Verification of the `UpDown` scheme

Johannes Hölzl

27. Mai 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 correctnes 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

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 ⋀*d*. *d* < *length p* ⟹ *lv p d* = *lv p′ d*
  and *length p* = *length p′*
  shows *level p′* = *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* ⇒ *dir* ⇒ *nat* ⇒ *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* ⟹ *odd* (*ix* (*child p dir d*) *d*)
⟨*proof*⟩

lemma *child-eq*: *p* ! *d* = (*l*, *i*) ⟹ ∃ *j*. *child p dir d* = *p*[*d* := (*l* + *1*, *j*)]
⟨*proof*⟩

lemma *child-other*: *d* ≠ *d*′ ⟹ *child p dir d* ! *d*′ = *p* ! *d*′
⟨*proof*⟩

lemma *child-invariant*: **assumes** *d*′ < *length p* **shows** (*child p dir d* ! *d*′ = *p* ! *d*′)
= (*d* ≠ *d*′)
⟨*proof*⟩

lemma *child-single-level*: *d* < *length p* ⟹ *lv* (*child p dir d*) *d* > *lv p d*
⟨*proof*⟩

lemma *child-lv*: *d* < *length p* ⟹ *lv* (*child p dir d*) *d* = *lv p d* + *1*
⟨*proof*⟩

lemma *child-lv-other*: **assumes** *d*′ ≠ *d* **shows** *lv* (*child p dir d*′) *d* = *lv p d*
⟨*proof*⟩

lemma *child-ix-left*: *d* < *length p* ⟹ *ix* (*child p left d*) *d* = *2* ∗ *ix p d* − *1*
⟨*proof*⟩

lemma *child-ix-right*: *d* < *length p* ⟹ *ix* (*child p right d*) *d* = *2* ∗ *ix p d* + *1*
⟨*proof*⟩

lemma *child-ix*: *d* < *length p* ⟹ *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*: ∃ *b*′. *child b dir d* = *child b*′ (*inv dir*) *d*
⟨*proof*⟩

lemma *child-level-gt*[*simp*]: *level* (*child p dir d*) >= *level p*
⟨*proof*⟩

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

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

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

lemma *start-lv*[*simp*]: $d < \mathit{dm} \Longrightarrow \mathit{lv}\ (\mathit{start}\ \mathit{dm})\ d = 0$
  ⟨*proof*⟩

lemma *start-ix*[*simp*]: $d < \mathit{dm} \Longrightarrow \mathit{ix}\ (\mathit{start}\ \mathit{dm})\ d = 1$
  ⟨*proof*⟩

lemma *start-length*[*simp*]: $\mathit{length}\ (\mathit{start}\ \mathit{dm}) = \mathit{dm}$
  ⟨*proof*⟩

lemma *level-start-0*[*simp*]: $\mathit{level}\ (\mathit{start}\ \mathit{dm}) = 0$
  ⟨*proof*⟩

end

2 Sparse Grids

theory *Grid*
imports *Grid-Point*
begin

2.1 Vectors

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

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

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

2.2 Inductive enumeration of all grid points

inductive-set
  grid :: *grid-point* ⇒ *nat set* ⇒ *grid-point set*
  for *b* :: *grid-point* **and** *ds* :: *nat set*
where
    Start[*intro*!]: *b* ∈ *grid b ds*
  | *Child*[*intro*!]: ⟦ *p* ∈ *grid b ds* ; *d* ∈ *ds* ⟧ ⟹ *child p dir d* ∈ *grid b ds*

lemma *grid-length*[*simp*]: *p'* ∈ *grid p ds* ⟹ *length p'* = *length p*
  ⟨*proof*⟩

lemma *grid-union-dims*: ⟦ *ds* ⊆ *ds'* ; *p* ∈ *grid b ds* ⟧ ⟹ *p* ∈ *grid b ds'*
  ⟨*proof*⟩

lemma *grid-transitive*: ⟦ *a* ∈ *grid b ds* ; *b* ∈ *grid c ds'* ; *ds'* ⊆ *ds''* ; *ds* ⊆ *ds''* ⟧
⟹ *a* ∈ *grid c ds''*
  ⟨*proof*⟩

lemma *grid-child*[*intro?*]: **assumes** *d* ∈ *ds* **and** *p-grid*: *p* ∈ *grid* (*child b dir d*) *ds*
  shows *p* ∈ *grid b ds*
  ⟨*proof*⟩

lemma *grid-single-level*[*simp*]: **assumes** *p* ∈ *grid b ds* **and** *d* < *length b*
  shows *lv b d* ≤ *lv p d*
  ⟨*proof*⟩

