

Verification of the UpDown scheme

Johannes Hölzl

13. September 2023

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	4
2.1	Vectors	4
2.2	Inductive enumeration of all grid points	5
2.3	Grid Restricted to a Level	7
2.4	Unbounded Sparse Grid	8
2.5	Sparse Grid	8
2.6	Compute Sparse Grid Points	9
2.7	Base Points	9
2.8	Lift Operation over all Grid Points	11
2.9	Parent Points	11
3	Hat Functions	12

4	UpDown Scheme	16
5	Up Part	16
6	Down part	17
7	UpDown	18
8	Imperative Version	18

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  $\text{level } p' = \text{level } p$ 
  ⟨proof⟩
```

```
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
  ⟨proof⟩
```

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

<proof>

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]*: $length\ (child\ p\ dir\ d) = length\ p$
<proof>

lemma *child-odd[simp]*: $d < length\ p \implies odd\ (ix\ (child\ p\ dir\ d)\ d)$
<proof>

lemma *child-eq*: $p ! d = (l, i) \implies \exists j. child\ p\ dir\ d = p[d := (l + 1, j)]$
<proof>

lemma *child-other*: $d \neq d' \implies child\ p\ dir\ d ! d' = p ! d'$
<proof>

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

lemma *child-single-level*: $d < length\ p \implies lv\ (child\ p\ dir\ d)\ d > lv\ p\ d$
<proof>

lemma *child-lv*: $d < length\ p \implies lv\ (child\ p\ dir\ d)\ d = lv\ p\ d + 1$
<proof>

lemma *child-lv-other*: **assumes** $d' \neq d$ **shows** $lv\ (child\ p\ dir\ d')\ d = lv\ p\ d$
<proof>

lemma *child-ix-left*: $d < length\ p \implies ix\ (child\ p\ left\ d)\ d = 2 * ix\ p\ d - 1$
<proof>

lemma *child-ix-right*: $d < length\ p \implies ix\ (child\ p\ right\ d)\ d = 2 * ix\ p\ d + 1$
<proof>

lemma *child-ix*: $d < length\ p \implies ix\ (child\ p\ dir\ d)\ d = 2 * ix\ p\ d + sgn\ dir$
<proof>

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

lemma *child-ex-neighbour*: $\exists b'. child\ b\ dir\ d = child\ b'\ (inv\ dir)\ d$
<proof>

lemma *child-level-gt[simp]*: $level\ (child\ p\ dir\ d) \geq level\ p$
<proof>

lemma *child-estimate-child*:
assumes $d < \text{length } p$
and $l \leq \text{lv } p \ d$
and *i'-range*: $ix \ p \ d < (i + 1) * 2^{\wedge}(\text{lv } p \ d - l) \wedge$
 $ix \ p \ d > (i - 1) * 2^{\wedge}(\text{lv } p \ d - l)$
(is *?top* $p \wedge$ *?bottom* $p)$
and *is-child*: $p' = \text{child } p \ \text{dir } d$
shows $?top \ p' \wedge ?bottom \ p'$
 $\langle \text{proof} \rangle$

lemma *child-neighbour*: **assumes** $\text{child } p \ (\text{inv } \text{dir}) \ d = \text{child } ps \ \text{dir } d$ **(is** *?c-p* $=$
?c-ps)
shows $ps = p[d := (\text{lv } p \ d, ix \ p \ d - \text{sgn } \text{dir})]$ **(is** $- = ?ps$)
 $\langle \text{proof} \rangle$

definition *start* :: $\text{nat} \Rightarrow \text{grid-point}$
where
 $\text{start } dm = \text{replicate } dm \ (0, 1)$

lemma *start-lv[simp]*: $d < dm \Longrightarrow \text{lv } (\text{start } dm) \ d = 0$
 $\langle \text{proof} \rangle$

lemma *start-ix[simp]*: $d < dm \Longrightarrow ix \ (\text{start } dm) \ d = 1$
 $\langle \text{proof} \rangle$

lemma *start-length[simp]*: $\text{length } (\text{start } dm) = dm$
 $\langle \text{proof} \rangle$

lemma *level-start-0[simp]*: $\text{level } (\text{start } dm) = 0$
 $\langle \text{proof} \rangle$

end

2 Sparse Grids

theory *Grid*
imports *Grid-Point*
begin

2.1 Vectors

type-synonym *vector* = $\text{grid-point} \Rightarrow \text{real}$

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

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

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 *grid b 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]*: $p' \in \text{grid } p \text{ ds} \Longrightarrow \text{length } p' = \text{length } p$

<proof>

lemma *grid-union-dims*: $\llbracket ds \subseteq ds' ; p \in \text{grid } b \text{ ds} \rrbracket \Longrightarrow p \in \text{grid } b \text{ ds}'$

<proof>

lemma *grid-transitive*: $\llbracket a \in \text{grid } b \text{ ds} ; b \in \text{grid } c \text{ ds}' ; ds' \subseteq ds'' ; ds \subseteq ds'' \rrbracket \Longrightarrow a \in \text{grid } c \text{ ds}''$

<proof>

