\}$  and  $d < \text{length } (\text{start } dm)$ 
    using  $*$  unfolding sparsegrid-def by auto
    have  $\text{odd } (ix\ p\ d)$ 
    proof (cases  $p\ !\ d = \text{start } dm\ !\ d$ )
      case True
      thus ?thesis unfolding start-def using  $\langle d < dm \rangle$  ix-def by auto
    next
      case False
      from grid-odd[OF - this **]
      show ?thesis using  $\langle d < dm \rangle$  by auto
    qed
    hence  $\text{odd } (ix\ p\ d) \wedge 0 < ix\ p\ d \wedge ix\ p\ d < 2^{(lv\ p\ d + 1)}$ 
    using grid-estimate[OF  $\langle d < \text{length } (\text{start } dm) \rangle **$ ]
    unfolding ix-def lv-def start-def using  $\langle d < dm \rangle$  by auto
  }
  ultimately show  $p \in ?set$ 
  using sparsegrid-def lgrid-def by auto
next
  fix  $p$ 
  assume  $p \in ?set$ 
  with grid-multi-dimensional-specification[of dm start dm p]
  have  $p \in \text{grid } (\text{start } dm)\ \{0..<dm\}$  and  $\text{level } p < lm$ 
  by auto
  thus  $p \in \text{sparsegrid } dm\ lm$ 
  unfolding sparsegrid-def lgrid-def by auto
qed
end

```

3 Hat Functions

theory *Triangular-Function*

imports

HOL-Analysis.Equivalence-Lebesgue-Henstock-Integration

Grid

begin

lemma *continuous-on-max*[*continuous-intros*]:

fixes $f :: - \Rightarrow 'a::linorder-topology$

shows $continuous-on\ S\ f \Longrightarrow continuous-on\ S\ g \Longrightarrow continuous-on\ S\ (\lambda x. max\ (f\ x)\ (g\ x))$

by (*auto simp: continuous-on-def intro: tendsto-max*)

definition $\varphi :: (nat \times int) \Rightarrow real \Rightarrow real$ **where**

$\varphi \equiv (\lambda(l,i)\ x. max\ 0\ (1 - |x * 2^{-(l+1)} - real-of-int\ i|))$

definition $\Phi :: (nat \times int)\ list \Rightarrow (nat \Rightarrow real) \Rightarrow real$ **where**

$\Phi\ p\ x = (\prod_{d < length\ p. \varphi\ (p!\ d)\ (x\ d))$

definition *l2- φ* **where**

$l2\text{-}\varphi\ p1\ p2 = (\int x. \varphi\ p1\ x * \varphi\ p2\ x\ \partial lborel)$

definition *l2* **where**

$l2\ a\ b = (\int x. \Phi\ a\ x * \Phi\ b\ x\ \partial(\prod_M\ d \in \{..<length\ a\}. lborel))$

lemma *measurable- φ* [*measurable*]: $\varphi\ p \in borel\text{-measurable}\ borel$

by (*cases p*) (*simp add: φ -def*)

lemma *φ -nonneg*: $0 \leq \varphi\ p\ x$

by (*simp add: φ -def split: prod.split*)

lemma *φ -zero-iff*:

$\varphi\ (l,i)\ x = 0 \iff x \notin \{real-of-int\ (i - 1) / 2^{-(l+1)} <..< real-of-int\ (i + 1) / 2^{-(l+1)}\}$

by (*auto simp: φ -def field-simps split: split-max*)

lemma *φ -zero*: $x \notin \{real-of-int\ (i - 1) / 2^{-(l+1)} <..< real-of-int\ (i + 1) / 2^{-(l+1)}\} \Longrightarrow \varphi\ (l,i)\ x = 0$

unfolding *φ -zero-iff* **by** *simp*

lemma *φ -eq-0*: **assumes** $x < 0 \vee 1 < x$ **and** $i < i < 2^{Suc\ l}$ **shows** $\varphi\ (l, i)\ x = 0$

using x

proof

assume $x < 0$

also have $0 \leq real-of-int\ (i - 1) / 2^{-(l+1)}$

using i **by** (*auto simp: field-simps*)

finally show *?thesis*

by (auto intro!: φ -zero simp: field-simps)
 next
 have $\text{real-of-int } (i + 1) / 2^{l+1} \leq 1$
 using i by (subst divide-le-eq-1-pos) (auto simp del: of-int-add power-Suc)
 also assume $1 < x$
 finally show ?thesis
 by (auto intro!: φ -zero simp: field-simps)
 qed

lemma $ix\text{-lt}$: $p \in \text{sparsegrid } dm \text{ } lm \implies d < dm \implies ix \text{ } p \text{ } d < 2^{l+1}$
 unfolding $\text{sparsegrid-def lgrid-def}$
 using $\text{grid-estimate}[of \text{ } d \text{ start } dm \text{ } p \{0 \dots dm\}]$ by auto

lemma $ix\text{-gt}$: $p \in \text{sparsegrid } dm \text{ } lm \implies d < dm \implies 0 < ix \text{ } p \text{ } d$
 unfolding $\text{sparsegrid-def lgrid-def}$
 using $\text{grid-estimate}[of \text{ } d \text{ start } dm \text{ } p \{0 \dots dm\}]$ by auto

lemma $\Phi\text{-eq-0}$: assumes $x: \exists d < \text{length } p. x \text{ } d < 0 \vee 1 < x \text{ } d$ and $p: p \in \text{sparsegrid } dm \text{ } lm$ shows $\Phi \text{ } p \text{ } x = 0$
 unfolding $\Phi\text{-def}$
 proof (rule prod-zero)
 from x guess $d \dots$
 with $p[THEN \text{ } ix\text{-lt}, of \text{ } d] \text{ } p[THEN \text{ } ix\text{-gt}, of \text{ } d] \text{ } p$
 show $\exists a \in \{0 \dots \text{length } p\}. \varphi (p \text{ } ! \text{ } a) (x \text{ } a) = 0$
 apply (cases $p \text{ } ! \text{ } d$)
 apply (intro $\text{beqI}[of \text{ } - \text{ } d]$)
 apply (auto intro!: $\varphi\text{-eq-0 simp: sparsegrid-length ix-def lv-def}$)
 done
 qed simp

lemma $\varphi\text{-left-support}'$:
 $x \in \{\text{real-of-int } (i - 1) / 2^{l+1} \dots \text{real-of-int } i / 2^{l+1}\} \implies \varphi (l, i) \text{ } x = 1 + x * 2^{l+1} - \text{real-of-int } i$
 by (auto simp: $\varphi\text{-def field-simps split: split-max}$)

lemma $\varphi\text{-left-support}$: $x \in \{-1 \dots 0::\text{real}\} \implies \varphi (l, i) ((x + \text{real-of-int } i) / 2^{l+1}) = 1 + x$
 by (auto simp: $\varphi\text{-def field-simps split: split-max}$)

lemma $\varphi\text{-right-support}'$:
 $x \in \{\text{real-of-int } i / 2^{l+1} \dots \text{real-of-int } (i + 1) / 2^{l+1}\} \implies \varphi (l, i) \text{ } x = 1 - x * 2^{l+1} + \text{real-of-int } i$
 by (auto simp: $\varphi\text{-def field-simps split: split-max}$)

lemma $\varphi\text{-right-support}$:
 $x \in \{0 \dots 1::\text{real}\} \implies \varphi (l, i) ((x + \text{real } i) / 2^{l+1}) = 1 - x$
 by (auto simp: $\varphi\text{-def field-simps split: split-max}$)

lemma $\text{integrable-}\varphi$: $\text{integrable lborel } (\varphi \text{ } p)$

proof (*induct p*)
case (*Pair l i*)
have *integrable lborel* ($\lambda x. \text{indicator } \{ \text{real-of-int } (i - 1) / 2^{\wedge}(l + 1) .. \text{real-of-int } (i + 1) / 2^{\wedge}(l + 1) \} x *_R \varphi (l, i) x$)
unfolding $\varphi\text{-def}$ **by** (*intro borel-integrable-compact*) (*auto intro!: continuous-intros*)
then show *?case*
by (*rule integrable-cong[THEN iffD1, rotated -1]*) (*auto simp: φ -zero-iff*)
qed

lemma *integrable- $\varphi 2$* : *integrable lborel* ($\lambda x. \varphi p x * \varphi q x$)
proof (*cases p q rule: prod.exhaust[case-product prod.exhaust]*)
case (*Pair-Pair l i l' i'*)
have *integrable lborel*
 $(\lambda x. \text{indicator } \{ \text{real-of-int } (i - 1) / 2^{\wedge}(l + 1) .. \text{real-of-int } (i + 1) / 2^{\wedge}(l + 1) \} x *_R (\varphi (l, i) x * \varphi (l', i') x))$
unfolding $\varphi\text{-def}$ **by** (*intro borel-integrable-compact*) (*auto intro!: continuous-intros*)
then show *?thesis unfolding Pair-Pair*
by (*rule integrable-cong[THEN iffD1, rotated -1]*) (*auto simp: φ -zero-iff*)
qed

lemma *l2- φ I-DERIV*:
assumes $n: \bigwedge x. x \in \{ \text{real-of-int } i' - 1 / 2^{\wedge}(l' + 1) .. \text{real-of-int } i' / 2^{\wedge}(l' + 1) \} \implies$
 $DERIV \Phi\text{-}n x := (\varphi (l', i') x * \varphi (l, i) x) \text{ (is } \bigwedge x. x \in \{ ?a..?b \} \implies DERIV -$
 $- := ?P x)$
and $p: \bigwedge x. x \in \{ \text{real-of-int } i' / 2^{\wedge}(l' + 1) .. \text{real-of-int } i' + 1 / 2^{\wedge}(l' + 1) \} \implies$
 $DERIV \Phi\text{-}p x := (\varphi (l', i') x * \varphi (l, i) x) \text{ (is } \bigwedge x. x \in \{ ?b..?c \} \implies -)$
shows $l2\text{-}\varphi (l', i') (l, i) = (\Phi\text{-}n ?b - \Phi\text{-}n ?a) + (\Phi\text{-}p ?c - \Phi\text{-}p ?b)$
proof -
have *has-bochner-integral lborel*
 $(\lambda x. ?P x * \text{indicator } \{ ?a..?b \} x + ?P x * \text{indicator } \{ ?b..?c \} x)$
 $((\Phi\text{-}n ?b - \Phi\text{-}n ?a) + (\Phi\text{-}p ?c - \Phi\text{-}p ?b))$
by (*intro has-bochner-integral-add has-bochner-integral-FTC-Icc-nonneg n p*)
(auto simp: φ -nonneg field-simps)
then have *has-bochner-integral lborel?* $((\Phi\text{-}n ?b - \Phi\text{-}n ?a) + (\Phi\text{-}p ?c - \Phi\text{-}p ?b))$
by (*rule has-bochner-integral-discrete-difference[where X={?b}, THEN iffD1, rotated -1]*)
(auto simp: power-add intro!: φ -zero integral-cong split: split-indicator)
then show *?thesis by (simp add: has-bochner-integral-iff l2- φ -def)*
qed

lemma *l2-eq*: $\text{length } a = \text{length } b \implies l2 a b = (\prod d < \text{length } a. l2\text{-}\varphi (a!d) (b!d))$
unfolding *l2-def l2- φ -def Φ -def*
apply (*simp add: prod.distrib[symmetric]*)
proof (*rule product-sigma-finite.product-integral-prod*)
show *product-sigma-finite* ($\lambda d. \text{lborel}$) ..
qed (*auto intro: integrable- $\varphi 2$*)

lemma *l2-when-disjoint*:
assumes $l \leq l'$
defines $d == l' - l$
assumes $(i + 1) * 2^d < i' \vee i' < (i - 1) * 2^d$ (**is** *?right* \vee *?left*)
shows $l2\text{-}\varphi(l', i')(l, i) = 0$
proof –
let $?sup = \lambda l i. \{real\text{-of-int}(i - 1) / 2^{l+1} <..< real\text{-of-int}(i + 1) / 2^{l+1}\}$

have $l': l' = l + d$
using *assms* **by** *simp*
have $*$: $\bigwedge i l. 2^l = real\text{-of-int}(2^l :: int)$
by *simp*
have [*arith*]: $0 < (2^d :: int)$
by *simp*

from (*?right* \vee *?left*) $(l \leq l')$ **have** *empty-support*: $?sup\ l\ i \cap ?sup\ l'\ i' = \{\}$
by (*auto simp add: min-def max-def divide-simps l' power-add * of-int-mult[symmetric]*
simp del: of-int-diff of-int-add of-int-mult of-int-power)
(simp-all add: field-simps)
then have $\bigwedge x. \varphi(l', i')\ x * \varphi(l, i)\ x = 0$
unfolding *φ -zero-iff mult-eq-0-iff* **by** *blast*
then show *?thesis*
by (*simp add: l2- φ -def del: mult-eq-0-iff vector-space-over-itself.scale-eq-0-iff*)
qed

lemma *l2-commutative*: $l2\text{-}\varphi\ p\ q = l2\text{-}\varphi\ q\ p$
by (*simp add: l2- φ -def mult.commute*)

lemma *l2-when-same*: $l2\text{-}\varphi(l, i)(l, i) = 1/3 / 2^l$

proof (*subst l2- φ I-DERIV*)
let $?l = (2 :: real)^{l+1}$
let $?in = real\text{-of-int}\ i - 1$
let $?ip = real\text{-of-int}\ i + 1$
let $?\varphi = \varphi(l, i)$
let $?\varphi2 = \lambda x. ?\varphi\ x * ?\varphi\ x$

{ fix x **assume** $x \in \{?in / ?l .. real\text{-of-int}\ i / ?l\}$
hence *φ -eq*: $?\varphi\ x = ?l * x - ?in$ **using** *φ -left-support'* **by** *auto*
show *DERIV* $(\lambda x. x^3 / 3 * ?l^2 + x * ?in^2 - x^2 / 2 * 2 * ?l * ?in)\ x :=$
 $?\varphi2\ x$
by (*auto intro!: derivative-eq-intros simp add: power2-eq-square field-simps*
 φ -eq) **}**

{ fix x **assume** $x \in \{real\text{-of-int}\ i / ?l .. ?ip / ?l\}$
hence *φ -eq*: $?\varphi\ x = ?ip - ?l * x$ **using** *φ -right-support'* **by** *auto*
show *DERIV* $(\lambda x. x^3 / 3 * ?l^2 + x * ?ip^2 - x^2 / 2 * 2 * ?l * ?ip)\ x :=$
 $?\varphi2\ x$

by (auto intro!: derivative-eq-intros simp add: power2-eq-square field-simps φ -eq) }

qed (simp-all add: field-simps power-eq-if[of - 2] power-eq-if[of - 3])

lemma *l2-when-left-child*:

assumes $l < l'$

and *i'-bot*: $i' > (i - 1) * 2^{l' - l}$

and *i'-top*: $i' < i * 2^{l' - l}$

shows $l2\text{-}\varphi(l', i')(l, i) = (1 + \text{real-of-int } i' / 2^{l' - l} - \text{real-of-int } i) / 2^{l' + 1}$

proof (subst *l2- φ I-DERIV*)

let $?l' = (2 :: \text{real})^{l' + 1}$

let $?in' = \text{real-of-int } i' - 1$

let $?ip' = \text{real-of-int } i' + 1$

let $?l = (2 :: \text{real})^l$

let $?i = \text{real-of-int } i - 1$

let $?varphi' = \varphi(l', i')$

let $?varphi = \varphi(l, i)$

let $?varphi2\ x = ?varphi' x * ?varphi x$

define $\Phi\text{-}n$ **where** $\Phi\text{-}n\ x = x^3 / 3 * ?l' * ?l + x * ?i * ?in' - x^2 / 2 * (?in' * ?l + ?i * ?l')$ **for** x

define $\Phi\text{-}p$ **where** $\Phi\text{-}p\ x = x^2 / 2 * (?ip' * ?l + ?i * ?l') - x^3 / 3 * ?l' * ?l - x * ?i * ?ip'$ **for** x

have *level-diff*: $2^{l' - l} = 2^{l'} / (2^l :: \text{real})$ **using** *power-diff*[of $2 :: \text{real } l\ l'$] $l < l'$ **by** *auto*

{ **fix** x **assume** $x: x \in \{?in' / ?l' .. ?ip' / ?l'\}$

have $?i * 2^{l' - l} \leq ?in'$

using *i'-bot int-less-real-le* **by** *auto*

hence $?i / ?l \leq ?in' / ?l'$

using *level-diff* **by** (auto simp: field-simps)

hence $?i / ?l \leq x$ **using** x **by** *auto*

moreover

have $?ip' \leq \text{real-of-int } i * 2^{l' - l}$

using *i'-top int-less-real-le* **by** *auto*

hence *ip'-le-i*: $?ip' / ?l' \leq \text{real-of-int } i / ?l$

using *level-diff* **by** (auto simp: field-simps)

hence $x \leq \text{real-of-int } i / ?l$ **using** x **by** *auto*

ultimately have $?varphi\ x = ?l * x - ?i$ **using** *φ -left-support'* **by** *auto*

} **note** $\varphi\text{-eq} = \text{this}$

{ **fix** x **assume** $x: x \in \{?in' / ?l' .. \text{real-of-int } i' / ?l'\}$

hence $\varphi'\text{-eq}$: $?varphi' x = ?l' * x - ?in'$ **using** *φ -left-support'* **by** *auto*

from x **have** x' : $x \in \{?in' / ?l' .. ?ip' / ?l'\}$ **by** (auto simp add: field-simps)

show *DERIV* $\Phi\text{-}n\ x :> ?varphi2\ x$ **unfolding** $\varphi\text{-eq}[OF\ x]$ $\varphi'\text{-eq}\ \Phi\text{-}n\text{-def}$

by (auto intro!: derivative-eq-intros simp add: power2-eq-square algebra-simps)

}