lemma *grid-child-level*:
  assumes *d* < *length b*
  and *p-grid*: *p* ∈ *grid* (*child b dir d*) *ds*
  shows *lv b d* < *lv p d*
⟨*proof*⟩

lemma *child-out*: *length p* ≤ *d* ⟹ *child p dir d* = *p*
  ⟨*proof*⟩

lemma *grid-dim-remove*:
  assumes *inset*: *p* ∈ *grid b* ({*d*} ∪ *ds*)
  and *eq*: *d* < *length b* ⟹ *p* ! *d* = *b* ! *d*
  shows *p* ∈ *grid b ds*
  ⟨*proof*⟩

lemma *gridgen-dim-restrict*:
  assumes *inset*: *p* ∈ *grid b* (*ds'* ∪ *ds*)
  and *eq*: ∀ *d* ∈ *ds'*. *d* ≥ *length b*
  shows *p* ∈ *grid b ds*
  ⟨*proof*⟩

lemma *grid-dim-remove-outer*: *grid b ds* = *grid b* {*d* ∈ *ds*. *d* < *length b*}
⟨*proof*⟩

lemma *grid-level*[*intro*]: **assumes** $p \in grid\ b\ ds$ **shows** $level\ b \leq level\ p$
⟨*proof*⟩
lemma *grid-empty-ds*[*simp*]: $grid\ b\ \{\} = \{\ b\ \}$
⟨*proof*⟩
lemma *grid-Start*: **assumes** *inset*: $p \in grid\ b\ ds$ **and** *eq*: $level\ p = level\ b$ **shows** $p = b$
⟨*proof*⟩
lemma *grid-estimate*:
assumes $d < length\ b$ **and** *p-grid*: $p \in grid\ b\ ds$
shows $ix\ p\ d < (ix\ b\ d + 1) * 2\hat{}(lv\ p\ d - lv\ b\ d) \wedge ix\ p\ d > (ix\ b\ d - 1) * 2\hat{}(lv\ p\ d - lv\ b\ d)$
⟨*proof*⟩
lemma *grid-odd*: **assumes** $d < length\ b$ **and** *p-diff*: $p\ !\ d \neq b\ !\ d$ **and** *p-grid*: $p \in grid\ b\ ds$
shows $odd\ (ix\ p\ d)$
⟨*proof*⟩
lemma *grid-invariant*: **assumes** $d < length\ b$ **and** $d \notin ds$ **and** *p-grid*: $p \in grid\ b\ ds$
shows $p\ !\ d = b\ !\ d$
⟨*proof*⟩
lemma *grid-part*: **assumes** $d < length\ b$ **and** *p-valid*: $p \in grid\ b\ \{d\}$ **and** *p'-valid*: $p' \in grid\ b\ \{d\}$
and *level*: $lv\ p'\ d \geq lv\ p\ d$
and *right*: $ix\ p'\ d \leq (ix\ p\ d + 1) * 2\hat{}(lv\ p'\ d - lv\ p\ d)$ (**is** *?right p p' d*)
and *left*: $ix\ p'\ d \geq (ix\ p\ d - 1) * 2\hat{}(lv\ p'\ d - lv\ p\ d)$ (**is** *?left p p' d*)
shows $p' \in grid\ p\ \{d\}$
⟨*proof*⟩
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* ∩ *grid ?r ds* = {})
⟨*proof*⟩

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*⟩

lemma *grid-partition*:
$grid\ p\ \{d\} = \{p\} \cup grid\ (child\ p\ left\ d)\ \{d\} \cup grid\ (child\ p\ right\ d)\ \{d\}$
(**is** - = - ∪ *grid ?l* {*d*} ∪ *grid ?r* {*d*})
⟨*proof*⟩
lemma *grid-change-dim*: **assumes** *grid*: $p \in grid\ b\ ds$
shows $p[d := X] \in grid\ (b[d := X])\ ds$
⟨*proof*⟩
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*⟩
lemma *grid-split*: **assumes** *grid*: $p \in grid\ b\ (ds' \cup ds)$ **shows** $\exists\ x \in grid\ b\ ds.\ p$