lemma *grid-child[intro?]*: **assumes** $d \in ds$ **and** *p-grid*: $p \in \text{grid } (\text{child } b \text{ dir } d) \text{ ds}$
shows $p \in \text{grid } b \text{ ds}$

<proof>

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

shows $lv \ b \ d \leq lv \ p \ d$

<proof>

lemma *grid-child-level*:

assumes $d < \text{length } b$

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

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

<proof>

lemma *child-out*: $\text{length } p \leq d \Longrightarrow \text{child } p \text{ dir } d = p$

<proof>

lemma *grid-dim-remove*:

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

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

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

<proof>

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

<proof>

lemma *grid-dim-remove-outer*: $\text{grid } b \ ds = \text{grid } b \ \{d \in ds. d < \text{length } b\}$

<proof>

lemma *grid-level[intro]*: **assumes** $p \in \text{grid } b \text{ } ds$ **shows** $\text{level } b \leq \text{level } p$
 $\langle \text{proof} \rangle$

lemma *grid-empty-ds[simp]*: $\text{grid } b \ \{\} = \{ b \}$
 $\langle \text{proof} \rangle$

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

lemma *grid-estimate*:
assumes $d < \text{length } b$ **and** *p-grid*: $p \in \text{grid } b \text{ } ds$
shows $ix \ p \ d < (ix \ b \ d + 1) * 2^{\wedge}(lv \ p \ d - lv \ b \ d) \wedge ix \ p \ d > (ix \ b \ d - 1) * 2^{\wedge}(lv \ p \ d - lv \ b \ d)$
 $\langle \text{proof} \rangle$

lemma *grid-odd*: **assumes** $d < \text{length } b$ **and** *p-diff*: $p ! d \neq b ! d$ **and** *p-grid*: $p \in \text{grid } b \text{ } ds$
shows $odd \ (ix \ p \ d)$
 $\langle \text{proof} \rangle$

lemma *grid-invariant*: **assumes** $d < \text{length } b$ **and** $d \notin ds$ **and** *p-grid*: $p \in \text{grid } b \text{ } ds$
shows $p ! d = b ! d$
 $\langle \text{proof} \rangle$

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 $p' \in \text{grid } p \ \{d\}$
 $\langle \text{proof} \rangle$

lemma *grid-disjunct*: **assumes** $d < \text{length } p$
shows $\text{grid } (\text{child } p \ \text{left } d) \ ds \cap \text{grid } (\text{child } p \ \text{right } d) \ ds = \{\}$
(**is** $\text{grid } ?l \ ds \cap \text{grid } ?r \ ds = \{\}$)
 $\langle \text{proof} \rangle$

lemma *grid-level-eq*: **assumes** *eq*: $\forall d \in ds. lv \ p \ d = lv \ b \ d$ **and** *grid*: $p \in \text{grid } b \text{ } ds$
shows $\text{level } p = \text{level } b$
 $\langle \text{proof} \rangle$

lemma *grid-partition*:
 $\text{grid } p \ \{d\} = \{p\} \cup \text{grid } (\text{child } p \ \text{left } d) \ \{d\} \cup \text{grid } (\text{child } p \ \text{right } d) \ \{d\}$
(**is** $- = - \cup \text{grid } ?l \ \{d\} \cup \text{grid } ?r \ \{d\}$)
 $\langle \text{proof} \rangle$

lemma *grid-change-dim*: **assumes** *grid*: $p \in \text{grid } b \text{ } ds$
shows $p[d := X] \in \text{grid } (b[d := X]) \ ds$
 $\langle \text{proof} \rangle$

lemma *grid-change-dim-child*: **assumes** *grid*: $p \in \text{grid } b \text{ } ds$ **and** $d \notin ds$
shows $\text{child } p \ \text{dir } d \in \text{grid } (\text{child } b \ \text{dir } d) \ ds$
 $\langle \text{proof} \rangle$

lemma *grid-split*: **assumes** *grid*: $p \in \text{grid } b \ (ds' \cup ds)$ **shows** $\exists x \in \text{grid } b \ ds. p$

$\in \text{grid } x \text{ } ds'$
 $\langle \text{proof} \rangle$

lemma *grid-union-eq*: $(\bigcup p \in \text{grid } b \text{ } ds. \text{grid } p \text{ } ds') = \text{grid } b \text{ } (ds' \cup ds)$
 $\langle \text{proof} \rangle$

lemma *grid-onedim-split*:

$\text{grid } b \text{ } (ds \cup \{d\}) = \text{grid } b \text{ } ds \cup \text{grid } (\text{child } b \text{ left } d) \text{ } (ds \cup \{d\}) \cup \text{grid } (\text{child } b \text{ right } d) \text{ } (ds \cup \{d\})$

$(\text{is } - = ?g \cup ?l \text{ } (ds \cup \{d\}) \cup ?r \text{ } (ds \cup \{d\}))$

$\langle \text{proof} \rangle$

lemma *grid-child-without-parent*: **assumes** *grid*: $p \in \text{grid } (\text{child } b \text{ dir } d) \text{ } ds$ **(is** $p \in \text{grid } ?c \text{ } ds)$ **and** $d < \text{length } b$