```

{ fix x assume x: x ∈ {real-of-int i' / ?l' .. ?ip' / ?l'}
  hence φ'-eq: ?φ' x = ?ip' - ?l' * x using φ-right-support' by auto
  from x have x': x ∈ {?in' / ?l' .. ?ip' / ?l'} by (simp add: field-simps)
  show DERIV Φ-p x :> ?φ2 x unfolding φ-eq[OF x'] φ'-eq Φ-p-def
    by (auto intro!: derivative-eq-intros simp add: power2-eq-square algebra-simps)
}
qed (simp-all add: field-simps power-eq-if[of - 2] power-eq-if[of - 3] power-diff[of
2::real, OF - <l <l'[THEN less-imp-le]])

```

lemma *l2-when-right-child*:

```

assumes l < l'
and i'-bot: i' > i * 2^(l' - l)
and i'-top: i' < (i + 1) * 2^(l' - l)
shows l2-φ (l', i') (l, i) = (1 - real-of-int i' / 2^(l' - l) + real-of-int i) / 2^(l'
+ 1)

```

proof (*subst l2-φI-DERIV*)

```

let ?l' = (2 :: real)^(l' + 1)
let ?in' = real-of-int i' - 1
let ?ip' = real-of-int i' + 1
let ?l = (2 :: real)^l
let ?i = real-of-int i + 1
let ?φ' = φ (l', i')
let ?φ = φ (l, i)
let ?φ2 = λx. ?φ' x * ?φ x
define Φ-n where Φ-n x = x^2 / 2 * (?in' * ?l + ?i * ?l') - x^3 / 3 * ?l' *
?l - x * ?i * ?in' for x
define Φ-p where Φ-p x = x^3 / 3 * ?l' * ?l + x * ?i * ?ip' - x^2 / 2 * (?ip'
* ?l + ?i * ?l') for x

```

```

have level-diff: 2^(l' - l) = 2^l' / (2^l :: real) using power-diff[of 2::real l l'] l
< l' by auto

```

```

{ fix x assume x: x ∈ {?in' / ?l' .. ?ip' / ?l'}
  have real-of-int i * 2^(l' - l) ≤ ?in'
    using i'-bot int-less-real-le by auto
  hence real-of-int i / ?l ≤ ?in' / ?l'
    using level-diff by (auto simp: field-simps)
  hence real-of-int i / ?l ≤ x using x by auto
  moreover
  have ?ip' ≤ ?i * 2^(l' - l)
    using i'-top int-less-real-le by auto
  hence ip'-le-i: ?ip' / ?l' ≤ ?i / ?l
    using level-diff by (auto simp: field-simps)
  hence x ≤ ?i / ?l using x by auto
  ultimately have ?φ x = ?i - ?l * x using φ-right-support' by auto
} note φ-eq = this

```