$\in$ *grid x ds'*
⟨*proof*⟩
lemma *grid-union-eq*: $(\bigcup\ p \in grid\ b\ ds.\ grid\ p\ ds') = grid\ b\ (ds' \cup ds)$
⟨*proof*⟩
lemma *grid-onedim-split*:
grid b $(ds \cup \{d\}) = grid\ b\ ds \cup grid\ (child\ b\ left\ d)\ (ds \cup \{d\}) \cup grid\ (child\ b\ right\ d)\ (ds \cup \{d\})$
(**is** - = *?g* $\cup$ *?l* $(ds \cup \{d\}) \cup$ *?r* $(ds \cup \{d\})$)
⟨*proof*⟩
lemma *grid-child-without-parent*: **assumes** *grid*: $p \in grid\ (child\ b\ dir\ d)\ ds$ (**is** $p \in grid\ ?c\ ds$) **and** $d < length\ b$
shows $p \neq b$
⟨*proof*⟩
lemma *grid-disjunct'*:
assumes $p \in grid\ b\ ds$ **and** $p' \in grid\ b\ ds$ **and** $x \in grid\ p\ ds'$ **and** $p \neq p'$ **and** $ds \cap ds' = \{\}$
shows $x \notin grid\ p'\ ds'$
⟨*proof*⟩
lemma *grid-split1*: **assumes** *grid*: $p \in grid\ b\ (ds' \cup ds)$ **and** $ds \cap ds' = \{\}$
shows $\exists!\ x \in grid\ b\ ds.\ p \in grid\ x\ ds'$
⟨*proof*⟩

2.3 Grid Restricted to a Level

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

lemma *lgridI*[*intro*]:
$[\![\ p \in grid\ b\ ds\ ;\ level\ p < lm\ ]\!] \Longrightarrow p \in lgrid\ b\ ds\ lm$
⟨*proof*⟩

lemma *lgridE*[*elim*]:
assumes $p \in lgrid\ b\ ds\ lm$
assumes $[\![\ p \in grid\ b\ ds\ ;\ level\ p < lm\ ]\!] \Longrightarrow P$
shows *P*
⟨*proof*⟩

lemma *lgridI-child*[*intro*]:
$d \in ds \Longrightarrow p \in lgrid\ (child\ b\ dir\ d)\ ds\ lm \Longrightarrow p \in lgrid\ b\ ds\ lm$
⟨*proof*⟩

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

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

lemma *grid-not-child*:
assumes [*simp*]: $d < length\ p$

shows $p \notin grid\ (child\ p\ dir\ d)\ ds$
⟨*proof*⟩

2.4 Unbounded Sparse Grid

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

lemma *grid-subset-alldim*:
assumes *p*: $p \in grid\ b\ ds$
defines $dm \equiv length\ b$
shows $p \in grid\ b\ \{0..<dm\}$
⟨*proof*⟩

lemma *sparsegrid′-length*[*simp*]:
$b \in sparsegrid'\ dm \Longrightarrow length\ b = dm$ ⟨*proof*⟩

lemma *sparsegrid′I*[*intro*]:
assumes *b*: $b \in sparsegrid'\ dm$ **and** *p*: $p \in grid\ b\ ds$
shows $p \in sparsegrid'\ dm$
⟨*proof*⟩

lemma *sparsegrid′-start*:
assumes $b \in grid\ (start\ dm)\ ds$
shows $b \in sparsegrid'\ dm$
⟨*proof*⟩

2.5 Sparse Grid

definition *sparsegrid* :: *nat* ⇒ *nat* ⇒ *grid-point set*
where
sparsegrid dm lm = *lgrid* (*start dm*) { *0* ..< *dm* } *lm*

lemma *sparsegrid-length*: $p \in sparsegrid\ dm\ lm \Longrightarrow length\ p = dm$
⟨*proof*⟩

lemma *sparsegrid-subset*[*intro*]: $p \in sparsegrid\ dm\ lm \Longrightarrow p \in sparsegrid'\ dm$
⟨*proof*⟩

lemma *sparsegridI*[*intro*]:
assumes $p \in sparsegrid'\ dm$ **and** $level\ p < lm$
shows $p \in sparsegrid\ dm\ lm$
⟨*proof*⟩

lemma *sparsegrid-start*:
assumes $b \in lgrid\ (start\ dm)\ ds\ lm$
shows $b \in sparsegrid\ dm\ lm$
⟨*proof*⟩

lemma *sparsegridE*[*elim*]:
assumes $p \in sparsegrid\ dm\ lm$
shows $p \in sparsegrid'\ dm$ **and** $level\ p < lm$
⟨*proof*⟩