shows $p \neq b$

$\langle \text{proof} \rangle$

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 $x \notin \text{grid } p' \text{ } ds'$

$\langle \text{proof} \rangle$

lemma *grid-split1*: **assumes** *grid*: $p \in \text{grid } b \text{ } (ds' \cup ds)$ **and** $ds \cap ds' = \{\}$

shows $\exists! x \in \text{grid } b \text{ } ds. p \in \text{grid } x \text{ } ds'$

$\langle \text{proof} \rangle$

2.3 Grid Restricted to a Level

definition *lgrid* :: *grid-point* \Rightarrow *nat set* \Rightarrow *nat* \Rightarrow *grid-point set*

where $\text{lgrid } b \text{ } ds \text{ } lm = \{ p \in \text{grid } b \text{ } ds. \text{level } p < lm \}$

lemma *lgridI[intro]*:

$\llbracket p \in \text{grid } b \text{ } ds ; \text{level } p < lm \rrbracket \Longrightarrow p \in \text{lgrid } b \text{ } ds \text{ } lm$

$\langle \text{proof} \rangle$

lemma *lgridE[elim]*:

assumes $p \in \text{lgrid } b \text{ } ds \text{ } lm$

assumes $\llbracket p \in \text{grid } b \text{ } ds ; \text{level } p < lm \rrbracket \Longrightarrow P$

shows P

$\langle \text{proof} \rangle$

lemma *lgridI-child[intro]*:

$d \in ds \Longrightarrow p \in \text{lgrid } (\text{child } b \text{ dir } d) \text{ } ds \text{ } lm \Longrightarrow p \in \text{lgrid } b \text{ } ds \text{ } lm$

$\langle \text{proof} \rangle$

lemma *lgrid-empty[simp]*: $\text{lgrid } p \text{ } ds \text{ } (\text{level } p) = \{\}$

$\langle \text{proof} \rangle$

lemma *lgrid-empty'*: **assumes** $lm \leq \text{level } p$ **shows** $\text{lgrid } p \text{ } ds \text{ } lm = \{\}$

$\langle \text{proof} \rangle$

lemma *grid-not-child*:

assumes [*simp*]: $d < \text{length } p$

shows $p \notin \text{grid } (\text{child } p \text{ dir } d) \text{ ds}$
(proof)

2.4 Unbounded Sparse Grid

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

$\text{sparsegrid}' \text{ dm} = \text{grid } (\text{start } \text{dm}) \{ 0 \dots < \text{dm} \}$

lemma $\text{grid-subset-alldim}$:
assumes $p: p \in \text{grid } b \text{ ds}$
defines $\text{dm} \equiv \text{length } b$
shows $p \in \text{grid } b \{ 0 \dots < \text{dm} \}$
(proof)

lemma $\text{sparsegrid}'\text{-length[simp]}$:
 $b \in \text{sparsegrid}' \text{ dm} \implies \text{length } b = \text{dm}$ (proof)

lemma $\text{sparsegrid}'I[\text{intro}]$:
assumes $b: b \in \text{sparsegrid}' \text{ dm}$ **and** $p: p \in \text{grid } b \text{ ds}$
shows $p \in \text{sparsegrid}' \text{ dm}$
(proof)

lemma $\text{sparsegrid}'\text{-start}$:
assumes $b \in \text{grid } (\text{start } \text{dm}) \text{ ds}$
shows $b \in \text{sparsegrid}' \text{ dm}$
(proof)

2.5 Sparse Grid

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

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

lemma sparsegrid-length : $p \in \text{sparsegrid } \text{dm } \text{lm} \implies \text{length } p = \text{dm}$
(proof)

lemma $\text{sparsegrid-subset[intro]}$: $p \in \text{sparsegrid } \text{dm } \text{lm} \implies p \in \text{sparsegrid}' \text{ dm}$
(proof)

lemma $\text{sparsegrid}I[\text{intro}]$:
assumes $p \in \text{sparsegrid}' \text{ dm}$ **and** $\text{level } p < \text{lm}$
shows $p \in \text{sparsegrid } \text{dm } \text{lm}$
(proof)

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

lemma *sparsegridE*[*elim*]:
assumes $p \in \text{sparsegrid } dm \text{ } lm$
shows $p \in \text{sparsegrid}' dm$ **and** $\text{level } p < lm$
 $\langle \text{proof} \rangle$

2.6 Compute Sparse Grid Points

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

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

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

lemma *gridgen-distinct*: $\text{distinct } (\text{gridgen } p \text{ } ds \text{ } l)$
 $\langle \text{proof} \rangle$

lemma *lgrid-finite*: $\text{finite } (\text{lgrid } b \text{ } ds \text{ } lm)$
 $\langle \text{proof} \rangle$

lemma *lgrid-sum*:
fixes $F :: \text{grid-point} \Rightarrow \text{real}$
assumes $d < \text{length } b$ **and** $\text{level } b < lm$
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 $(\sum p \in ?\text{grid } b. F p) = (\sum p \in ?\text{grid } ?l . F p) + (?sum (?grid ?r)) + F b$