```

{ fix x assume x: x ∈ {?in' / ?l' .. real-of-int i' / ?l'}
  hence φ'-eq: ?φ' x = ?l' * x - ?in' using φ-left-support' by auto

```

from x **have** x' : $x \in \{?in' / ?l' .. ?ip' / ?l'\}$ **by** (*simp add: field-simps*)
show $DERIV \Phi\text{-}n$ x $:$ $>$ $? \varphi 2$ x **unfolding** $\Phi\text{-}n\text{-}def$ $\varphi\text{-}eq[OF x']$ $\varphi'\text{-}eq$
by (*auto intro!: derivative-eq-intros simp add: simp add: power2-eq-square algebra-simps*) }

{ **fix** x **assume** x : $x \in \{real\text{-}of\text{-}int i' / ?l' .. ?ip' / ?l'\}$
hence $\varphi'\text{-}eq$: $? \varphi' x = ?ip' - ?l' * x$ **using** $\varphi\text{-}right\text{-}support'$ **by** *auto*
from x **have** x' : $x \in \{?in' / ?l' .. ?ip' / ?l'\}$ **by** (*auto simp: field-simps*)
show $DERIV \Phi\text{-}p$ x $:$ $>$ $? \varphi 2$ x **unfolding** $\varphi\text{-}eq[OF x']$ $\varphi'\text{-}eq$ $\Phi\text{-}p\text{-}def$
by (*auto intro!: derivative-eq-intros simp add: power2-eq-square algebra-simps*)
}

qed (*simp-all add: field-simps power-eq-if[of - 2] power-eq-if[of - 3] power-diff[of 2::real, OF - <l <l'[THEN less-imp-le]]*)

lemma *level-shift*: $lc > l \implies (x :: real) / 2^{lc - Suc l} = x * 2 / 2^{lc - l}$
by (*auto simp add: power-diff*)

lemma *l2-child*: **assumes** $d < length\ b$
and $p\text{-}grid$: $p \in grid\ (child\ b\ dir\ d)\ ds$ (**is** $p \in grid\ ?child\ ds$)
shows $l2\text{-}\varphi\ (p\ !\ d)\ (b\ !\ d) = (1 - real\text{-}of\text{-}int\ (sgn\ dir) * (real\text{-}of\text{-}int\ (ix\ p\ d) / 2^{lv\ p\ d - lv\ b\ d} - real\text{-}of\text{-}int\ (ix\ b\ d))) / 2^{lv\ p\ d + 1}$

proof –

have $lv\ ?child\ d \leq lv\ p\ d$ **using** $\langle d < length\ b \rangle$ **and** $p\text{-}grid$
using *grid-single-level* **by** *auto*
hence $lv\ b\ d < lv\ p\ d$ **using** $\langle d < length\ b \rangle$ **and** $p\text{-}grid$
using *child-lv* **by** *auto*

let $?i\text{-}c = ix\ ?child\ d$ **and** $?l\text{-}c = lv\ ?child\ d$
let $?i\text{-}p = ix\ p\ d$ **and** $?l\text{-}p = lv\ p\ d$
let $?i\text{-}b = ix\ b\ d$ **and** $?l\text{-}b = lv\ b\ d$

have $(2::int) * 2^{?l\text{-}p - ?l\text{-}c} = 2^{Suc\ (?l\text{-}p - ?l\text{-}c)}$ **by** *auto*
also **have** $\dots = 2^{Suc\ ?l\text{-}p - ?l\text{-}c}$

proof –

have $Suc\ (?l\text{-}p - ?l\text{-}c) = Suc\ ?l\text{-}p - ?l\text{-}c$
using $\langle lv\ ?child\ d \leq lv\ p\ d \rangle$ **by** *auto*
thus *thesis* **by** *auto*

qed

also **have** $\dots = 2^{?l\text{-}p - ?l\text{-}b}$
using $\langle d < length\ b \rangle$ **and** $\langle lv\ b\ d < lv\ p\ d \rangle$
by (*auto simp add: child-def lv-def*)

finally **have** *level*: $2^{?l\text{-}p - ?l\text{-}b} = (2::int) * 2^{?l\text{-}p - ?l\text{-}c} ..$

from $\langle d < length\ b \rangle$ **and** $p\text{-}grid$

have *range-left*: $?i\text{-}p > (?i\text{-}c - 1) * 2^{?l\text{-}p - ?l\text{-}c}$ **and**
range-right: $?i\text{-}p < (?i\text{-}c + 1) * 2^{?l\text{-}p - ?l\text{-}c}$
using *grid-estimate* **by** *auto*

show *?thesis*
proof (*cases dir*)
 case *left*
 with *child-ix-left*[*OF* $\langle d < \text{length } b \rangle$]
 have $(?i-b - 1) * 2^{(?l-p - ?l-b)} = (?i-c - 1) * 2^{(?l-p - ?l-c)}$ **and**
 $?i-b * 2^{(?l-p - ?l-b)} = (?i-c + 1) * 2^{(?l-p - ?l-c)}$ **using** *level by auto*
 hence $?i-p > (?i-b - 1) * 2^{(?l-p - ?l-b)}$ **and**
 $?i-p < ?i-b * 2^{(?l-p - ?l-b)}$
 using *range-left and range-right by auto*
 with $\langle ?l-b < ?l-p \rangle$
 have $l2-\varphi (?l-p, ?i-p) (?l-b, ?i-b) =$
 $(1 + \text{real-of-int } ?i-p / 2^{(?l-p - ?l-b)} - \text{real-of-int } ?i-b) / 2^{(?l-p + 1)}$
 by (*rule l2-when-left-child*)
 thus *?thesis using left by (auto simp add: ix-def lv-def)*
 next
 case *right*
 hence $?i-c = 2 * ?i-b + 1$ **using** *child-ix-right and* $\langle d < \text{length } b \rangle$ **by** *auto*
 hence $?i-b * 2^{(?l-p - ?l-b)} = (?i-c - 1) * 2^{(?l-p - ?l-c)}$ **and**
 $(?i-b + 1) * 2^{(?l-p - ?l-b)} = (?i-c + 1) * 2^{(?l-p - ?l-c)}$ **using** *level by*
auto
 hence $?i-p > ?i-b * 2^{(?l-p - ?l-b)}$ **and**
 $?i-p < (?i-b + 1) * 2^{(?l-p - ?l-b)}$
 using *range-left and range-right by auto*
 with $\langle ?l-b < ?l-p \rangle$
 have $l2-\varphi (?l-p, ?i-p) (?l-b, ?i-b) =$
 $(1 - \text{real-of-int } ?i-p / 2^{(?l-p - ?l-b)} + \text{real-of-int } ?i-b) / 2^{(?l-p + 1)}$
 by (*rule l2-when-right-child*)
 thus *?thesis using right by (auto simp add: ix-def lv-def)*
 qed
qed

lemma *l2-same*: $l2-\varphi (p!d) (p!d) = 1/3 / 2^{(lv\ p\ d)}$
proof –
 have $l2-\varphi (p!d) (p!d) = l2-\varphi (lv\ p\ d, ix\ p\ d) (lv\ p\ d, ix\ p\ d)$
 by (*auto simp add: lv-def ix-def*)
 thus *?thesis using l2-when-same by auto*
qed

lemma *l2-disjoint*: **assumes** $d < \text{length } b$ **and** $p \in \text{grid } b \{d\}$ **and** $p' \in \text{grid } b \{d\}$
and $p' \notin \text{grid } p \{d\}$ **and** $lv\ p'\ d \geq lv\ p\ d$
shows $l2-\varphi (p'\ !\ d) (p'\ !\ d) = 0$
proof –
 have *range*: $ix\ p'\ d > (ix\ p\ d + 1) * 2^{(lv\ p'\ d - lv\ p\ d)} \vee ix\ p'\ d < (ix\ p\ d - 1) * 2^{(lv\ p'\ d - lv\ p\ d)}$
 proof (*rule ccontr*)
 assume $\neg ?thesis$
 hence $ix\ p'\ d \leq (ix\ p\ d + 1) * 2^{(lv\ p'\ d - lv\ p\ d)}$ **and** $ix\ p'\ d \geq (ix\ p\ d - 1) * 2^{(lv\ p'\ d - lv\ p\ d)}$ **by** *auto*

with $\langle p' \in \text{grid } b \{d\} \rangle$ **and** $\langle p \in \text{grid } b \{d\} \rangle$ **and** $\langle lv \ p' \ d \geq lv \ p \ d \rangle$ **and** $\langle d < \text{length } b \rangle$
have $p' \in \text{grid } p \{d\}$ **using** *grid-part*[**where** $p=p$ **and** $b=b$ **and** $d=d$ **and** $p'=p'$] **by** *auto*
with $\langle p' \notin \text{grid } p \{d\} \rangle$ **show** *False* **by** *auto*
qed

have $l2-\varphi \ (p' ! d) \ (p ! d) = l2-\varphi \ (lv \ p' \ d, \ ix \ p' \ d) \ (lv \ p \ d, \ ix \ p \ d)$ **by** (*auto simp add: ix-def lv-def*)
also have $\dots = 0$ **using** *range* **and** $\langle lv \ p' \ d \geq lv \ p \ d \rangle$ **and** *l2-when-disjoint* **by** *auto*
finally show *?thesis* .
qed

lemma *l2-down2*:

fixes $pc \ pd \ p$
assumes $d < \text{length } pd$
assumes *pc-in-grid*: $pc \in \text{grid} \ (\text{child } pd \ \text{dir } d) \ \{d\}$
assumes *pd-is-child*: $pd = \text{child } p \ \text{dir } d$ (**is** $pd = ?pd$)
shows $l2-\varphi \ (pc ! d) \ (pd ! d) / 2 = l2-\varphi \ (pc ! d) \ (p ! d)$

proof –

have $d < \text{length } p$ **using** *pd-is-child* $\langle d < \text{length } pd \rangle$ **by** *auto*

moreover

have $pc \in \text{grid } ?pd \ \{d\}$ **using** *pd-is-child* **and** *grid-child* **and** *pc-in-grid* **by** *auto*
hence $lv \ p \ d < lv \ pc \ d$ **using** *grid-child-level* **and** $\langle d < \text{length } pd \rangle$ **and** *pd-is-child* **by** *auto*

moreover

have $\text{real-of-int} \ (\text{sgn } dir) * \text{real-of-int} \ (\text{sgn } dir) = 1$ **by** (*cases dir, auto*)

ultimately show *?thesis*

unfolding *l2-child*[*OF* $\langle d < \text{length } pd \rangle$ *pc-in-grid*]
 $l2-\text{child}$ [*OF* $\langle d < \text{length } p \rangle$ $\langle pc \in \text{grid } ?pd \ \{d\} \rangle$]
using *child-lv* **and** *child-ix* **and** *pd-is-child* **and** *level-shift*
by (*auto simp add: algebra-simps diff-divide-distrib add-divide-distrib*)

qed

lemma *l2-zigzag*:

assumes $d < \text{length } p$ **and** *p-child*: $p = \text{child } p \ \text{dir } d$
and *p'-grid*: $p' \in \text{grid} \ (\text{child } p \ (\text{inv } dir) \ d) \ \{d\}$
and *ps-intro*: $\text{child } p \ (\text{inv } dir) \ d = \text{child } ps \ \text{dir } d$ (**is** $?c-p = ?c-ps$)
shows $l2-\varphi \ (p' ! d) \ (p-p ! d) = l2-\varphi \ (p' ! d) \ (ps ! d) + l2-\varphi \ (p' ! d) \ (p ! d) / 2$

proof –

have $\text{length } p = \text{length } ?c-p$ **by** *auto*

also have $\dots = \text{length } ?c-ps$ **using** *ps-intro* **by** *auto*

finally have $\text{length } p = \text{length } ps$ **using** *ps-intro* **by** *auto*

hence $d < \text{length } p-p$ **using** *p-child* **and** $\langle d < \text{length } p \rangle$ **by** *auto*

moreover
from *ps-intro* **have** $ps = p[d := (lv\ p\ d, ix\ p\ d - sgn\ dir)]$ **by** (*rule child-neighbour*)
hence $lv\ ps\ d = lv\ p\ d$ **and** $real-of-int\ (ix\ ps\ d) = real-of-int\ (ix\ p\ d) - real-of-int\ (sgn\ dir)$
using *lv-def* **and** *ix-def* **and** $\langle length\ p = length\ ps \rangle$ **and** $\langle d < length\ p \rangle$ **by** *auto*

moreover
have $d < length\ ps$ **and** $*: p' \in grid\ (child\ ps\ dir\ d)\ \{d\}$
using *p'-grid ps-intro* $\langle length\ p = length\ ps \rangle$ $\langle d < length\ p \rangle$ **by** *auto*

have $p' \in grid\ p\ \{d\}$ **using** *p'-grid* **and** *grid-child* **by** *auto*
hence *p-p-grid*: $p' \in grid\ (child\ p-p\ dir\ d)\ \{d\}$ **using** *p-child* **by** *auto*
hence $lv\ p'\ d > lv\ p-p\ d$ **using** *grid-child-level* **and** $\langle d < length\ p-p \rangle$ **by** *auto*

moreover
have $real-of-int\ (sgn\ dir) * real-of-int\ (sgn\ dir) = 1$ **by** (*cases dir, auto*)

ultimately show *?thesis*
unfolding *l2-child*[*OF* $\langle d < length\ p \rangle$ *p'-grid*] *l2-child*[*OF* $\langle d < length\ ps \rangle$ *]
l2-child[*OF* $\langle d < length\ p-p \rangle$ *p-p-grid*]
using *child-lv* **and** *child-ix* **and** *p-child level-shift*
by (*auto simp add: add-divide-distrib algebra-simps diff-divide-distrib*)

qed

end

4 UpDown Scheme

theory *UpDown-Scheme*

imports *Grid*

begin

fun *down'* :: $nat \Rightarrow nat \Rightarrow grid-point \Rightarrow real \Rightarrow real \Rightarrow vector \Rightarrow vector$

where

$down'\ d\ 0\ p\ f_l\ f_r\ \alpha = \alpha$
 $| down'\ d\ (Suc\ l)\ p\ f_l\ f_r\ \alpha = (let$
 $\quad f_m = (f_l + f_r) / 2 + (\alpha\ p);$
 $\quad \alpha = \alpha(p := ((f_l + f_r) / 4 + (1 / 3) * (\alpha\ p)) / 2 \wedge (lv\ p\ d));$
 $\quad \alpha = down'\ d\ l\ (child\ p\ left\ d)\ f_l\ f_m\ \alpha;$
 $\quad \alpha = down'\ d\ l\ (child\ p\ right\ d)\ f_m\ f_r\ \alpha$
 $in\ \alpha)$

definition *down* :: $nat \Rightarrow nat \Rightarrow nat \Rightarrow vector \Rightarrow vector$ **where**

$down = lift\ (\lambda\ d\ l\ p.\ down'\ d\ l\ p\ 0\ 0)$

fun *up'* :: $nat \Rightarrow nat \Rightarrow grid-point \Rightarrow vector \Rightarrow (real * real) * vector$ **where**

$up'\ d\ 0\ p\ \alpha = ((0, 0), \alpha)$
 $| up'\ d\ (Suc\ l)\ p\ \alpha = (let$
 $\quad (f_l, f_m), \alpha = up'\ d\ l\ (child\ p\ left\ d)\ \alpha;$

$$\begin{aligned} & ((f_{mr}, f_r), \alpha) = \text{up}' d l (\text{child } p \text{ right } d) \alpha; \\ & \text{result} = (f_{ml} + f_{mr} + (\alpha p) / 2 \wedge (lv p d) / 2) / 2 \\ & \text{in } ((f_l + \text{result}, f_r + \text{result}), \alpha(p := f_{ml} + f_{mr})) \end{aligned}$$

definition $\text{up} :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{vector} \Rightarrow \text{vector}$ **where**
 $\text{up} = \text{lift } (\lambda d \text{ lm } p \alpha. \text{snd } (\text{up}' d \text{ lm } p \alpha))$

fun $\text{updown}' :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{vector} \Rightarrow \text{vector}$ **where**
 $\text{updown}' dm \text{ lm } 0 \alpha = \alpha$
 $| \text{updown}' dm \text{ lm } (\text{Suc } d) \alpha =$
 $(\text{sum-vector } (\text{updown}' dm \text{ lm } d (\text{up } dm \text{ lm } d \alpha)) (\text{down } dm \text{ lm } d (\text{updown}' dm$
 $\text{lm } d \alpha)))$

definition $\text{updown} :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{vector} \Rightarrow \text{vector}$ **where**
 $\text{updown } dm \text{ lm } \alpha = \text{updown}' dm \text{ lm } dm \alpha$

end

5 Up Part

theory Up
imports $UpDown\text{-Scheme } Triangular\text{-Function}$
begin

lemma $\text{up}'\text{-inplace}$:

assumes $p'\text{-in}$: $p' \notin \text{grid } p \text{ ds}$ **and** $d \in \text{ds}$
shows $\text{snd } (\text{up}' d l p \alpha) p' = \alpha p'$
using $p'\text{-in}$

proof ($\text{induct } l \text{ arbitrary}$: $p \alpha$)

case $(\text{Suc } l)$
let $?ch \text{ dir} = \text{child } p \text{ dir } d$
let $?up \text{ dir } \alpha = \text{up}' d l (?ch \text{ dir}) \alpha$
let $?upl = \text{snd } (?up \text{ left } \alpha)$

from $\text{contrapos-nn}[OF \langle p' \notin \text{grid } p \text{ ds} \rangle \text{grid-child}[OF \langle d \in \text{ds} \rangle]]$

have left : $p' \notin \text{grid } (?ch \text{ left}) \text{ ds}$ **and**
 right : $p' \notin \text{grid } (?ch \text{ right}) \text{ ds}$ **by auto**

have $p \neq p'$ **using** $\text{grid.Start Suc.prem}$ **by auto**

with $\text{Suc.hyps}[OF \text{left}, \text{of } \alpha]$ $\text{Suc.hyps}[OF \text{right}, \text{of } ?upl]$

show $?case$

by ($\text{cases } ?up \text{ left } \alpha, \text{cases } ?up \text{ right } ?upl, \text{auto simp add: Let-def}$)

qed auto

lemma $\text{up}'\text{-fl-fr}$:

$$\begin{aligned} & \llbracket d < \text{length } p ; p = (\text{child } p\text{-r right } d) ; p = (\text{child } p\text{-l left } d) \rrbracket \\ & \implies \text{fst } (\text{up}' d \text{ lm } p \alpha) = \\ & \quad (\sum p' \in \text{lgrid } p \{d\} (\text{lm} + \text{level } p). (\alpha p') * \text{l2-}\varphi (p' ! d) (p\text{-r} ! d), \end{aligned}$$

$\sum p' \in \text{lgrid } p \{d\} (lm + \text{level } p). (\alpha p') * \text{l2-}\varphi (p' ! d) (p-l ! d)$

proof (*induct lm arbitrary: p p-l p-r α*)
case (*Suc lm*)
note $\langle d < \text{length } p \rangle [\text{simp}]$

from *child-ex-neighbour*
obtain *pc-r pc-l*
where *pc-r-def: child p right d = child pc-r (inv right) d*
and *pc-l-def: child p left d = child pc-l (inv left) d by blast*

define *pc* **where** *pc dir = (case dir of right \Rightarrow pc-r | left \Rightarrow pc-l) for dir*
{ fix dir have *child p (inv dir) d = child (pc (inv dir)) dir d*
by (*cases dir, auto simp add: pc-def pc-r-def pc-l-def*) **} note** *pc-child = this*
{ fix dir have *child p dir d = child (pc dir) (inv dir) d*
by (*cases dir, auto simp add: pc-def pc-r-def pc-l-def*) **} note** *pc-child-inv = this*

hence $!! \text{ dir. length (child p dir d) = length (child (pc dir) (inv dir) d)$ **by** *auto*
hence $!! \text{ dir. length } p = \text{length (pc dir)}$ **by** *auto*
hence $[\text{simp}]: !! \text{ dir. } d < \text{length (pc dir)}$ **by** *auto*

let $?l = \lambda s. lm + \text{level } s$
let $?C = \lambda p p'. (\alpha p) * \text{l2-}\varphi (p ! d) (p' ! d)$
let $?sum' = \lambda s p'. \sum p' \in \text{lgrid } s \{d\} (\text{Suc } lm + \text{level } p). ?C p' p''$
let $?sum = \lambda s \text{ dir } p. \sum p' \in \text{lgrid (child } s \text{ dir } d) \{d\} (?l (\text{child } s \text{ dir } d)). ?C p' p$
let $?ch = \lambda \text{ dir. child } p \text{ dir } d$
let $?f = \lambda \text{ dir. ?sum } p \text{ dir (pc dir)}$
let $?fm = \lambda \text{ dir. ?sum } p \text{ dir } p$
let $?result = (?fm \text{ left} + ?fm \text{ right} + (\alpha p) / 2 \wedge (lv p d) / 2) / 2$
let $?up = \lambda lm p \alpha. up' d lm p \alpha$

define βl **where** $\beta l = \text{snd} (?up lm (?ch \text{ left}) \alpha)$
define βr **where** $\beta r = \text{snd} (?up lm (?ch \text{ right}) \beta l)$

define *p-d* **where** *p-d dir = (case dir of right \Rightarrow p-r | left \Rightarrow p-l) for dir*
have *p-d-child: p = child (p-d dir) dir d for dir*
using *Suc.premis p-d-def by (cases dir) auto*
hence $\wedge \text{ dir. length } p = \text{length (child (p-d dir) dir d)}$ **by** *auto*
hence $\wedge \text{ dir. } d < \text{length (p-d dir)}$ **by** *auto*

{ fix dir

{ fix p' assume $p' \in \text{lgrid} (?ch (\text{inv dir})) \{d\} (?l (?ch (\text{inv dir})))$
hence $?C p' (pc (\text{inv dir})) + (?C p' p) / 2 = ?C p' (p-d \text{ dir})$
using *l2-zigzag[OF - p-d-child[of dir] - pc-child[of dir]]*
by (*cases dir (auto simp add: algebra-simps)*) **}**

hence *inv-dir-sum: ?sum p (inv dir) (pc (inv dir)) + (?sum p (inv dir) p) / 2*
 $= ?sum p (\text{inv dir}) (p-d \text{ dir})$
by (*auto simp add: sum.distrib[symmetric] sum-divide-distrib*)

have $?sum\ p\ dir\ p / 2 = ?sum\ p\ dir\ (p-d\ dir)$
using $l2-down2[OF\ -\ -\ \langle p = child\ (p-d\ dir)\ dir\ d \rangle]$
by (*force intro! sum.cong simp add: sum-divide-distrib*)
moreover
have $?C\ p\ (p-d\ dir) = (\alpha\ p) / 2 \wedge (lv\ p\ d) / 4$
using $l2-child[OF\ \langle d < length\ (p-d\ dir) \rangle, of\ p\ dir\ \{d\}] p-d-child[of\ dir]$
 $\langle d < length\ (p-d\ dir) \rangle child-lv\ child-ix\ grid.Start[of\ p\ \{d\}]$
by (*cases dir*) (*auto simp add: add-divide-distrib field-simps*)
ultimately
have $?sum' p\ (p-d\ dir) =$
 $?sum\ p\ (inv\ dir)\ (pc\ (inv\ dir)) +$
 $(?sum\ p\ (inv\ dir)\ p) / 2 + ?sum\ p\ dir\ p / 2 + (\alpha\ p) / 2 \wedge (lv\ p\ d) / 4$
using $lgrid-sum[where\ b=p]$ **and** $child-level$ **and** $inv-dir-sum$
by (*cases dir*) *auto*
hence $?sum\ p\ (inv\ dir)\ (pc\ (inv\ dir)) + ?result = ?sum' p\ (p-d\ dir)$
by (*cases dir*) *auto* }
note $this[of\ left]\ this[of\ right]$
moreover
note $eq = up'-inplace[OF\ grid-not-child[OF\ \langle d < length\ p \rangle], of\ d\ \{d\}\ lm]$
{ fix p' **assume** $p' \in lgrid\ (?ch\ right)\ \{d\}\ (lm + level\ (?ch\ right))$
with $grid-disjunct[of\ d\ p]\ up'-inplace[of\ p'\ ?ch\ left\ \{d\}\ d\ lm\ \alpha]\ \beta l-def$
have $\beta l\ p' = \alpha\ p'$ **by** *auto* }
hence $fst\ (?up\ (Suc\ lm)\ p\ \alpha) = (?f\ left + ?result, ?f\ right + ?result)$
using $\beta l-def\ pc-child-inv[of\ left]\ pc-child-inv[of\ right]$
 $Suc.hyps[of\ ?ch\ left\ pc\ left\ p\ \alpha]\ eq[of\ left\ \alpha]$
 $Suc.hyps[of\ ?ch\ right\ p\ pc\ right\ \beta l]\ eq[of\ right\ \beta l]$
by (*cases ?up lm (?ch left) α , cases ?up lm (?ch right) βl*) (*simp add: Let-def*)
ultimately show $?case$ **by** (*auto simp add: p-d-def*)
next
case 0
show $?case$ **by** *simp*
qed

lemma $up'-\beta$:
 $\llbracket d < length\ b ; l + level\ b = lm ; b \in sparsegrid'\ dm ; p \in sparsegrid'\ dm \rrbracket$
 \implies
 $(snd\ (up'\ d\ l\ b\ \alpha))\ p =$
 $(if\ p \in lgrid\ b\ \{d\}\ lm$
 $then\ \sum\ p' \in (lgrid\ p\ \{d\}\ lm) - \{p\}. \alpha\ p' * l2-\varphi\ (p'!\ d)\ (p!\ d)$
 $else\ \alpha\ p)$
 $(is\ \llbracket - ; - ; - ; - \rrbracket \implies (?goal\ l\ b\ p\ \alpha))$
proof (*induct l arbitrary: b p α*)
case ($Suc\ l$)

let $?l = child\ b\ left\ d$ **and** $?r = child\ b\ right\ d$
obtain $p-l$ **where** $p-l-def: ?r = child\ p-l\ left\ d$ **using** $child-ex-neighbour[where\ dir=right]$ **by** *auto*
obtain $p-r$ **where** $p-r-def: ?l = child\ p-r\ right\ d$ **using** $child-ex-neighbour[where\ dir=left]$ **by** *auto*