2.6 Compute Sparse Grid Points

fun *gridgen* :: *grid-point* ⇒ *nat set* ⇒ *nat* ⇒ *grid-point list*
where
gridgen p ds 0 = []
| *gridgen p ds* (*Suc l*) = (*let*
sub = λ *d*. *gridgen* (*child p left d*) { $d' \in ds\ .\ d' \leq d$ } *l* @
gridgen (*child p right d*) { $d' \in ds\ .\ d' \leq d$ } *l*
in p # *concat* (*map sub* [$d \leftarrow [0\ ..<\ length\ p].\ d \in ds$]))

lemma *gridgen-lgrid-eq*: *set* (*gridgen p ds l*) = *lgrid p ds* (*level p* + *l*)
⟨*proof*⟩

lemma *gridgen-distinct*: *distinct* (*gridgen p ds l*)
⟨*proof*⟩

lemma *lgrid-finite*: *finite* (*lgrid b ds lm*)
⟨*proof*⟩

lemma *lgrid-sum*:
fixes *F* :: *grid-point* ⇒ *real*
assumes *d* < *length b* **and** *level b* < *lm*
shows ($\sum$ *p* ∈ *lgrid b* {*d*} *lm*. *F p*) =
($\sum$ *p* ∈ *lgrid* (*child b left d*) {*d*} *lm*. *F p*) + ($\sum$ *p* ∈ *lgrid* (*child b right d*) {*d*} *lm*. *F p*) + *F b*
(**is** ($\sum$ *p* ∈ *?grid b*. *F p*) = ($\sum$ *p* ∈ *?grid ?l* . *F p*) + (*?sum* (*?grid ?r*)) + *F b*)
⟨*proof*⟩

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-grid*: *p* ∈ *sparsegrid' dm*
shows *base ds p* ∈ *grid* (*start dm*) ({*0*..<*dm*} − *ds*)
and *p* ∈ *grid* (*base ds p*) *ds*
⟨*proof*⟩

lemma *baseI*: **assumes** *x-grid*: *x* ∈ *grid* (*start dm*) ({*0*..<*dm*} − *ds*) **and** *p-xgrid*: *p* ∈ *grid x ds*
shows *base ds p* = *x*
⟨*proof*⟩

lemma *base-empty*: **assumes** *p-grid*: *p* ∈ *sparsegrid' dm* **shows** *base* {} *p* = *p*

⟨*proof*⟩

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

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

lemma *base-grid*: **assumes** *p-grid*: $p \in sparsegrid'\ dm$ **shows** $grid\ (base\ ds\ p)\ ds \subseteq sparsegrid'\ dm$
⟨*proof*⟩
lemma *base-length*[*simp*]: **assumes** *p-grid*: $p \in sparsegrid'\ dm$ **shows** $length\ (base\ ds\ p) = dm$
⟨*proof*⟩
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*⟩
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*⟩
lemma *base-base*: **assumes** *p-grid*: $p \in sparsegrid'\ dm$ **shows** $base\ ds\ (base\ ds'\ p) = base\ (ds \cup ds')\ p$
⟨*proof*⟩
lemma *grid-base-out*: **assumes** $d < dm$ **and** $d \notin ds$ **and** *p-grid*: $b \in sparsegrid'\ dm$ **and** $p \in grid\ (base\ ds\ b)\ ds$
shows $p\ !\ d = b\ !\ d$
⟨*proof*⟩

lemma *grid-grid-inj-on*: **assumes** $ds \cap ds' = \{\}$ **shows** $inj\text{-}on\ snd\ (\bigcup p' \in grid\ b\ ds.\ \bigcup p'' \in grid\ p'\ ds'.\ \{(p', p'')\})$
⟨*proof*⟩

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

lemma *grid-base-base*: **assumes** $b \in sparsegrid'\ dm$
shows $base\ ds'\ b \in grid\ (base\ ds\ (base\ ds'\ b))\ (ds \cup ds')$
⟨*proof*⟩

lemma *grid-base-union*: **assumes** *b-spg*: $b \in sparsegrid'\ dm$ **and** *p-grid*: $p \in grid\ (base\ ds\ b)\ ds$ **and** *x-grid*: $x \in grid\ (base\ ds'\ p)\ ds'$
shows $x \in grid\ (base\ (ds \cup ds')\ b)\ (ds \cup ds')$
⟨*proof*⟩
lemma *grid-base-dim-add*: **assumes** $ds' \subseteq ds$ **and** *b-spg*: $b \in sparsegrid'\ dm$ **and** *p-grid*: $p \in grid\ (base\ ds'\ b)\ ds'$
shows $p \in grid\ (base\ ds\ b)\ ds$