 $\langle \text{proof} \rangle$

2.7 Base Points

definition *base* :: *nat set* \Rightarrow *grid-point* \Rightarrow *grid-point*
where $\text{base } ds \text{ } p = (\text{THE } b. b \in \text{grid } (\text{start } (\text{length } p)) (\{0 ..< \text{length } p\} - ds) \wedge$
 $p \in \text{grid } b \text{ } ds)$

lemma *baseE*: **assumes** *p-grid*: $p \in \text{sparsegrid}' dm$
shows $\text{base } ds \text{ } p \in \text{grid } (\text{start } dm) (\{0 ..< dm\} - ds)$
and $p \in \text{grid } (\text{base } ds \text{ } p) \text{ } ds$
 $\langle \text{proof} \rangle$

lemma *baseI*: **assumes** *x-grid*: $x \in \text{grid } (\text{start } dm) (\{0 ..< dm\} - ds)$ **and** *p-xgrid*:
 $p \in \text{grid } x \text{ } ds$
shows $\text{base } ds \text{ } p = x$
 $\langle \text{proof} \rangle$

lemma *base-empty*: **assumes** *p-grid*: $p \in \text{sparsegrid}' dm$ **shows** $\text{base } \{\} p = p$

<proof>

lemma *base-start-eq*: **assumes** *p-spg*: $p \in \text{sparsegrid } dm \text{ } lm$
shows $\text{start } dm = \text{base } \{0..<dm\} p$
<proof>

lemma *base-in-grid*: **assumes** *p-grid*: $p \in \text{sparsegrid}' dm$ **shows** $\text{base } ds p \in \text{grid } (\text{start } dm) \{0..<dm\}$
<proof>

lemma *base-grid*: **assumes** *p-grid*: $p \in \text{sparsegrid}' dm$ **shows** $\text{grid } (\text{base } ds p) ds \subseteq \text{sparsegrid}' dm$
<proof>

lemma *base-length[simp]*: **assumes** *p-grid*: $p \in \text{sparsegrid}' dm$ **shows** $\text{length } (\text{base } ds p) = dm$
<proof>

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

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

lemma *base-base*: **assumes** *p-grid*: $p \in \text{sparsegrid}' dm$ **shows** $\text{base } ds (\text{base } ds' p) = \text{base } (ds \cup ds') p$
<proof>

lemma *grid-base-out*: **assumes** $d < dm$ **and** $d \notin ds$ **and** *p-grid*: $b \in \text{sparsegrid}' dm$ **and** $p \in \text{grid } (\text{base } ds b) ds$ **shows** $p ! d = b ! d$
<proof>

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>

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>

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>

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>

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

<proof>

lemma *grid-replace-dim*: **assumes** $d < \text{length } b'$ **and** $d < \text{length } b$ **and** $p\text{-grid}$: $p \in \text{grid } b \text{ } ds$ **and** $p'\text{-grid}$: $p' \in \text{grid } b' \text{ } ds$

shows $p[d := p' ! d] \in \text{grid } (b[d := b' ! d]) \text{ } ds$ (**is** $- \in \text{grid } ?b \text{ } ds$)

<proof>

lemma *grid-shift-base*:

assumes $ds\text{-dj}$: $ds \cap ds' = \{\}$ **and** $b\text{-spg}$: $b \in \text{sparsegrid}' \text{ } dm$ **and** $p\text{-grid}$: $p \in \text{grid } (\text{base } (ds' \cup ds) \text{ } b) \text{ } (ds' \cup ds)$

shows $\text{base } ds' \text{ } p \in \text{grid } (\text{base } (ds \cup ds') \text{ } b) \text{ } ds$

<proof>

2.8 Lift Operation over all Grid Points

definition *lift* :: $(\text{nat} \Rightarrow \text{nat} \Rightarrow \text{grid-point} \Rightarrow \text{vector} \Rightarrow \text{vector}) \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{vector} \Rightarrow \text{vector}$

where $\text{lift } f \text{ } dm \text{ } lm \text{ } d = \text{foldr } (\lambda p. f \text{ } d \text{ } (lm - \text{level } p) \text{ } p) \text{ } (\text{gridgen } (\text{start } dm) \text{ } (\{ 0 ..< dm \} - \{ d \}) \text{ } lm)$

lemma *lift*:

assumes $d < dm$ **and** $p \in \text{sparsegrid } dm \text{ } lm$

and $F\text{intro}$: $\bigwedge l \text{ } b \text{ } p \text{ } \alpha. \llbracket b \in \text{lgrid } (\text{start } dm) \text{ } (\{ 0 ..< dm \} - \{ d \}) \text{ } lm ;$
 $l + \text{level } b = lm ; p \in \text{sparsegrid } dm \text{ } lm \rrbracket$

$\implies F \text{ } d \text{ } l \text{ } b \text{ } \alpha \text{ } p = (\text{if } b = \text{base } \{ d \} \text{ } p$
 $\text{then } (\sum p' \in \text{lgrid } b \text{ } \{ d \} \text{ } lm. S (\alpha \text{ } p') \text{ } p \text{ } p')$