```

let ?ul = up' d l ?l α
let ?ur = up' d l ?r (snd ?ul)

let ?C p' = α p' * l2-φ (p' ! d) (p ! d)
let ?s s = ∑ p' ∈ (lgrid s {d} lm). ?C p'

from ⟨b ∈ sparsegrid' dm⟩ have length b = dm unfolding sparsegrid'-def start-def
  by auto
hence d < dm using ⟨d < length b⟩ by auto

{ fix p' assume p' ∈ grid ?r {d}
  hence p' ∉ grid ?l {d}
  using grid-disjunct[OF ⟨d < length b⟩] by auto
  hence snd ?ul p' = α p' using up'-inplace by auto
} note eq = this

show ?goal (Suc l) b p α
proof (cases p = b)
  case True

  let ?C p' = α p' * l2-φ (p' ! d) (b ! d)
  let ?s s = ∑ p' ∈ (lgrid s {d} lm). ?C p'

  have d < length ?l using ⟨d < length b⟩ by auto
  from up'-fl-fr[OF this p-r-def]
  have fml: snd (fst ?ul) = (∑ p' ∈ lgrid ?l {d} (l + level ?l). ?C p') by simp

  have d < length ?r using ⟨d < length b⟩ by auto
  from up'-fl-fr[OF this - p-l-def, where α=snd ?ul]
  have fmr: fst (fst ?ur) = (∑ p' ∈ lgrid ?r {d} (l + level ?r).
    ((snd ?ul) p') * l2-φ (p' ! d) (b ! d)) by simp

  have level b < lm using ⟨Suc l + level b = lm⟩ by auto
  hence { b } ⊆ lgrid b {d} lm unfolding lgrid-def by auto
  from sum-diff[OF lgrid-finite this]
  have (∑ p' ∈ (lgrid b {d} lm) - {b}. ?C p') = ?s b - ?C b by simp
  also have ... = ?s ?l + ?s ?r
    using lgrid-sum and ⟨level b < lm⟩ and ⟨d < length b⟩ by auto
  also have ... = snd (fst ?ul) + fst (fst ?ur) using fml and fmr
    and ⟨Suc l + level b = lm⟩ and child-level[OF ⟨d < length b⟩]
    using eq unfolding True lgrid-def by auto

  finally show ?thesis unfolding up'.simps Let-def and fun-upd-def lgrid-def
    using ⟨p = b⟩ and ⟨level b < lm⟩
    by (cases ?ul, cases ?ur, auto)
next
  case False

```

```

have ?r ∈ sparsegrid' dm and ?l ∈ sparsegrid' dm
  using ⟨b ∈ sparsegrid' dm⟩ and ⟨d < dm⟩ unfolding sparsegrid'-def by auto
from Suc.hyps[OF - - this(1)] Suc.hyps[OF - - this(2)]
have ?goal l ?l p α and ?goal l ?r p (snd ?ul)
  using ⟨d < length b⟩ and ⟨Suc l + level b = lm⟩ and ⟨p ∈ sparsegrid' dm⟩ by
auto

show ?thesis
proof (cases p ∈ lgrid b {d} lm)
  case True
  hence level p < lm and p ∈ grid b {d} unfolding lgrid-def by auto
  hence p ∈ grid ?l {d} ∨ p ∈ grid ?r {d}
    unfolding grid-partition[of b] using ⟨p ≠ b⟩ by auto
  thus ?thesis
  proof (rule disjE)
    assume p ∈ grid (child b left d) {d}
    hence p ∉ grid (child b right d) {d}
      using grid-disjunct[OF ⟨d < length b⟩] by auto
    thus ?thesis
      using ⟨?goal l ?l p α⟩ and ⟨?goal l ?r p (snd ?ul)⟩
      using ⟨p ≠ b⟩ ⟨p ∈ lgrid b {d} lm⟩
      unfolding lgrid-def grid-partition[of b]
      by (cases ?ul, cases ?ur, auto simp add: Let-def)
  next
    assume *: p ∈ grid (child b right d) {d}
    hence p ∉ grid (child b left d) {d}
      using grid-disjunct[OF ⟨d < length b⟩] by auto
    moreover
    { fix p' assume p' ∈ grid p {d}
      from grid-transitive[OF this *] eq[of p']
      have snd ?ul p' = α p' by simp
    }
    ultimately show ?thesis
      using ⟨?goal l ?l p α⟩ and ⟨?goal l ?r p (snd ?ul)⟩
      using ⟨p ≠ b⟩ ⟨p ∈ lgrid b {d} lm⟩ *
      unfolding lgrid-def
      by (cases ?ul, cases ?ur, auto simp add: Let-def)
  qed
next
  case False
  then have p ∉ lgrid ?l {d} lm and p ∉ lgrid ?r {d} lm
    unfolding lgrid-def and grid-partition[where p=b] by auto
  with False show ?thesis using ⟨?goal l ?l p α⟩ and ⟨?goal l ?r p (snd ?ul)⟩
    using ⟨p ≠ b⟩ ⟨p ∉ lgrid b {d} lm⟩
    unfolding lgrid-def
    by (cases ?ul, cases ?ur, auto simp add: Let-def)
  qed
qed
next

```

```

case 0
then have lgrid b {d} lm = {}
  using lgrid-empty'[where p=b and lm=lm and ds={d}] by auto
with 0 show ?case unfolding up'.simps by auto
qed

lemma up:
  assumes d < dm and p ∈ sparsegrid dm lm
  shows (up dm lm d α) p = (∑ p' ∈ (lgrid p {d} lm) - {p}. α p' * l2-φ (p' ! d)
  (p ! d))
proof -
  let ?S = λ x p p'. if p' ∈ grid p {d} - {p} then x * l2-φ (p' ! d) (p ! d) else 0
  let ?F = λ d lm p α. snd (up' d lm p α)

  { fix p b assume p ∈ grid b {d}
    from grid-transitive[OF - this subset-refl subset-refl]
    have lgrid b {d} lm ∩ (grid p {d} - {p}) = lgrid p {d} lm - {p}
      unfolding lgrid-def by auto
    } note lgrid-eq = this

  { fix l b p α
    assume b: b ∈ lgrid (start dm) ({0..<dm} - {d}) lm
    hence b ∈ sparsegrid' dm and d < length b using sparsegrid'-start ⟨d < dm⟩
  } by auto
  assume l: l + level b = lm and p: p ∈ sparsegrid dm lm
  note sparsegridE[OF p]

  note up' = up'-β[OF ⟨d < length b⟩ l ⟨b ∈ sparsegrid' dm⟩ ⟨p ∈ sparsegrid' dm⟩]

  have ?F d l b α p =
    (if b = base {d} p then (∑ p' ∈ lgrid b {d} lm. ?S (α p') p p') else α p)
proof (cases b = base {d} p)
  case True with baseE(2)[OF ⟨p ∈ sparsegrid' dm⟩] ⟨level p < lm⟩
  have p ∈ lgrid b {d} lm and p ∈ grid b {d} by auto
  show ?thesis
    using lgrid-eq[OF ⟨p ∈ grid b {d}⟩]
    unfolding up' if-P[OF True] if-P[OF ⟨p ∈ lgrid b {d} lm⟩]
    by (intro sum.mono-neutral-cong-left lgrid-finite) auto
  next
  case False
  moreover have p ∉ lgrid b {d} lm
  proof (rule ccontr)
    assume ¬ ?thesis
    hence base {d} p = b using b by (auto intro!: baseI)
    thus False using False by auto
  qed
  ultimately show ?thesis unfolding up' by auto
qed }
with lift[where F = ?F, OF ⟨d < dm⟩ ⟨p ∈ sparsegrid dm lm⟩]

```

```

have lift-eq: lift ?F dm lm d  $\alpha$  p =
  ( $\sum p' \in \text{lgrid } (\text{base } \{d\} p) \{d\} \text{ lm. } ?S (\alpha p') p p'$ ) by auto
from lgrid-eq[OF baseE(2)[OF sparsegrid-subset[OF  $\langle p \in \text{sparsegrid } dm \text{ lm} \rangle$ ]]]
show ?thesis
  unfolding up-def lift-eq by (intro sum.mono-neutral-cong-right lgrid-finite) auto
qed

end

```

6 Down part

```

theory Down
imports Triangular-Function UpDown-Scheme
begin

```

```

lemma sparsegrid'-parents:
  assumes b:  $b \in \text{sparsegrid}' dm$  and p':  $p' \in \text{parents } d b p$ 
  shows  $p' \in \text{sparsegrid}' dm$ 
  using assms parents-def sparsegrid'I by auto

```

```

lemma down'- $\beta$ :  $\llbracket d < \text{length } b ; l + \text{level } b = lm ; b \in \text{sparsegrid}' dm ; p \in \text{sparsegrid}' dm \rrbracket \implies$ 
  down' d l b fl fr  $\alpha$  p = (if  $p \in \text{lgrid } b \{d\} \text{ lm}$ 
  then
    (fl + (fr - fl) / 2 * (real-of-int (ix p d) / 2^(lv p d - lv b d) - real-of-int (ix
    b d) + 1)) / 2^(lv p d + 1) +
    ( $\sum p' \in \text{parents } d b p. (\alpha p') * \text{l2-}\varphi (p ! d) (p' ! d)$ )
  else  $\alpha p$ )
proof (induct l arbitrary: b  $\alpha$  fl fr p)
  case (Suc l)

```

```

  let ?l = child b left d and ?r = child b right d
  let ?result = ((fl + fr) / 4 + (1 / 3) * ( $\alpha b$ )) / 2^(lv b d)
  let ?fm = (fl + fr) / 2 + ( $\alpha b$ )
  let ?down-l = down' d l (child b left d) fl ?fm ( $\alpha(b := ?result)$ )

```

```

  have length b = dm using  $\langle b \in \text{sparsegrid}' dm \rangle$ 
  unfolding sparsegrid'-def start-def by auto
  hence  $d < dm$  using  $\langle d < \text{length } b \rangle$  by auto

```

```

  have !!dir.  $d < \text{length } (\text{child } b \text{ dir } d)$  using  $\langle d < \text{length } b \rangle$  by auto
  have !!dir.  $l + \text{level } (\text{child } b \text{ dir } d) = lm$ 
    using  $\langle d < \text{length } b \rangle$  and  $\langle \text{Suc } l + \text{level } b = lm \rangle$  and child-level by auto
  have !!dir.  $(\text{child } b \text{ dir } d) \in \text{sparsegrid}' dm$ 
    using  $\langle b \in \text{sparsegrid}' dm \rangle$  and  $\langle d < dm \rangle$  and sparsegrid'-def by auto
  note hyps = Suc.hyps[OF !!dir.  $d < \text{length } (\text{child } b \text{ dir } d)$ ]
    !!dir.  $l + \text{level } (\text{child } b \text{ dir } d) = lm$ 
    !!dir.  $(\text{child } b \text{ dir } d) \in \text{sparsegrid}' dm$ ]

```

```

show ?case
proof (cases p ∈ lgrid b {d} lm)
  case False
  moreover hence p ≠ b and p ∉ lgrid ?l {d} lm
    and p ∉ lgrid ?r {d} lm unfolding lgrid-def
  unfolding grid-partition[where p=b] using ⟨Suc l + level b = lm⟩ by auto
  ultimately show ?thesis
    unfolding down'.simps Let-def fun-upd-def hyps[OF ⟨p ∈ sparsegrid' dm⟩]
    by auto
next
case True hence level p < lm and p ∈ grid b {d} unfolding lgrid-def by auto
let ?lb = lv b d and ?ib = real-of-int (ix b d)
let ?lp = lv p d and ?ip = real-of-int (ix p d)
show ?thesis
proof (cases ∃ dir. p ∈ grid (child b dir d){d})
  case True
  obtain dir where p-grid: p ∈ grid (child b dir d) {d} using True by auto
  hence p ∈ lgrid (child b dir d) {d} lm using ⟨level p < lm⟩ unfolding lgrid-def
by auto
  have lv b d < lv p d using child-lv[OF ⟨d < length b⟩] and grid-single-level[OF
p-grid ⟨d < length (child b dir d)⟩] by auto

  let ?ch = child b dir d
  let ?ich = child b (inv dir) d

  show ?thesis
proof (cases dir)
  case right
  hence p ∈ lgrid ?r {d} lm and p ∈ grid ?r {d}
    using ⟨p ∈ grid ?ch {d}⟩ and ⟨level p < lm⟩ unfolding lgrid-def by auto