 $\text{else } \alpha \text{ } p)$

shows $\text{lift } F \text{ } dm \text{ } lm \text{ } d \text{ } \alpha \text{ } p = (\sum p' \in \text{lgrid } (\text{base } \{ d \} \text{ } p) \text{ } \{ d \} \text{ } lm. S (\alpha \text{ } p') \text{ } p \text{ } p')$
(**is** $?lift = ?S \text{ } p \text{ } \alpha$)

<proof>

2.9 Parent Points

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

where $\text{parents } d \text{ } b \text{ } p = \{ x \in \text{grid } b \text{ } \{ d \}. p \in \text{grid } x \text{ } \{ d \} \}$

lemma *parents-split*: **assumes** $p\text{-grid}$: $p \in \text{grid } (\text{child } b \text{ } \text{dir } d) \text{ } \{ d \}$

shows $\text{parents } d \text{ } b \text{ } p = \{ b \} \cup \text{parents } d \text{ } (\text{child } b \text{ } \text{dir } d) \text{ } p$

<proof>

lemma *parents-no-parent*: **assumes** $d < \text{length } b$ **shows** $b \notin \text{parents } d \text{ } (\text{child } b \text{ } \text{dir } d) \text{ } p$ (**is** $- \notin \text{parents } - \text{ } ?ch \text{ } -$)

<proof>

lemma *parents-subset-lgrid*: $\text{parents } d \text{ } b \text{ } p \subseteq \text{lgrid } b \text{ } \{ d \}$ ($\text{level } p + 1$)

<proof>

lemma *parents-finite*: $\text{finite } (\text{parents } d \text{ } b \text{ } p)$

<proof>

lemma *parent-sum*: **assumes** $p\text{-grid}$: $p \in \text{grid } (\text{child } b \text{ } \text{dir } d) \text{ } \{ d \}$ **and** $d < \text{length } b$

shows $(\sum x \in \text{parents } d \ b \ p. F \ x) = F \ b + (\sum x \in \text{parents } d \ (\text{child } b \ \text{dir } d) \ p. F \ x)$
 <proof>

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

lemma *grid-single-dimensional-specification*:

assumes $d < \text{length } b$
and $\text{odd } i$
and $lv \ b \ d + l' = l$
and $i < (ix \ b \ d + 1) * 2^{l'}$
and $i > (ix \ b \ d - 1) * 2^{l'}$
shows $b[d := (l, i)] \in \text{grid } b \ \{d\}$
 <proof>

lemma *grid-multi-dimensional-specification*:

assumes $dm \leq \text{length } b$ **and** $\text{length } p = \text{length } b$
and $\bigwedge d. d < dm \implies$
 $\text{odd } (ix \ p \ d) \wedge$
 $lv \ b \ d \leq lv \ p \ d \wedge$
 $ix \ p \ d < (ix \ b \ d + 1) * 2^{(lv \ p \ d - lv \ b \ d)} \wedge$
 $ix \ p \ d > (ix \ b \ d - 1) * 2^{(lv \ p \ d - lv \ b \ d)}$
(is $\bigwedge d. d < dm \implies ?\text{bounded } p \ d)$
and $\bigwedge d. \llbracket dm \leq d ; d < \text{length } b \rrbracket \implies p ! d = b ! d$
shows $p \in \text{grid } b \ \{0..<dm\}$
 <proof>

lemma *sparsegrid*:

$\text{sparsegrid } dm \ lm = \{p.$
 $\text{length } p = dm \wedge \text{level } p < lm \wedge$
 $(\forall d < dm. \text{odd } (ix \ p \ d) \wedge 0 < ix \ p \ d \wedge ix \ p \ d < 2^{(lv \ p \ d + 1)})\}$
(is $- = ?\text{set}$)
 <proof>

end

3 Hat Functions

theory *Triangular-Function*

imports

HOL-Analysis.Equivalence-Lebesgue-Henstock-Integration

Grid

begin

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

fixes $f :: - \Rightarrow 'a::\text{linorder-topology}$
shows $\text{continuous-on } S \ f \implies \text{continuous-on } S \ g \implies \text{continuous-on } S \ (\lambda x. \max (f \ x) (g \ x))$

<proof>

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

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

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

$$\Phi p x = (\prod d < \text{length } p. \varphi (p ! d) (x d))$$

definition $l2\text{-}\varphi$ **where**

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

definition $l2$ **where**

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

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

<proof>

lemma $\varphi\text{-nonneg}$: $0 \leq \varphi p x$

<proof>

lemma $\varphi\text{-zero-iff}$:

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

<proof>

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

<proof>

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

<proof>

lemma $ix\text{-lt}$: $p \in \text{sparsegrid } dm \text{ } lm \implies d < dm \implies ix p d < 2^{\wedge}(lv p d + 1)$

<proof>