  { fix p' fix fl fr x assume p': p' ∈ parents d (child b right d) p
  hence p' ∈ grid (child b right d) {d} unfolding parents-def by simp
  hence p' ∉ lgrid (child b left d) {d} lm and p' ≠ b
    unfolding lgrid-def
    using grid-disjunct[OF ⟨d < length b⟩] grid-not-child by auto

  from hyps[OF sparsegrid'-parents[OF ⟨child b right d ∈ sparsegrid' dm⟩
p']] this
  have down' d l (child b left d) fl fr (α(b := x)) p' = α p' by auto }
thus ?thesis
  unfolding down'.simps Let-def hyps[OF ⟨p ∈ sparsegrid' dm⟩]
    parent-sum[OF ⟨p ∈ grid ?r {d}⟩ ⟨d < length b⟩]
    l2-child[OF ⟨d < length b⟩ ⟨p ∈ grid ?r {d}⟩]
  using child-ix child-lv ⟨d < length b⟩ level-shift[OF ⟨lv b d < lv p d⟩]
    sgn.simps ⟨p ∈ lgrid b {d} lm⟩ ⟨p ∈ lgrid ?r {d} lm⟩
  by (auto simp add: algebra-simps diff-divide-distrib add-divide-distrib)
next
case left

```

hence $p \in \text{lgrid } ?l \{d\} \text{ lm}$ and $p \in \text{grid } ?l \{d\}$
 using $\langle p \in \text{grid } ?ch \{d\} \rangle$ and $\langle \text{level } p < \text{lm} \rangle$ **unfolding** *lgrid-def* **by** *auto*
 hence $\neg p \in \text{lgrid } ?r \{d\} \text{ lm}$
 using *grid-disjunct*[*OF* $\langle d < \text{length } b \rangle$] **unfolding** *lgrid-def* **by** *auto*
 { **fix** p' **assume** p' : $p' \in \text{parents } d \text{ (child } b \text{ left } d) \text{ } p$
 hence $p' \in \text{grid (child } b \text{ left } d) \{d\}$ **unfolding** *parents-def* **by** *simp*
 hence $p' \neq b$ **using** *grid-not-child*[*OF* $\langle d < \text{length } b \rangle$] **by** *auto* }
thus *?thesis*
unfolding *down'.simps* *Let-def* *hyps*[*OF* $\langle p \in \text{sparsegrid}' \text{ } dm \rangle$]
parent-sum[*OF* $\langle p \in \text{grid } ?l \{d\} \rangle \langle d < \text{length } b \rangle$]
l2-child[*OF* $\langle d < \text{length } b \rangle \langle p \in \text{grid } ?l \{d\} \rangle$] *sgn.simps*
if-P[*OF* $\langle p \in \text{lgrid } b \{d\} \text{ lm} \rangle$] *if-P*[*OF* $\langle p \in \text{lgrid } ?l \{d\} \text{ lm} \rangle$]
if-not-P[*OF* $\langle p \notin \text{lgrid } ?r \{d\} \text{ lm} \rangle$]
using *child-ix* *child-lv* $\langle d < \text{length } b \rangle$ *level-shift*[*OF* $\langle lv \text{ } b \text{ } d < lv \text{ } p \text{ } d \rangle$]
by (*auto simp add: algebra-simps diff-divide-distrib add-divide-distrib*)
qed
next
case *False* **hence** *not-child*: !! *dir*. $\neg p \in \text{grid (child } b \text{ dir } d) \{d\}$ **by** *auto*
hence $p = b$ **using** *grid-onedim-split*[**where** $ds=\{\}$] **and** $d=d$ **and** $b=b$] $\langle p \in \text{grid } b \{d\} \rangle$ **unfolding** *grid-empty-ds*[**where** $b=b$] **by** *auto*
from *not-child* **have** *lnot-child*: !! *dir*. $\neg p \in \text{lgrid (child } b \text{ dir } d) \{d\} \text{ lm}$
unfolding *lgrid-def* **by** *auto*
have *result*: $((fl + fr) / 4 + 1 / 3 * \alpha b) / 2 \wedge lv \text{ } b \text{ } d = (fl + (fr - fl) / 2) / 2 \wedge (lv \text{ } b \text{ } d + 1) + \alpha b * l2-\varphi (b ! d) (b ! d)$
by (*auto simp: l2-same diff-divide-distrib add-divide-distrib times-divide-eq-left*[*symmetric*]
algebra-simps)
show *?thesis*
unfolding *down'.simps* *Let-def* *fun-upd-def* *hyps*[*OF* $\langle p \in \text{sparsegrid}' \text{ } dm \rangle$]
if-P[*OF* $\langle p \in \text{lgrid } b \{d\} \text{ lm} \rangle$] *if-not-P*[*OF* *lnot-child*] *if-P*[*OF* $\langle p = b \rangle$]
unfolding $\langle p = b \rangle$ *parents-single* **unfolding** *result* **by** *auto*
qed
qed
next
case *0*
have $p \notin \text{lgrid } b \{d\} \text{ lm}$
proof (*rule ccontr*)
assume $\neg p \notin \text{lgrid } b \{d\} \text{ lm}$
hence $p \in \text{grid } b \{d\}$ **and** $\text{level } p < \text{lm}$ **unfolding** *lgrid-def* **by** *auto*
moreover **from** *grid-level*[*OF* $\langle p \in \text{grid } b \{d\} \rangle$] **and** $\langle 0 + \text{level } b = \text{lm} \rangle$ **have**
 $\text{lm} \leq \text{level } p$ **by** *auto*
ultimately **show** *False* **by** *auto*
qed
thus *?case* **unfolding** *down'.simps* **by** *auto*
qed

lemma *down*: **assumes** $d < dm$ **and** p : $p \in \text{sparsegrid } dm \text{ } lm$
shows $(\text{down } dm \text{ } lm \text{ } d \text{ } \alpha) p = (\sum p' \in \text{parents } d \text{ (base } \{d\} \text{ } p) p. (\alpha p') * l2-\varphi (p ! d) (p' ! d))$
proof –


```

let ?F d l p = down' d l p 0 0
let ?S x p p' = if p' ∈ parents d (base {d} p) p then x * l2-φ (p ! d) (p' ! d) else
0

{ fix p α assume p ∈ sparsegrid dm lm
  from le-less-trans[OF grid-level sparsegridE(2)][OF this]
  have parents d (base {d} p) p ⊆ lgrid (base {d} p) {d} lm
    unfolding lgrid-def parents-def by auto
  hence (∑ p' ∈ lgrid (base {d} p) {d} lm. ?S (α p') p p') =
    (∑ p' ∈ parents d (base {d} p) p. α p' * l2-φ (p ! d) (p' ! d))
    using lgrid-finite by (intro sum.mono-neutral-cong-right) auto
} note sum-eq = this

{ fix l p b α
  assume b: b ∈ lgrid (start dm) ({0..<dm} - {d}) lm and l + level b = lm
    and p ∈ sparsegrid dm lm
  hence b-spg: b ∈ sparsegrid' dm and p-spg: p ∈ sparsegrid' dm and
    d < length b and level p < lm
    using sparsegrid'-start sparsegrid'-subset ⟨d < dm⟩ by auto
  have ?F d l b α p = (if b = base {d} p then ∑ p' ∈ lgrid b {d} lm. ?S (α p') p
p' else α p)
  proof (cases b = base {d} p)
  case True
  have p ∈ lgrid (base {d} p) {d} lm
    using baseE(2)[OF p-spg] and ⟨level p < lm⟩
    unfolding lgrid-def by auto
  thus ?thesis unfolding if-P[OF True]
    unfolding True sum-eq[OF ⟨p ∈ sparsegrid dm lm⟩]
    unfolding down'-β[OF ⟨d < length b⟩ ⟨l + level b = lm⟩ b-spg p-spg,
unfolded True] by auto
  next
  case False
  have p ∉ lgrid b {d} lm
  proof (rule ccontr)
  assume ¬ ?thesis hence p ∈ grid b {d} by auto
  from b this have b = base {d} p using baseI by auto
  thus False using False by simp
  qed
  thus ?thesis
    unfolding if-not-P[OF False]
    unfolding down'-β[OF ⟨d < length b⟩ ⟨l + level b = lm⟩ b-spg p-spg]
    by auto
  qed }
from lift[OF ⟨d < dm⟩ ⟨p ∈ sparsegrid dm lm⟩, where F = ?F and S = ?S, OF
this]
show ?thesis
  unfolding down-def
  unfolding sum-eq[OF p] by simp
qed

```

end

7 UpDown

theory *Up-Down*
imports *Up Down*
begin

lemma *updown'*: $\llbracket d \leq dm; p \in \text{sparsegrid } dm \text{ } lm \rrbracket$
 $\implies (\text{updown}' \text{ } dm \text{ } lm \text{ } d \text{ } \alpha) \text{ } p = (\sum p' \in \text{lgrid } (\text{base } \{0 \dots d\} \text{ } p) \{0 \dots d\} \text{ } lm. \alpha$
 $p' * (\prod d' \in \{0 \dots d\}. \text{l2-}\varphi (p' ! d') (p ! d'))$
 $(\text{is } \llbracket - ; - \rrbracket \implies - = (\sum p' \in ?\text{subgrid } d \text{ } p. \alpha \text{ } p' * ?\text{prod } d \text{ } p' \text{ } p))$
proof (*induct d arbitrary: α p*)
case 0 **hence** $?\text{subgrid } 0 \text{ } p = \{p\}$ **using** *base-empty unfolding lgrid-def and*
sparsegrid-def sparsegrid'-def **by** *auto*
thus $?\text{case}$ **unfolding** *updown'.simps* **by** *auto*
next
case (*Suc d*)
let $?\text{leafs } p = (\text{lgrid } p \text{ } \{d\} \text{ } lm) - \{p\}$
let $?\text{parents} = \text{parents } d \text{ } (\text{base } \{d\} \text{ } p) \text{ } p$
let $?b = \text{base } \{0 \dots d\} \text{ } p$
have $d < dm$ **using** $\langle \text{Suc } d \leq dm \rangle$ **by** *auto*

have $p\text{-spg}: p \in \text{grid } (\text{start } dm) \{0 \dots dm\}$ **and** $p\text{-spg}' : p \in \text{sparsegrid}' \text{ } dm$ **and**
level p < lm **using** $\langle p \in \text{sparsegrid } dm \text{ } lm \rangle$
unfolding *sparsegrid-def and sparsegrid'-def and lgrid-def* **by** *auto*
have $p'\text{-in-spg}: !! p'. p' \in ?\text{subgrid } d \text{ } p \implies p' \in \text{sparsegrid } dm \text{ } lm$
using *base-grid[OF p-spg'] unfolding sparsegrid'-def sparsegrid-def lgrid-def*
by *auto*

from *baseE[OF p-spg', where ds={0..<d}]*
have $?b \in \text{grid } (\text{start } dm) \{d \dots dm\}$ **and** $p\text{-bgrid}: p \in \text{grid } ?b \{0 \dots d\}$ **by** *auto*
hence $d < \text{length } ?b$ **using** $\langle \text{Suc } d \leq dm \rangle$ **by** *auto*
have $p ! d = ?b ! d$ **using** *base-out[OF - - p-spg'] $\langle \text{Suc } d \leq dm \rangle$* **by** *auto*

have $\text{length } p = dm$ **using** $\langle p \in \text{sparsegrid } dm \text{ } lm \rangle$ **unfolding** *sparsegrid-def*
lgrid-def **by** *auto*
hence $d < \text{length } p$ **using** $\langle d < dm \rangle$ **by** *auto*

have $\text{updown}' \text{ } dm \text{ } lm \text{ } d \text{ } (\text{up } dm \text{ } lm \text{ } d \text{ } \alpha) \text{ } p =$
 $(\sum p' \in ?\text{subgrid } d \text{ } p. (\text{up } dm \text{ } lm \text{ } d \text{ } \alpha) \text{ } p' * (?\text{prod } d \text{ } p' \text{ } p))$
using *Suc* **by** *auto*
also have $\dots = (\sum p' \in ?\text{subgrid } d \text{ } p. (\sum p'' \in ?\text{leafs } p'. \alpha \text{ } p'' * ?\text{prod } (\text{Suc } d)$
 $p'' \text{ } p))$
proof (*intro sum.cong refl*)
fix p' **assume** $p' \in ?\text{subgrid } d \text{ } p$
hence $d < \text{length } p'$ **unfolding** *lgrid-def* **using** *base-length[OF p-spg'] $\langle \text{Suc } d$*
 $\leq dm \rangle$ **by** *auto*

have $up\ dm\ lm\ d\ \alpha\ p' * ?prod\ d\ p' p =$
 $(\sum p'' \in ?leafs\ p'. \alpha\ p'' * l2-\varphi\ (p''! d)\ (p'! d)) * ?prod\ d\ p' p$
using $\langle p' \in ?subgrid\ d\ p \rangle\ up\ \langle Suc\ d \leq dm \rangle\ p'\text{-in-spg}$ **by** *auto*
also have $\dots = (\sum p'' \in ?leafs\ p'. \alpha\ p'' * l2-\varphi\ (p''! d)\ (p'! d)) * ?prod\ d\ p' p$
using *sum-distrib-right* **by** *auto*
also have $\dots = (\sum p'' \in ?leafs\ p'. \alpha\ p'' * ?prod\ (Suc\ d)\ p'' p)$
proof (*intro sum.cong refl*)
fix p'' **assume** $p'' \in ?leafs\ p'$
have $?prod\ d\ p' p = ?prod\ d\ p'' p$
proof (*intro prod.cong refl*)
fix d' **assume** $d' \in \{0..<d\}$
hence $d\text{-lt-}p$: $d' < length\ p'$ **and** $d'\text{-not-}d$: $d' \notin \{d\}$ **using** $\langle d < length\ p' \rangle$
by *auto*
hence $p'! d' = p''! d'$ **using** $\langle p'' \in ?leafs\ p' \rangle\ grid\text{-invariant}[OF\ d\text{-lt-}p\ d'\text{-not-}d]$ **unfolding** *lgrid-def* **by** *auto*
thus $l2-\varphi\ (p'! d')\ (p! d') = l2-\varphi\ (p''! d')\ (p! d')$ **by** *auto*
qed
moreover have $p'! d = p! d$
using $\langle p' \in ?subgrid\ d\ p \rangle$ **and** $grid\text{-invariant}[OF\ \langle d < length\ ?b \rangle]$, **where**
 $p=p'$ **and** $ds=\{0..<d\}$ **unfolding** *lgrid-def* $\langle p! d = ?b! d \rangle$ **by** *auto*
ultimately have $l2-\varphi\ (p''! d)\ (p'! d) * ?prod\ d\ p' p =$
 $l2-\varphi\ (p''! d)\ (p! d) * ?prod\ d\ p'' p$ **by** *auto*
also have $\dots = ?prod\ (Suc\ d)\ p'' p$
proof –
have $insert\ d\ \{0..<d\} = \{0..<Suc\ d\}$ **by** *auto*
moreover from *prod.insert*
have $prod\ (\lambda\ d'. l2-\varphi\ (p''! d')\ (p! d'))\ (insert\ d\ \{0..<d\}) =$
 $(\lambda\ d'. l2-\varphi\ (p''! d')\ (p! d'))\ d * prod\ (\lambda\ d'. l2-\varphi\ (p''! d')\ (p! d'))\ \{0..<d\}$
by *auto*
ultimately show *?thesis* **by** *auto*
qed
finally show $\alpha\ p'' * l2-\varphi\ (p''! d)\ (p'! d) * ?prod\ d\ p' p = \alpha\ p'' * ?prod\ (Suc\ d)\ p'' p$ **by** *auto*
qed
finally show $(up\ dm\ lm\ d\ \alpha)\ p' * (?prod\ d\ p' p) = (\sum p'' \in ?leafs\ p'. \alpha\ p'' * ?prod\ (Suc\ d)\ p'' p)$ **by** *auto*
qed
also have $\dots = (\sum (p', p'') \in Sigma\ (?subgrid\ d\ p)\ (\lambda p'. (?leafs\ p')). (\alpha\ p')) * (?prod\ (Suc\ d)\ p'' p)$
by (*rule sum.Sigma, auto simp add: lgrid-finite*)
also have $\dots = (\sum p''' \in (\bigcup p' \in ?subgrid\ d\ p. (\bigcup p'' \in ?leafs\ p'. \{ (p', p'') \})))$
 $((\lambda p''. \alpha\ p'' * ?prod\ (Suc\ d)\ p'' p) o snd)\ p''$) **unfolding** *Sigma-def* **by**
(*rule sum.cong[OF refl], auto*)
also have $\dots = (\sum p'' \in snd\ \langle \bigcup p' \in ?subgrid\ d\ p. (\bigcup p'' \in ?leafs\ p'. \{ (p', p'') \}) \rangle)$
 $\alpha\ p'' * (?prod\ (Suc\ d)\ p'' p)$ **unfolding** *lgrid-def*
by (*rule sum.reindex[symmetric]*),

rule subset-inj-on[*OF grid-grid-inj-on*[*OF ivl-disj-int*(15)[**where** $l=0$ **and** $m=d$ **and** $u=d$], **where** $b=?b$]])

auto

also have $\dots = (\sum p'' \in (\bigcup p' \in ?subgrid\ d\ p. (\bigcup p'' \in ?leafs\ p'.\ snd\ \{ (p', p'') \})))$.

$\alpha\ p'' * (?prod\ (Suc\ d)\ p''\ p)$ **by** (*auto simp only: image-UN*)

also have $\dots = (\sum p'' \in (\bigcup p' \in ?subgrid\ d\ p. ?leafs\ p').\ \alpha\ p'' * (?prod\ (Suc\ d)\ p''\ p))$ **by** *auto*

finally have *up-part: updown' dm lm d (up dm lm d α) p* $= (\sum p'' \in (\bigcup p' \in ?subgrid\ d\ p. ?leafs\ p').\ \alpha\ p'' * (?prod\ (Suc\ d)\ p''\ p))$.

have *down dm lm d (updown' dm lm d α) p* $= (\sum p' \in ?parents. (updown' dm lm d\ \alpha\ p') * l2-\varphi\ (p\ !\ d)\ (p'\ !\ d))$

using $\langle Suc\ d \leq dm \rangle$ **and** *down and* $\langle p \in sparsegrid\ dm\ lm \rangle$ **by** *auto*

also have $\dots = (\sum p' \in ?parents. \sum p'' \in ?subgrid\ d\ p'.\ \alpha\ p'' * ?prod\ (Suc\ d)\ p''\ p)$

proof (*rule sum.cong*[*OF refl*])

fix p' **let** $?b' = base\ \{d\}\ p$

assume $p' \in ?parents$

hence *p-lgrid: p' ∈ lgrid ?b' {d} (level p + 1)* **using** *parents-subset-lgrid* **by** *auto*

hence $p' \in sparsegrid\ dm\ lm$ **and** *p'-spg': p' ∈ sparsegrid' dm* **using** $\langle level\ p < lm \rangle$ *base-grid*[*OF p-spg'*] **unfolding** *sparsegrid-def lgrid-def sparsegrid'-def* **by** *auto*

hence $length\ p' = dm$ **unfolding** *sparsegrid-def lgrid-def* **by** *auto*

hence $d < length\ p'$ **using** $\langle Suc\ d \leq dm \rangle$ **by** *auto*

from *p-lgrid* **have** *p'-grid: p' ∈ grid ?b' {d}* **unfolding** *lgrid-def* **by** *auto*

have $(updown' dm lm d\ \alpha\ p') * l2-\varphi\ (p\ !\ d)\ (p'\ !\ d) = (\sum p'' \in ?subgrid\ d\ p'.\ \alpha\ p'' * ?prod\ d\ p''\ p') * l2-\varphi\ (p\ !\ d)\ (p'\ !\ d)$

using $\langle p' \in sparsegrid\ dm\ lm \rangle$ *Suc* **by** *auto*

also have $\dots = (\sum p'' \in ?subgrid\ d\ p'.\ \alpha\ p'' * ?prod\ d\ p''\ p' * l2-\varphi\ (p\ !\ d)\ (p'\ !\ d))$

using *sum-distrib-right* **by** *auto*

also have $\dots = (\sum p'' \in ?subgrid\ d\ p'.\ \alpha\ p'' * ?prod\ (Suc\ d)\ p''\ p)$

proof (*rule sum.cong*[*OF refl*])

fix p'' **assume** $p'' \in ?subgrid\ d\ p'$

have $?prod\ d\ p''\ p' = ?prod\ d\ p''\ p$

proof (*rule prod.cong, rule refl*)

fix d' **assume** $d' \in \{0..<d\}$

hence $d' < dm$ **and** $d' \notin \{d\}$ **using** $\langle Suc\ d \leq dm \rangle$ **by** *auto*

from *grid-base-out*[*OF this p-spg' p'-grid*]

show $l2-\varphi\ (p''!\ d')\ (p'\ !\ d') = l2-\varphi\ (p''!\ d')\ (p'\ !\ d')$ **by** *auto*

qed

moreover

have $l2-\varphi\ (p\ !\ d)\ (p'\ !\ d) = l2-\varphi\ (p''\ !\ d)\ (p\ !\ d)$

proof –

have $d < dm$ **and** $d \notin \{0..<d\}$ **using** $\langle \text{Suc } d \leq dm \rangle$ *base-length p'-spg'* **by**
auto
from *grid-base-out[OF this p'-spg']* $\langle p'' \in ?\text{subgrid } d \ p' \rangle$ [*unfolded lgrid-def*]
show *?thesis using l2-commutative by auto*
qed
moreover have $?\text{prod } d \ p'' \ p * \text{l2-}\varphi \ (p'' \ ! \ d) \ (p \ ! \ d) = ?\text{prod } (\text{Suc } d) \ p'' \ p$
proof –
have $\text{insert } d \ \{0..<d\} = \{0..<\text{Suc } d\}$ **by** *auto*
moreover from *prod.insert*
have $(\lambda \ d'. \ \text{l2-}\varphi \ (p'' \ ! \ d') \ (p \ ! \ d')) \ d * \text{prod } (\lambda \ d'. \ \text{l2-}\varphi \ (p'' \ ! \ d') \ (p \ ! \ d'))$
 $\{0..<d\} =$
 $\text{prod } (\lambda \ d'. \ \text{l2-}\varphi \ (p'' \ ! \ d') \ (p \ ! \ d')) \ (\text{insert } d \ \{0..<d\})$
by *auto*
hence $(\text{prod } (\lambda \ d'. \ \text{l2-}\varphi \ (p'' \ ! \ d') \ (p \ ! \ d')) \ \{0..<d\}) * (\lambda \ d'. \ \text{l2-}\varphi \ (p'' \ ! \ d')$
 $(p \ ! \ d')) \ d =$
 $\text{prod } (\lambda \ d'. \ \text{l2-}\varphi \ (p'' \ ! \ d') \ (p \ ! \ d')) \ (\text{insert } d \ \{0..<d\})$
by *auto*
ultimately show *?thesis by auto*
qed
ultimately show $\alpha \ p'' * ?\text{prod } d \ p'' \ p' * \text{l2-}\varphi \ (p \ ! \ d) \ (p' \ ! \ d) = \alpha \ p'' * ?\text{prod}$
 $(\text{Suc } d) \ p'' \ p$ **by** *auto*
qed
finally show $(\text{updown}' \ dm \ lm \ d \ \alpha \ p') * \text{l2-}\varphi \ (p \ ! \ d) \ (p' \ ! \ d) = (\sum \ p'' \in ?\text{subgrid}$
 $d \ p'. \ \alpha \ p'' * ?\text{prod } (\text{Suc } d) \ p'' \ p)$ **by** *auto*
qed
also have $\dots = (\sum \ (p', \ p'') \in (\text{Sigma } ?\text{parents } (? \ \text{subgrid } d)). \ \alpha \ p'' * ?\text{prod } (\text{Suc}$
 $d) \ p'' \ p)$
by *(rule sum.Sigma, auto simp add: parents-finite lgrid-finite)*
also have $\dots = (\sum \ p''' \in (\bigcup \ p' \in ?\text{parents}. \ (\bigcup \ p'' \in ?\text{subgrid } d \ p'. \ \{ (p', \ p'')$
 $\})))$
 $((\lambda \ p''. \ \alpha \ p'' * ?\text{prod } (\text{Suc } d) \ p'' \ p) \ o \ \text{snd}) \ p'')$ **unfolding** *Sigma-def by*
(rule sum.cong[OF refl], auto)
also have $\dots = (\sum \ p'' \in \text{snd } ' (\bigcup \ p' \in ?\text{parents}. \ (\bigcup \ p'' \in ?\text{subgrid } d \ p'. \ \{ (p',$
 $p'' \})))$. $\alpha \ p'' * (? \ \text{prod } (\text{Suc } d) \ p'' \ p)$
proof *(rule sum.reindex[symmetric], rule inj-onI)*
fix $x \ y$
assume $x \in (\bigcup \ p' \in \text{parents } d \ (\text{base } \{d\} \ p) \ p. \ \bigcup \ p'' \in \text{lgrid } (\text{base } \{0..<d\} \ p')$
 $\{0..<d\} \ \text{lm}. \ \{(p', \ p'')\})$
hence $x\text{-snd}: \text{snd } x \in \text{grid } (\text{base } \{0..<d\} \ (\text{fst } x)) \ \{0..<d\}$ **and** $\text{fst } x \in \text{grid } (\text{base}$
 $\{d\} \ p) \ \{d\}$ **and** $p \in \text{grid } (\text{fst } x) \ \{d\}$
unfolding *parents-def lgrid-def by auto*
hence $x\text{-spg}: \text{fst } x \in \text{sparsegrid}' \ dm$ **using** *base-grid[OF p-spg'] by auto*

assume $y \in (\bigcup \ p' \in \text{parents } d \ (\text{base } \{d\} \ p) \ p. \ \bigcup \ p'' \in \text{lgrid } (\text{base } \{0..<d\} \ p')$
 $\{0..<d\} \ \text{lm}. \ \{(p', \ p'')\})$
hence $y\text{-snd}: \text{snd } y \in \text{grid } (\text{base } \{0..<d\} \ (\text{fst } y)) \ \{0..<d\}$ **and** $\text{fst } y \in \text{grid}$
 $(\text{base } \{d\} \ p) \ \{d\}$ **and** $p \in \text{grid } (\text{fst } y) \ \{d\}$
unfolding *parents-def lgrid-def by auto*
hence $y\text{-spg}: \text{fst } y \in \text{sparsegrid}' \ dm$ **using** *base-grid[OF p-spg'] by auto*

hence $\text{length } (fst y) = dm$ **unfolding** *sparsegrid'-def* **by** *auto*

assume $snd x = snd y$
have $fst x = fst y$
proof (*rule nth-equalityI*)
show $l\text{-eq: length } (fst x) = \text{length } (fst y)$ **using** *grid-length[OF <p ∈ grid (fst y) {d}>]* *grid-length[OF <p ∈ grid (fst x) {d}>]*
by *auto*
show $fst x ! i = fst y ! i$ **if** $i < \text{length } (fst x)$ **for** i
proof –
have $i < \text{length } (fst y)$ **and** $i < dm$ **using** *that l-eq and <length (fst y) = dm>* **by** *auto*
show $fst x ! i = fst y ! i$
proof (*cases i = d*)
case *False* **hence** $i \notin \{d\}$ **by** *auto*
with *grid-invariant[OF <i < length (fst x)> this <p ∈ grid (fst x) {d}>]*
grid-invariant[OF <i < length (fst y)> this <p ∈ grid (fst y) {d}>]
show *?thesis* **by** *auto*
next
case *True* **with** *grid-base-out[OF <i < dm> - y-spg y-snd]* *grid-base-out[OF <i < dm> - x-spg x-snd]*
show *?thesis* **using** $\langle snd x = snd y \rangle$ **by** *auto*
qed
qed
qed
show $x = y$ **using** *prod-eqI[OF <fst x = fst y> <snd x = snd y>]* .
qed
also **have** $\dots = (\sum p'' \in (\bigcup p' \in ?parents. (\bigcup p'' \in ?subgrid d p'. snd ' \{ (p', p'') \})))$
 $\alpha p'' * (?prod (Suc d) p'' p)$ **by** (*auto simp only: image-UN*)
also **have** $\dots = (\sum p'' \in (\bigcup p' \in ?parents. ?subgrid d p'). \alpha p'' * ?prod (Suc d) p'' p)$ **by** *auto*
finally **have** *down-part: down dm lm d (updown' dm lm d α) p =*
 $(\sum p'' \in (\bigcup p' \in ?parents. ?subgrid d p'). \alpha p'' * ?prod (Suc d) p'' p)$.
have *updown' dm lm (Suc d) α p =*
 $(\sum p'' \in (\bigcup p' \in ?subgrid d p. ?leafs p'). \alpha p'' * ?prod (Suc d) p'' p) +$
 $(\sum p'' \in (\bigcup p' \in ?parents. ?subgrid d p'). \alpha p'' * ?prod (Suc d) p'' p)$
unfolding *sum-vector-def updown'.simps down-part and up-part ..*
also **have** $\dots = (\sum p'' \in (\bigcup p' \in ?subgrid d p. ?leafs p') \cup (\bigcup p' \in ?parents. ?subgrid d p'). \alpha p'' * ?prod (Suc d) p'' p)$
proof (*rule sum.union-disjoint[symmetric]*, *simp add: lgrid-finite, simp add: lgrid-finite parents-finite,*
rule iffD2[OF disjoint-iff-not-equal], *rule ballI, rule ballI*)
fix $x y$
assume $x \in (\bigcup p' \in ?subgrid d p. ?leafs p')$
then **obtain** px **where** $px \in \text{grid } (base \{0..<d\} p) \{0..<d\}$ **and** $x \in \text{grid } px \{d\}$ **and** $x \neq px$ **unfolding** *lgrid-def* **by** *auto*
with *grid-base-out[OF - - p-spg' this(1)] <Suc d ≤ dm>* *base-length[OF p-spg']*

grid-level-d
have $lv\ px\ d < lv\ x\ d$ **and** $px\ !\ d = p\ !\ d$ **by** *auto*
hence $lv\ p\ d < lv\ x\ d$ **unfolding** *lv-def* **by** *auto*
moreover
assume $y \in (\bigcup p' \in ?parents. ?subgrid\ d\ p')$
then obtain py **where** *y-grid*: $y \in grid\ (base\ \{0..<d\}\ py)\ \{0..<d\}$ **and** $py \in ?parents$ **unfolding** *lgrid-def* **by** *auto*
hence $py \in grid\ (base\ \{d\}\ p)\ \{d\}$ **and** $p \in grid\ py\ \{d\}$ **unfolding** *parents-def* **by** *auto*
hence *py-spg*: $py \in sparsegrid'\ dm$ **using** *base-grid*[*OF p-spg'*] **by** *auto*
have $y\ !\ d = py\ !\ d$ **using** *grid-base-out*[*OF - - py-spg y-grid*] $\langle Suc\ d \leq dm \rangle$ **by** *auto*
hence $lv\ y\ d \leq lv\ p\ d$ **using** *grid-single-level*[*OF* $\langle p \in grid\ py\ \{d\} \rangle$] $\langle Suc\ d \leq dm \rangle$ **and** *sparsegrid'-length*[*OF py-spg*] **unfolding** *lv-def* **by** *auto*
ultimately
show $x \neq y$ **by** *auto*
qed
also have $\dots = (\sum p' \in ?subgrid\ (Suc\ d)\ p. \alpha\ p' * ?prod\ (Suc\ d)\ p'\ p)$ **(is** $(\sum x \in ?in. ?F\ x) = (\sum x \in ?out. ?F\ x)$ **)**
proof (*rule sum.mono-neutral-left, simp add: lgrid-finite*)
show $?in \subseteq ?out$ **(is** $?children \cup ?siblings \subseteq -$ **)**
proof (*rule subsetI*)
fix x **assume** $x \in ?in$
show $x \in ?out$
proof (*cases x ∈ ?children*)
case *False* **hence** $x \in ?siblings$ **using** $\langle x \in ?in \rangle$ **by** *auto*
then obtain px **where** $px \in parents\ d\ (base\ \{d\}\ p)\ p$ **and** $x \in lgrid\ (base\ \{0..<d\}\ px)\ \{0..<d\}\ lm$ **by** *auto*
hence $level\ x < lm$ **and** $px \in grid\ (base\ \{d\}\ p)\ \{d\}$ **and** $x \in grid\ (base\ \{0..<d\}\ px)\ \{0..<d\}$ **and** $\{d\} \cup \{0..<d\} = \{0..<Suc\ d\}$ **unfolding** *lgrid-def parents-def* **by** *auto*
with *grid-base-union*[*OF p-spg' this(2) this(3)*] **show** *?thesis* **unfolding** *lgrid-def* **by** *auto*
next
have *d-eq*: $\{0..<Suc\ d\} \cup \{d\} = \{0..<Suc\ d\}$ **by** *auto*
case *True*
then obtain px **where** $px \in ?subgrid\ d\ p$ **and** $x \in lgrid\ px\ \{d\}\ lm$ **and** $x \neq px$ **by** *auto*
hence $px \in grid\ (base\ \{0..<d\}\ p)\ \{0..<d\}$ **and** $x \in grid\ px\ \{d\}$ **and** $level\ x < lm$ **and** $\{d\} \cup \{0..<d\} = \{0..<Suc\ d\}$ **unfolding** *lgrid-def* **by** *auto*
from *grid-base-dim-add*[*OF - p-spg' this(1)*]
have $px \in grid\ (base\ \{0..<Suc\ d\}\ p)\ \{0..<Suc\ d\}$ **by** *auto*
from *grid-transitive*[*OF* $\langle x \in grid\ px\ \{d\} \rangle$] *this*
show *?thesis* **unfolding** *lgrid-def* **using** $\langle level\ x < lm \rangle$ *d-eq* **by** *auto*
qed
qed
show $\forall x \in ?out - ?in. ?F\ x = 0$
proof

fix x **assume** $x \in ?out - ?in$

hence $x \in ?out$ **and** $up-ps'$: $!! p'. p' \in ?subgrid\ d\ p \implies x \notin lgrid\ p'\ \{d\}\ lm - \{p'\}$

and $down-ps'$: $!! p'. p' \in ?parents \implies x \notin ?subgrid\ d\ p'$ **by** *auto*

hence $x-eq$: $x \in grid\ (base\ \{0..<Suc\ d\}\ p)\ \{0..<Suc\ d\}$ **and** $level\ x < lm$

unfolding $lgrid-def$ **by** *auto*

hence $up-ps$: $!! p'. p' \in ?subgrid\ d\ p \implies x \notin grid\ p'\ \{d\} - \{p'\}$ **and**

$down-ps$: $!! p'. p' \in ?parents \implies x \notin grid\ (base\ \{0..<d\}\ p')\ \{0..<d\}$

using $up-ps'$ $down-ps'$ **unfolding** $lgrid-def$ **by** *auto*

have $ds-eq$: $\{0..<Suc\ d\} = \{0..<d\} \cup \{d\}$ **by** *auto*

have $x \notin grid\ (base\ \{0..<d\}\ p)\ \{0..<Suc\ d\} - grid\ (base\ \{0..<d\}\ p)\ \{0..<d\}$

proof

assume $x \in grid\ (base\ \{0..<d\}\ p)\ \{0..<Suc\ d\} - grid\ (base\ \{0..<d\}\ p)\ \{0..<d\}$

hence $x \in grid\ (base\ \{0..<d\}\ p)\ (\{d\} \cup \{0..<d\})$ **and** $x-ngrid$: $x \notin grid\ (base\ \{0..<d\}\ p)\ \{0..<d\}$ **using** $ds-eq$ **by** *auto*

from $grid-split[OF\ this(1)]$ **obtain** px **where** $px-grid$: $px \in grid\ (base\ \{0..<d\}\ p)\ \{0..<d\}$ **and** $x \in grid\ px\ \{d\}$ **by** *auto*

from $grid-level[OF\ this(2)]$ $\langle level\ x < lm \rangle$ **have** $level\ px < lm$ **by** *auto*

hence $px \in ?subgrid\ d\ p$ **using** $px-grid$ **unfolding** $lgrid-def$ **by** *auto*

hence $x \notin grid\ px\ \{d\} - \{px\}$ **using** $up-ps$ **by** *auto*

moreover **have** $x \neq px$ **proof** (*rule ccontr*) **assume** $\neg x \neq px$ **with** $px-grid$

and $x-ngrid$ **show** *False* **by** *auto* **qed**

ultimately **show** *False* **using** $\langle x \in grid\ px\ \{d\} \rangle$ **by** *auto*

qed

moreover **have** $p \in ?parents$ **unfolding** $parents-def$ **using** $baseE(2)[OF\ p-spg]$ **by** *auto*

hence $x \notin grid\ (base\ \{0..<d\}\ p)\ \{0..<d\}$ **by** (*rule down-ps*)

ultimately **have** $x-ngrid$: $x \notin grid\ (base\ \{0..<d\}\ p)\ \{0..<Suc\ d\}$ **by** *auto*

have $x-spg$: $x \in sparsegrid'\ dm$ **using** $base-grid[OF\ p-spg]$ $x-eq$ **by** *auto*

hence $length\ x = dm$ **using** $grid-length$ **by** *auto*

let $?bx = base\ \{0..<d\}\ x$ **and** $?bp = base\ \{0..<d\}\ p$ **and** $?bx1 = base\ \{d\}\ x$

and $?bp1 = base\ \{d\}\ p$ **and** $?px = p[d := x!\ d]$

have $x-nochild-p$: $?bx \notin grid\ ?bp\ \{d\}$

proof (*rule ccontr*)

assume $\neg base\ \{0..<d\}\ x \notin grid\ (base\ \{0..<d\}\ p)\ \{d\}$

hence $base\ \{0..<d\}\ x \in grid\ (base\ \{0..<d\}\ p)\ \{d\}$ **by** *auto*

from $grid-transitive[OF\ baseE(2)[OF\ x-spg]\ this$

have $x \in grid\ (base\ \{0..<d\}\ p)\ \{0..<Suc\ d\}$ **using** $ds-eq$ **by** *auto*

thus *False* **using** $x-ngrid$ **by** *auto*

qed

have $d < length\ ?bx$ **and** $d < length\ ?bp$ **and** $d < length\ ?bx1$ **and** $d < length\ ?bp1$ **using** $base-length[OF\ x-spg]$ $base-length[OF\ p-spg]$ **and** $\langle d < dm \rangle$ **by** *auto*

have $p\text{-nochild-}x$: $?bp \notin \text{grid } ?bx \{d\}$ (**is** $?assm$)
proof (rule *ccontr*)
have ds : $\{0..<d\} \cup \{0..<Suc\ d\} = \{d\} \cup \{0..<d\}$ **by** *auto*
have $d\text{-sub}$: $\{d\} \subseteq \{0..<Suc\ d\}$ **by** *auto*
assume $\neg ?assm$ **hence** $b\text{-in-}bx$: $\text{base } \{0..<d\} p \in \text{grid } ?bx \{d\}$ **by** *auto*

have $d \notin \{0..<d\}$ **and** $d \in \{d\}$ **by** *auto*
from *grid-replace-dim*[*OF* $\langle d < \text{length } ?bx \rangle \langle d < \text{length } p \rangle$ *grid.Start*[**where**
 $b=p$ **and** $ds=\{d\}$] *b-in-bx*]
have $p \in \text{grid } ?px \{d\}$ **unfolding** *base-out*[*OF* $\langle d < dm \rangle \langle d \notin \{0..<d\} \rangle$
 $x\text{-spg}$] *base-out*[*OF* $\langle d < dm \rangle \langle d \notin \{0..<d\} \rangle$] *p-spg*] *list-update-id* .
moreover
from *grid-replace-dim*[*OF* $\langle d < \text{length } ?bx1 \rangle \langle d < \text{length } ?bp1 \rangle$ *baseE*(2)[*OF*
 $p\text{-spg}'$, **where** $ds=\{d\}$] *baseE*(2)[*OF* $x\text{-spg}$, **where** $ds=\{d\}$]]
have $?px \in \text{grid } ?bp1 \{d\}$ **unfolding** *base-in*[*OF* $\langle d < dm \rangle \langle d \in \{d\} \rangle$] $x\text{-spg}$
unfolding *base-in*[*OF* $\langle d < dm \rangle \langle d \in \{d\} \rangle$] $p\text{-spg}'$, *symmetric*] *list-update-id* .
ultimately
have $x \notin \text{grid } (\text{base } \{0..<d\} ?px) \{0..<d\}$ **using** *down-ps*[*unfolded parents-def*,
where $p'=?px$] **by** (*auto simp only*):
moreover
have $\text{base } \{0..<d\} ?px = ?bx$
proof (rule *nth-equalityI*)
from $\langle ?px \in \text{grid } ?bp1 \{d\} \rangle$ **have** $px\text{-spg}$: $?px \in \text{sparsegrid}'\ dm$ **using**
base-grid[*OF* $p\text{-spg}'$] **by** *auto*
from *base-length*[*OF* *this*] *base-length*[*OF* $x\text{-spg}$] **show** $l\text{-eq}$: $\text{length } (\text{base } \{0..<d\} ?px) = \text{length } ?bx$ **by** *auto*
show $\text{base } \{0..<d\} ?px ! i = ?bx ! i$ **if** $i < \text{length } (\text{base } \{0..<d\} ?px)$ **for** i
proof –
have $i < \text{length } ?bx$ **and** $i < dm$ **using** *that l-eq and base-length*[*OF*
 $px\text{-spg}$] **by** *auto*
show $\text{base } \{0..<d\} ?px ! i = ?bx ! i$
proof (*cases* $i < d$)
case *True* **hence** $i \in \{0..<d\}$ **by** *auto*
from *base-in*[*OF* $\langle i < dm \rangle$ *this*] **show** $?thesis$ **using** $px\text{-spg } x\text{-spg}$ **by**
auto

next
case *False* **hence** $i \notin \{0..<d\}$ **by** *auto*
have $?px ! i = x ! i$
proof (*cases* $i > d$)
have $i\text{-le}$: $i < \text{length } (\text{base } \{0..<Suc\ d\} p)$ **using** *base-length*[*OF*
 $p\text{-spg}$] **and** $\langle i < dm \rangle$ **by** *auto*
case *True* **hence** $i \notin \{0..<Suc\ d\}$ **by** *auto*
from *grid-invariant*[*OF* $i\text{-le}$ *this* $x\text{-eq}$] *base-out*[*OF* $\langle i < dm \rangle$ *this* $p\text{-spg}$]
show $?thesis$ **using** *list-update-id* **and** *True* **by** *auto*
next
case *False* **hence** $d = i$ **using** $\langle \neg i < d \rangle$ **by** *auto*
thus $?thesis$ **unfolding** $\langle d = i \rangle$ **using** $\langle i < dm \rangle \langle \text{length } p = dm \rangle$
nth-list-update-eq **by** *auto*
qed