lemma $ix\text{-gt}$: $p \in \text{sparsegrid } dm \text{ } lm \implies d < dm \implies 0 < ix p d$

<proof>

lemma $\Phi\text{-eq-0}$: **assumes** $x: \exists d < \text{length } p. x d < 0 \vee 1 < x d$ **and** $p: p \in \text{sparsegrid } dm \text{ } lm$ **shows** $\Phi p x = 0$

<proof>

lemma $\varphi\text{-left-support'}$:

$$x \in \{\text{real-of-int } (i-1) / 2^{\wedge}(l+1) .. \text{real-of-int } i / 2^{\wedge}(l+1)\} \implies \varphi (l,i) x = 1 + x * 2^{\wedge}(l+1) - \text{real-of-int } i$$

<proof>

lemma $\varphi\text{-left-support}$: $x \in \{-1 .. 0::\text{real}\} \implies \varphi (l,i) ((x + \text{real-of-int } i) / 2^{\wedge}l)$

+ 1)) = 1 + x
 ⟨proof⟩

lemma φ -right-support':

$x \in \{\text{real-of-int } i / 2^{\wedge}(l+1) .. \text{real-of-int } (i+1) / 2^{\wedge}(l+1)\} \implies \varphi(l, i) x = 1 - x * 2^{\wedge}(l+1) + \text{real-of-int } i$
 ⟨proof⟩

lemma φ -right-support:

$x \in \{0 .. 1::\text{real}\} \implies \varphi(l, i) ((x + \text{real } i) / 2^{\wedge}(l+1)) = 1 - x$
 ⟨proof⟩

lemma integrable- φ : integrable lborel (φ p)

⟨proof⟩

lemma integrable- φ 2: integrable lborel ($\lambda x. \varphi p x * \varphi q x$)

⟨proof⟩

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

$DERIV \Phi\text{-}n x :> (\varphi(l', i') x * \varphi(l, i) x) \text{ (is } \bigwedge x. x \in \{?a..?b\} \implies DERIV -$

$-\text{:} > ?P x)$
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⟩

lemma l2-eq: length $a =$ length $b \implies l2 a b = (\prod d < \text{length } a. l2\text{-}\varphi(a!d)(b!d))$

⟨proof⟩

lemma l2-when-disjoint:

assumes $l \leq l'$

defines $d == l' - l$

assumes $(i+1) * 2^{\wedge}d < i' \vee i' < (i-1) * 2^{\wedge}d$ (is ?right \vee ?left)

shows $l2\text{-}\varphi(l', i')(l, i) = 0$

⟨proof⟩

lemma l2-commutative: $l2\text{-}\varphi p q = l2\text{-}\varphi q p$

⟨proof⟩

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

⟨proof⟩

lemma l2-when-left-child:

assumes $l < l'$

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

and $i'\text{-top}: i' < i * 2^{\wedge}(l'-l)$

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

lemma *l2-when-right-child*:

assumes $l < l'$
and *i'-bot*: $i' > i * 2^{\wedge}(l' - l)$
and *i'-top*: $i' < (i + 1) * 2^{\wedge}(l' - l)$
shows $l2-\varphi (l', i') (l, i) = (1 - \text{real-of-int } i' / 2^{\wedge}(l' - l) + \text{real-of-int } i) / 2^{\wedge}(l' + 1)$
 <proof>

lemma *level-shift*: $lc > l \implies (x :: \text{real}) / 2^{\wedge}(lc - \text{Suc } l) = x * 2 / 2^{\wedge}(lc - l)$
 <proof>

lemma *l2-child*: **assumes** $d < \text{length } b$

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

lemma *l2-same*: $l2-\varphi (p ! d) (p ! d) = 1/3 / 2^{\wedge}(lv \text{ p } d)$
 <proof>

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** $lv \text{ p' } d \geq lv \text{ p } d$
shows $l2-\varphi (p' ! d) (p ! d) = 0$
 <proof>

lemma *l2-down2*:

fixes $pc \text{ pd } p$
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>

lemma *l2-zigzag*:

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

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*
f_m = (*f_l* + *f_r*) / 2 + (α *p*);
 $\alpha = \alpha$ (*p* := ((*f_l* + *f_r*) / 4 + (1 / 3) * (α *p*)) / 2 ^ (*lv* *p* *d*));
 $\alpha =$ *down'* *d* *l* (*child* *p* *left* *d*) *f_l* *f_m* α ;
 $\alpha =$ *down'* *d* *l* (*child* *p* *right* *d*) *f_m* *f_r* α
in α)

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

down = *lift* (λ *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*
(*f_l*, *f_m*), α) = *up'* *d* *l* (*child* *p* *left* *d*) α ;
(*f_m*, *f_r*), α) = *up'* *d* *l* (*child* *p* *right* *d*) α ;
result = (*f_m* + *f_m* + (α *p*) / 2 ^ (*lv* *p* *d*) / 2) / 2
in ((*f_l* + *result*, *f_r* + *result*), α (*p* := *f_m* + *f_m*)))

definition *up* :: *nat* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *vector* \Rightarrow *vector* **where**

up = *lift* (λ *d* *lm* *p* α . *snd* (*up'* *d* *lm* *p* α))

fun *updown'* :: *nat* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *vector* \Rightarrow *vector* **where**

updown' *dm* *lm* 0 $\alpha = \alpha$
| *updown'* *dm* *lm* (*Suc* *d*) $\alpha =$