```

      thus ?thesis using base-out[OF ‹i < dm› ‹i ∉ {0..<d}› px-spg]
base-out[OF ‹i < dm› ‹i ∉ {0..<d}› x-spg] by auto
    qed
  qed
  qed
  ultimately have x ∉ grid ?bx {0..<d} by auto
  thus False using baseE(2)[OF x-spg] by auto
qed

  have x-grid: ?bx ∈ grid (base {0..<Suc d} p) {d} using grid-shift-base[OF -
p-spg' x-eq[unfolded ds-eq]] unfolding ds-eq by auto

  have p-grid: ?bp ∈ grid (base {0..<Suc d} p) {d} using grid-shift-base[OF -
p-spg' baseE(2)[OF p-spg', where ds={0..<d} ∪ {d}]] unfolding ds-eq by auto

  have l2-φ (?bp ! d) (?bx ! d) = 0
  proof (cases lv ?bx d ≤ lv ?bp d)
    case True from l2-disjoint[OF - x-grid p-grid p-nochild-x this] ‹d < dm›
and base-length[OF p-spg']
    show ?thesis by auto
  next
    case False hence lv ?bx d ≥ lv ?bp d by auto
    from l2-disjoint[OF - p-grid x-grid x-nochild-p this] ‹d < dm› and
base-length[OF p-spg']
    show ?thesis by (auto simp: l2-commutative)
  qed
  hence l2-φ (p ! d) (x ! d) = 0 using base-out[OF ‹d < dm›] p-spg' x-spg by
auto
  hence ∃ d ∈ {0..<Suc d}. l2-φ (p ! d) (x ! d) = 0 by auto
  from prod-zero[OF - this]
  show ?F x = 0 by (auto simp: l2-commutative)
  qed
  qed
  finally show ?case .
qed

theorem updown:
  assumes p-spg: p ∈ sparsegrid dm lm
  shows updown dm lm α p = (∑ p' ∈ sparsegrid dm lm. α p' * l2 p' p)
  proof -
    have p ∈ sparsegrid' dm using p-spg unfolding sparsegrid-def sparsegrid'-def
lgrid-def by auto
    have !!p'. p' ∈ lgrid (base {0..<dm} p) {0..<dm} lm ⇒ length p' = dm
  proof -
    fix p' assume p' ∈ lgrid (base {0..<dm} p) {0..<dm} lm
    with base-grid[OF ‹p ∈ sparsegrid' dm›] have p' ∈ sparsegrid' dm unfolding
lgrid-def by auto
    thus length p' = dm by auto
  qed
  qed

```

```

thus ?thesis
  unfolding updown-def sparsegrid-def base-start-eq[OF p-spg]
  using updown'[OF - p-spg, where d=dm] p-spg[unfolding sparsegrid-def lgrid-def]
  by (auto simp: atLeast0LessThan p-spg[THEN sparsegrid-length] l2-eq)
qed

```

corollary

```

fixes  $\alpha$ 
assumes  $p: p \in \text{sparsegrid } dm \text{ } lm$ 
defines  $f_\alpha \equiv \lambda x. (\sum_{p \in \text{sparsegrid } dm \text{ } lm. } \alpha \ p \ * \ \Phi \ p \ x)$ 
shows  $\text{updown } dm \text{ } lm \ \alpha \ p = (\int x. f_\alpha \ x \ * \ \Phi \ p \ x \ \partial(\prod_M \ d \in \{.. < dm\}. \text{lborel}))$ 
unfolding updown[OF p] l2-def f $\alpha$ -def sum-distrib-right
apply (intro has-bochner-integral-integral-eq[symmetric] has-bochner-integral-sum)
apply (subst mult.assoc)
apply (intro has-bochner-integral-mult-right)
apply (simp add: sparsegrid-length)
apply (rule has-bochner-integral-integrable)
using p
apply (simp add: sparsegrid-length  $\Phi$ -def prod.distrib[symmetric])
proof (rule product-sigma-finite.product-integrable-prod)
  show product-sigma-finite ( $\lambda d. \text{lborel}$ ) ..
qed (auto intro: integrable- $\varphi 2$ )

```

end

8 Imperative Version

theory Imperative

imports UpDown-Scheme Separation-Logic-Imperative-HOL.Sep-Main

begin

type-synonym pointmap = grid-point \Rightarrow nat

type-synonym impgrid = rat array

instance rat :: heap ..

primrec rat-pair **where** rat-pair (a, b) = (of-rat a, of-rat b)

declare rat-pair.simps [simp del]

definition

zipWithA :: ('a::heap \Rightarrow 'b::heap \Rightarrow 'a::heap) \Rightarrow 'a array \Rightarrow 'b array \Rightarrow 'a array
Heap

where