(*sum-vector* (*updown'* *dm* *lm* *d* (*up* *dm* *lm* *d* α)) (*down* *dm* *lm* *d* (*updown'* *dm* *lm* *d* α)))

definition *updown* :: *nat* \Rightarrow *nat* \Rightarrow *vector* \Rightarrow *vector* **where**

updown *dm* *lm* $\alpha =$ *updown'* *dm* *lm* *dm* α

end

5 Up Part

theory *Up*

imports *UpDown-Scheme* *Triangular-Function*

begin

lemma *up'-inplace*:

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

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

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

$\langle \text{proof} \rangle$

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

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

$\langle \text{proof} \rangle$

lemma up :

assumes $d < dm$ **and** $p \in \text{sparsegrid } dm \text{lm}$
shows $(\text{up } dm \text{lm } d \alpha) p = (\sum p' \in (\text{lgrid } p \{d\} \text{lm}) - \{p\}. \alpha p' * \text{l2-}\varphi (p' ! d)$
 $(p ! d))$
 $\langle \text{proof} \rangle$

end

6 Down part

theory *Down*

imports *Triangular-Function UpDown-Scheme*

begin

lemma $\text{sparsegrid}'\text{-parents}$:

assumes $b: b \in \text{sparsegrid}' dm$ **and** $p': p' \in \text{parents } d b p$
shows $p' \in \text{sparsegrid}' dm$
 $\langle \text{proof} \rangle$

lemma $\text{down}'\text{-}\beta$: $\llbracket d < \text{length } b ; l + \text{level } b = \text{lm} ; b \in \text{sparsegrid}' dm ; p \in \text{sparsegrid}' dm \rrbracket \implies$

$\text{down}' d l b \text{fl fr } \alpha p = (\text{if } p \in \text{lgrid } b \{d\} \text{lm}$
 then
 $(\text{fl} + (\text{fr} - \text{fl}) / 2 * (\text{real-of-int } (ix p d) / 2 \wedge (\text{lv } p d - \text{lv } b d) - \text{real-of-int } (ix$
 $b d) + 1)) / 2 \wedge (\text{lv } p d + 1) +$
 $(\sum p' \in \text{parents } d b p. (\alpha p') * \text{l2-}\varphi (p' ! d) (p' ! d))$
 $\text{else } \alpha p)$
 $\langle \text{proof} \rangle$

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

end

7 UpDown

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

lemma *updown'*: $\llbracket d \leq dm; p \in \text{sparsegrid } dm \text{ } lm \rrbracket$
 $\implies (\text{updown}' \text{ } dm \text{ } lm \text{ } d \text{ } \alpha) p = (\sum p' \in \text{lgrid } (\text{base } \{0 \dots d\} p) \{0 \dots d\} \text{ } lm.$
 $\alpha p' * (\prod d' \in \{0 \dots d\}. l2\text{-}\varphi (p' ! d') (p ! d')))$
(is $\llbracket - ; - \rrbracket \implies - = (\sum p' \in ?\text{subgrid } d \text{ } p. \alpha p' * ?\text{prod } d \text{ } p' p)$
 $\langle \text{proof} \rangle$

theorem *updown*:

assumes *p-spg*: $p \in \text{sparsegrid } dm \text{ } lm$
shows $\text{updown } dm \text{ } lm \text{ } \alpha p = (\sum p' \in \text{sparsegrid } dm \text{ } lm. \alpha p' * l2 \text{ } p' p)$
 $\langle \text{proof} \rangle$

corollary

fixes α
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 $\text{updown } dm \text{ } lm \text{ } \alpha p = (\int x. f_\alpha x * \Phi p x \text{ } \partial(\prod_M d \in \{.. < dm\}. \text{l borel}))$
 $\langle \text{proof} \rangle$

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* $\langle \text{proof} \rangle$

primrec *rat-pair* **where** *rat-pair* $(a, b) = (\text{of-rat } a, \text{of-rat } b)$

declare *rat-pair.simps* [*simp del*]

definition

$zipWithA :: ('a::heap \Rightarrow 'b::heap \Rightarrow 'a::heap) \Rightarrow 'a \text{ array} \Rightarrow 'b \text{ array} \Rightarrow 'a \text{ array}$
Heap

where

```
zipWithA f a b = do {
  n ← Array.len a;
  Heap-Monad.fold-map (λn. do {
    x ← Array.nth a n ;
    y ← Array.nth b n ;
    Array.upd n (f x y) a
  }) [0..<n];
  return a
}
```

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

fixes $xs\ ys :: 'a::heap \text{ list}$
assumes $length\ xs = length\ ys$
shows $\langle a \mapsto_a xs * b \mapsto_a ys \rangle zipWithA\ f\ a\ b < \lambda r. (a \mapsto_a \text{map}\ (case\text{-prod}\ f)\ (zip\ xs\ ys)) * b \mapsto_a ys * \uparrow(a = r) \rangle$
<proof>

definition $copy\text{-array} :: 'a::heap \text{ array} \Rightarrow ('a::heap \text{ array})$ *Heap* **where**

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

theorem $copy\text{-array}$ [*sep-heap-rules*]:

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

definition $sum\text{-array} :: rat \text{ array} \Rightarrow rat \text{ array} \Rightarrow unit$ *Heap* **where**

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

theorem $sum\text{-array}$ [*sep-heap-rules*]:

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

locale $linearization =$

fixes $dm\ lm :: nat$

fixes $pm :: pointmap$

assumes $pm: \text{bij-betw}\ pm\ (sparsegrid\ dm\ lm)\ \{..\ < card\ (sparsegrid\ dm\ lm)\}$

begin

lemma $linearizationD$:

$p \in sparsegrid\ dm\ lm \implies pm\ p < card\ (sparsegrid\ dm\ lm)$
<proof>

definition $gridI :: impgrid \Rightarrow (grid\text{-point} \Rightarrow real) \Rightarrow assn$ **where**

$gridI\ a\ v =$
 $(\exists_A\ xs.\ a \mapsto_a\ xs * \uparrow((\forall p \in sparsegrid\ dm\ lm.\ v\ p = of-rat\ (xs\ !\ pm\ p)) \wedge length\ xs = card\ (sparsegrid\ dm\ lm)))$

lemma $gridI\text{-}nth\text{-}rule$ [$sep\text{-}heap\text{-}rules$]:

$g \in sparsegrid\ dm\ lm \implies \langle gridI\ a\ v \rangle Array.nth\ a\ (pm\ g) \langle \lambda r.\ gridI\ a\ v * \uparrow (of-rat\ r = v\ g) \rangle$
 $\langle proof \rangle$

lemma $gridI\text{-}upd\text{-}rule$ [$sep\text{-}heap\text{-}rules$]:

$g \in sparsegrid\ dm\ lm \implies$
 $\langle gridI\ a\ v \rangle Array.upd\ (pm\ g)\ x\ a \langle \lambda a' . gridI\ a\ (fun-upd\ v\ g\ (of-rat\ x)) * \uparrow (a' = a) \rangle$
 $\langle proof \rangle$

primrec $upI' :: nat \Rightarrow nat \Rightarrow grid\text{-}point \Rightarrow impgrid \Rightarrow (rat * rat)\ Heap$ **where**

$upI'\ d\ 0\ p\ a = return\ (0,\ 0) \mid$
 $upI'\ d\ (Suc\ l)\ p\ a = do\ \{$
 $\quad (fl,\ fml) \leftarrow upI'\ d\ l\ (child\ p\ left\ d)\ a ;$
 $\quad (fmr,\ fr) \leftarrow upI'\ d\ l\ (child\ p\ right\ d)\ a ;$
 $\quad val \leftarrow Array.nth\ a\ (pm\ p) ;$
 $\quad Array.upd\ (pm\ p)\ (fml + fmr)\ a ;$
 $\quad let\ result = ((fml + fmr + val / 2 \wedge (lv\ p\ d) / 2) / 2) ;$
 $\quad return\ (fl + result,\ fr + result)$
 $\}$

lemma upI' [$sep\text{-}heap\text{-}rules$]:

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

primrec $downI' :: nat \Rightarrow nat \Rightarrow grid\text{-}point \Rightarrow impgrid \Rightarrow rat \Rightarrow rat \Rightarrow unit\ Heap$ **where**

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

lemma $downI'$ [$sep\text{-}heap\text{-}rules$]:

assumes $lin[simp]$: $d < dm$
and l : $level\ p + l = lm\ l = 0 \vee p \in sparsegrid\ dm\ lm$
shows $\langle gridI\ a\ v \rangle downI'\ d\ l\ p\ a\ fl\ fr \langle \lambda r.\ gridI\ a\ (down'\ d\ l\ p\ (of-rat\ fl)\ (of-rat\ fr)\ v) \rangle$

<proof>

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

liftI f d a =
foldr (λ p n. n ≫ 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]: ∧v p. p ∈ lgrid (start dm) ({0..<dm} - {d}) lm ⇒
< gridI a v > f d (lm - level p) p a <λr. gridI a (f' d (lm - level p) p v) >

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

<proof>

definition *upI* **where** *upI* = liftI (λd l p a. upI' d l p a ≫ return ())

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

assumes [simp]: d < dm

shows < gridI a v > upI d a <λr. gridI a (up dm lm d v) >

<proof>

definition *downI* **where** *downI* = liftI (λd 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 <λr. gridI a (down dm lm d v) >

<proof>

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

< gridI a v > copy-array a <λr. gridI a v * gridI r v >

<proof>

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

< gridI a v * gridI b w > sum-array a b <λr. gridI a (sum-vector v w) * gridI b w >

<proof>

primrec *updownI'* :: nat ⇒ impgrid ⇒ unit Heap **where**

updownI' 0 a = return () |

updownI' (Suc d) a = do {

b ← 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>

definition *updownI* **where** $\text{updownI } a = \text{updownI}' \ dm \ a$

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

$\langle \text{gridI } a \ v \rangle \text{updownI } a \ \langle \lambda r. \text{gridI } a \ (\text{updown} \ dm \ lm \ v) \rangle_t$
<proof>

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.