```

zipWithA f a b = do {
  n  $\leftarrow$  Array.len a;
  Heap-Monad.fold-map ( $\lambda n. \text{do}$  {
    x  $\leftarrow$  Array.nth a n ;
    y  $\leftarrow$  Array.nth b n ;

```

```

    Array.upd n (f x y) a
  }) [0..<n];
  return a
}

```

theorem *zipWithA* [*sep-heap-rules*]:

fixes *xs ys* :: 'a::heap list

assumes *length xs = length ys*

shows $\langle a \mapsto_a xs * b \mapsto_a ys \rangle \text{zipWithA } f \ a \ b < \lambda r. (a \mapsto_a \text{map } (\text{case-prod } f) (\text{zip } xs \ ys)) * b \mapsto_a ys * \uparrow(a = r) >$

proof –

{ **fix** *n* and *xs* :: 'a list

let *?part-res* = $\lambda n \ xs. (\text{map } (\text{case-prod } f) (\text{zip } (\text{take } n \ xs) (\text{take } n \ ys))) @ \text{drop } n \ xs)$

assume $n \leq \text{length } xs \ \text{length } xs = \text{length } ys$

then have $\langle a \mapsto_a xs * b \mapsto_a ys \rangle \text{Heap-Monad.fold-map } (\lambda n. \text{do } \{$

$x \leftarrow \text{Array.nth } a \ n ;$

$y \leftarrow \text{Array.nth } b \ n ;$

$\text{Array.upd } n \ (f \ x \ y) \ a$

$\}) [0..<n] < \lambda r. a \mapsto_a \ ?part-res \ n \ xs * b \mapsto_a \ ys >$

proof (*induct n arbitrary: xs*)

case 0 then **show** *?case* **by** *sep-auto*

next

case (*Suc n*)

note *Suc.hyps* [*sep-heap-rules*]

have $*$: $(\ ?part-res \ n \ xs)[n := f (\ ?part-res \ n \ xs ! n) (ys ! n)] = \ ?part-res \ (Suc \ n) \ xs$

using *Suc.prem*s **by** (*simp add: nth-append take-Suc-conv-app-nth upd-conv-take-nth-drop*)

from *Suc.prem*s **show** *?case*

by (*sep-auto simp add: fold-map-append **)

qed }

note *this*[*sep-heap-rules*]

show *?thesis*

unfolding *zipWithA-def*

by (*sep-auto simp add: assms*)

qed

definition *copy-array* :: 'a::heap array \Rightarrow ('a::heap array) Heap **where**

copy-array a = $\text{Array.freeze } a \gg \text{Array.of-list}$

theorem *copy-array* [*sep-heap-rules*]:

$\langle a \mapsto_a xs \rangle \text{copy-array } a < \lambda r. a \mapsto_a xs * r \mapsto_a xs >$

unfolding *copy-array-def*

by *sep-auto*

definition *sum-array* :: rat array \Rightarrow rat array \Rightarrow unit Heap **where**

sum-array a b = $\text{zipWithA } (+) \ a \ b \gg \text{return } ()$

theorem *sum-array* [*sep-heap-rules*]:

fixes $xs\ ys :: \text{rat list}$
shows $\text{length } xs = \text{length } ys \implies \langle a \mapsto_a xs * b \mapsto_a ys \rangle \text{ sum-array } a\ b < \lambda r.$
 $(a \mapsto_a \text{map } (\lambda(a, b). a + b) (\text{zip } xs\ ys)) * b \mapsto_a ys \rangle$
unfolding sum-array-def **by** sep-auto

locale $\text{linearization} =$
fixes $dm\ lm :: \text{nat}$
fixes $pm :: \text{pointmap}$
assumes $pm: \text{bij-betw } pm (\text{sparsegrid } dm\ lm) \{..< \text{card } (\text{sparsegrid } dm\ lm)\}$
begin

lemma linearizationD :
 $p \in \text{sparsegrid } dm\ lm \implies pm\ p < \text{card } (\text{sparsegrid } dm\ lm)$
using pm **by** $(\text{auto simp: bij-betw-def})$

definition $\text{gridI} :: \text{impgrid} \Rightarrow (\text{grid-point} \Rightarrow \text{real}) \Rightarrow \text{assn}$ **where**
 $\text{gridI } a\ v =$
 $(\exists_A xs. a \mapsto_a xs * \uparrow((\forall p \in \text{sparsegrid } dm\ lm. v\ p = \text{of-rat } (xs\ !\ pm\ p)) \wedge \text{length } xs = \text{card } (\text{sparsegrid } dm\ lm)))$

lemma gridI-nth-rule [sep-heap-rules]:
 $g \in \text{sparsegrid } dm\ lm \implies \langle \text{gridI } a\ v \rangle \text{Array.nth } a\ (pm\ g) < \lambda r. \text{gridI } a\ v * \uparrow$
 $(\text{of-rat } r = v\ g) \rangle$
using pm **by** $(\text{sep-auto simp: bij-betw-def gridI-def})$

lemma gridI-upd-rule [sep-heap-rules]:
 $g \in \text{sparsegrid } dm\ lm \implies$
 $\langle \text{gridI } a\ v \rangle \text{Array.upd } (pm\ g)\ x\ a < \lambda a'. \text{gridI } a\ (\text{fun-upd } v\ g\ (\text{of-rat } x)) * \uparrow$
 $(a' = a) \rangle$
unfolding gridI-def **using** pm
by $(\text{sep-auto simp: bij-betw-def inj-onD intro!: nth-list-update-eq[symmetric] nth-list-update-neq[symmetric]})$

primrec $\text{upI}' :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{grid-point} \Rightarrow \text{impgrid} \Rightarrow (\text{rat} * \text{rat}) \text{Heap}$ **where**
 $\text{upI}'\ d\ \ \ \ \ 0\ p\ a = \text{return } (0, 0) \mid$
 $\text{upI}'\ d\ (\text{Suc } l)\ p\ a = \text{do } \{$
 $\ \ (\fl, fml) \leftarrow \text{upI}'\ d\ l\ (\text{child } p\ \text{left } d)\ a ;$
 $\ \ (fmr, fr) \leftarrow \text{upI}'\ d\ l\ (\text{child } p\ \text{right } d)\ a ;$
 $\ \ \text{val} \leftarrow \text{Array.nth } a\ (pm\ p) ;$
 $\ \ \text{Array.upd } (pm\ p)\ (fml + fmr)\ a ;$
 $\ \ \text{let } \text{result} = ((fml + fmr + \text{val} / 2 \wedge (lv\ p\ d) / 2) / 2) ;$
 $\ \ \text{return } (\fl + \text{result}, fr + \text{result})$
 $\ \}$

lemma upI' [sep-heap-rules]:
assumes $\text{lin}[\text{simp}]$: $d < dm$
and l : $\text{level } p + l = lm\ l = 0 \vee p \in \text{sparsegrid } dm\ lm$
shows $\langle \text{gridI } a\ v \rangle \text{upI}'\ d\ l\ p\ a < \lambda r. \text{let } (r', v') = \text{up}'\ d\ l\ p\ v \text{ in } \text{gridI } a\ v' * \uparrow$
 $(\text{rat-pair } r = r') \rangle$
using l

```

proof (induct l arbitrary: p v)
  note rat-pair.simps [simp]
  case 0 then show ?case by sep-auto
next
  case (Suc l)
  from Suc.prem1 ⟨d < dm⟩
  have [simp]: level (child p left d) + l = lm level (child p right d) + l = lm p ∈
  sparsegrid dm lm
    by (auto simp: sparsegrid-length)

  have [simp]: child p left d ∉ sparsegrid dm lm ⟹ l = 0 child p right d ∉
  sparsegrid dm lm ⟹ l = 0
    using Suc.prem1 by (auto simp: sparsegrid-def lgrid-def)

  note Suc(1)[sep-heap-rules]
  show ?case
  by (sep-auto split: prod.split simp: of-rat-add of-rat-divide of-rat-power of-rat-mult
  rat-pair-def Let-def)
qed

```

```

primrec downI' :: nat ⇒ nat ⇒ grid-point ⇒ impgrid ⇒ rat ⇒ rat ⇒ unit Heap
where

```

```

  downI' d 0 p a fl fr = return () |
  downI' d (Suc l) p a fl fr = do {
    val ← Array.nth a (pm p) ;
    let fm = ((fl + fr) / 2 + val) ;
    Array.upd (pm p) (((fl + fr) / 4 + (1 / 3) * val) / 2 ^ (lv p d)) a ;
    downI' d l (child p left d) a fl fm ;
    downI' d l (child p right d) a fm fr
  }

```

```

lemma downI' [sep-heap-rules]:

```

```

  assumes lin[simp]: d < dm
  and l: level p + l = lm l = 0 ∨ p ∈ sparsegrid dm lm
  shows < gridI a v > downI' d l p a fl fr < λr. gridI a (down' d l p (of-rat fl)
  (of-rat fr) v) >
  using l

```

```

proof (induct l arbitrary: p v fl fr)

```

```

  note rat-pair.simps [simp]
  case 0 then show ?case by sep-auto

```

```

next

```

```

  case (Suc l)
  from Suc.prem1 ⟨d < dm⟩
  have [simp]: level (child p left d) + l = lm level (child p right d) + l = lm p ∈
  sparsegrid dm lm
    by (auto simp: sparsegrid-length)

```

```

  have [simp]: child p left d ∉ sparsegrid dm lm ⟹ l = 0 child p right d ∉
  sparsegrid dm lm ⟹ l = 0

```

using *Suc.prem*s by (auto simp: sparsegrid-def lgrid-def)

note *Suc(1)[sep-heap-rules]*

show *?case*

by (*sep-auto split: prod.split simp: of-rat-add of-rat-divide of-rat-power of-rat-mult rat-pair-def Let-def fun-upd-def*)

qed

definition *liftI* :: (nat \Rightarrow nat \Rightarrow grid-point \Rightarrow impgrid \Rightarrow unit Heap) \Rightarrow nat \Rightarrow impgrid \Rightarrow unit Heap **where**

liftI *f* *d* *a* =

foldr (λ *p* *n*. *n* \gg *f* *d* (*lm* - level *p*) *p* *a*) (*gridgen* (*start* *dm*) ($\{0..<dm\} - \{d\}$) *lm*) (*return* ())

theorem *liftI* [*sep-heap-rules*]:

assumes *d* < *dm*

and *f* [*sep-heap-rules*]: $\bigwedge v$ *p*. *p* \in *lgrid* (*start* *dm*) ($\{0..<dm\} - \{d\}$) *lm* \implies

< *gridI* *a* *v* > *f* *d* (*lm* - level *p*) *p* *a* < λ r. *gridI* *a* (*f'* *d* (*lm* - level *p*) *p* *v*) >

shows < *gridI* *a* *v* > *liftI* *f* *d* *a* < λ r. *gridI* *a* (*Grid.lift* *f'* *dm* *lm* *d* *v*) >

proof -

let *?ds* = $\{0..<dm\} - \{d\}$ **and** *?g* = *gridI* *a*

{ fix *ps* **assume** *set* *ps* \subseteq *set* (*gridgen* (*start* *dm*) *?ds* *lm*) **and** *distinct* *ps*

then have < *?g* *v* >

foldr (λ *p* *n*. (*n* :: unit Heap) \gg *f* *d* (*lm* - level *p*) *p* *a*) *ps* (*return* ())

< λ r. *?g* (*foldr* (λ *p* α . *f'* *d* (*lm* - level *p*) *p* α) *ps* *v*) >

by (*induct* *ps* *arbitrary: v*) (*sep-auto simp: gridgen-lgrid-eq*) **+** }

from *this* [*OF subset-refl gridgen-distinct*]

show *?thesis*

by (*simp add: liftI-def Grid.lift-def*)

qed

definition *upI* **where** *upI* = *liftI* (λ *d* *l* *p* *a*. *upI'* *d* *l* *p* *a* \gg *return* ())

theorem *upI* [*sep-heap-rules*]:

assumes [*simp*]: *d* < *dm*

shows < *gridI* *a* *v* > *upI* *d* *a* < λ r. *gridI* *a* (*up* *dm* *lm* *d* *v*) >

unfolding *up-def upI-def*

by (*sep-auto simp: lgrid-def sparsegrid-def lgrid-def split: prod.split intro: grid-union-dims*[of $\{0..<dm\} - \{d\}$ $\{0..<dm\}$])

definition *downI* **where** *downI* = *liftI* (λ *d* *l* *p* *a*. *downI'* *d* *l* *p* *a* 0 0)

theorem *downI* [*sep-heap-rules*]:

assumes [*simp*]: *d* < *dm*

shows < *gridI* *a* *v* > *downI* *d* *a* < λ r. *gridI* *a* (*down* *dm* *lm* *d* *v*) >

unfolding *down-def downI-def*

by (*sep-auto simp: lgrid-def sparsegrid-def lgrid-def split: prod.split intro: grid-union-dims*[of $\{0..<dm\} - \{d\}$ $\{0..<dm\}$])

theorem *copy-array-gridI* [*sep-heap-rules*]:
 $\langle \text{gridI } a \ v \rangle \text{ copy-array } a \ \langle \lambda r. \text{gridI } a \ v * \text{gridI } r \ v \rangle$
unfolding *gridI-def*
by *sep-auto*

theorem *sum-array-gridI* [*sep-heap-rules*]:
 $\langle \text{gridI } a \ v * \text{gridI } b \ w \rangle \text{ sum-array } a \ b \ \langle \lambda r. \text{gridI } a \ (\text{sum-vector } v \ w) * \text{gridI } b \ w \rangle$
unfolding *gridI-def*
by (*sep-auto simp: sum-vector-def nth-map linearizationD of-rat-add*)

primrec *updownI'* :: *nat* \Rightarrow *impgrid* \Rightarrow *unit Heap* **where**
updownI' 0 a = return () |
updownI' (Suc d) a = do {
b \leftarrow *copy-array a* ;
upI d a ;
updownI' d a ;
updownI' d b ;
downI d b ;
sum-array a b
}

theorem *updownI'* [*sep-heap-rules*]:
 $d \leq dm \implies \langle \text{gridI } a \ v \rangle \text{ updownI' } d \ a \ \langle \lambda r. \text{gridI } a \ (\text{updown' } dm \ lm \ d \ v) \rangle_t$
proof (*induct d arbitrary: a v*)
case (*Suc d*)
note *Suc.hyps* [*sep-heap-rules*]
from *Suc.prem*s **show** *?case*
by *sep-auto*
qed *sep-auto*

definition *updownI* **where** *updownI a = updownI' dm a*

theorem *updownI* [*sep-heap-rules*]:
 $\langle \text{gridI } a \ v \rangle \text{ updownI } a \ \langle \lambda r. \text{gridI } a \ (\text{updown } dm \ lm \ v) \rangle_t$
unfolding *updown-def updownI-def* **by** *sep-auto*

end

end

Literatur

- [1] J. Hölzl. Automatischer Korrektheitsbeweis von Algorithmen zur hierarchischen Basistransformation. IDP, Institut für Informatik, Technische Universität München, 2009.
- [2] D. Pflüger. *Spatially Adaptive Sparse Grids for High-Dimensional*

Problems. Verlag Dr. Hut, München, Aug. 2010.