⟨*proof*⟩
lemma *grid-replace-dim*: **assumes** $d < \mathit{length}\ b'$ **and** $d < \mathit{length}\ b$ **and** *p-grid*: $p \in \mathit{grid}\ b\ \mathit{ds}$ **and** *p'-grid*: $p' \in \mathit{grid}\ b'\ \mathit{ds}$
shows $p[d := p'\ !\ d] \in \mathit{grid}\ (b[d := b'\ !\ d])\ \mathit{ds}$ (**is** - ∈ *grid ?b ds*)
⟨*proof*⟩
lemma *grid-shift-base*:
assumes *ds-dj*: $\mathit{ds} \cap \mathit{ds}' = \{\}$ **and** *b-spg*: $b \in \mathit{sparsegrid}'\ \mathit{dm}$ **and** *p-grid*: $p \in \mathit{grid}\ (\mathit{base}\ (\mathit{ds}' \cup \mathit{ds})\ b)\ (\mathit{ds}' \cup \mathit{ds})$
shows $\mathit{base}\ \mathit{ds}'\ p \in \mathit{grid}\ (\mathit{base}\ (\mathit{ds} \cup \mathit{ds}')\ b)\ \mathit{ds}$
⟨*proof*⟩

2.8 Lift Operation over all Grid Points

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

lemma *lift*:
assumes $d < \mathit{dm}$ **and** $p \in \mathit{sparsegrid}\ \mathit{dm}\ \mathit{lm}$
and *Fintro*: $\bigwedge\ l\ b\ p\ \alpha.\ [\![\ b \in \mathit{lgrid}\ (\mathit{start}\ \mathit{dm})\ (\{0..{<}\mathit{dm}\} - \{d\})\ \mathit{lm}\ ;$
$l + \mathit{level}\ b = \mathit{lm}\ ;\ p \in \mathit{sparsegrid}\ \mathit{dm}\ \mathit{lm}\ ]\!]$
$\Longrightarrow F\ d\ l\ b\ \alpha\ p = (\mathit{if}\ b = \mathit{base}\ \{d\}\ p$
$\mathit{then}\ (\sum\ p' \in \mathit{lgrid}\ b\ \{d\}\ \mathit{lm}.\ S\ (\alpha\ p')\ p\ p')$
$\mathit{else}\ \alpha\ p)$
shows $\mathit{lift}\ F\ \mathit{dm}\ \mathit{lm}\ d\ \alpha\ p = (\sum\ p' \in \mathit{lgrid}\ (\mathit{base}\ \{d\}\ p)\ \{d\}\ \mathit{lm}.\ S\ (\alpha\ p')\ p\ p')$
(**is** *?lift* = *?S p* α)
⟨*proof*⟩

2.9 Parent Points

definition *parents* :: $\mathit{nat} \Rightarrow \mathit{grid\text{-}point} \Rightarrow \mathit{grid\text{-}point} \Rightarrow \mathit{grid\text{-}point}\ \mathit{set}$
where $\mathit{parents}\ d\ b\ p = \{\ x \in \mathit{grid}\ b\ \{d\}.\ p \in \mathit{grid}\ x\ \{d\}\ \}$

lemma *parents-split*: **assumes** *p-grid*: $p \in \mathit{grid}\ (\mathit{child}\ b\ \mathit{dir}\ d)\ \{d\}$
shows $\mathit{parents}\ d\ b\ p = \{\ b\ \} \cup \mathit{parents}\ d\ (\mathit{child}\ b\ \mathit{dir}\ d)\ p$
⟨*proof*⟩

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

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

lemma *parents-finite*: *finite* $(\mathit{parents}\ d\ b\ p)$
⟨*proof*⟩

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

shows $(\sum\ x \in parents\ d\ b\ p.\ F\ x) = F\ b + (\sum\ x \in parents\ d\ (child\ b\ dir\ d)\ p.\ F\ x)$
⟨*proof*⟩

lemma *parents-single*: $parents\ d\ b\ b = \{\ b\ \}$
⟨*proof*⟩

lemma *grid-single-dimensional-specification*:
assumes $d < length\ b$
and $odd\ i$
and $lv\ b\ d + l' = l$
and $i < (ix\ b\ d + 1) * 2\hat{}l'$
and $i > (ix\ b\ d - 1) * 2\hat{}l'$
shows $b[d := (l,i)] \in grid\ b\ \{d\}$
⟨*proof*⟩

lemma *grid-multi-dimensional-specification*:
assumes $dm \leq length\ b$ **and** $length\ p = length\ b$
and $\bigwedge\ d.\ d < dm \Longrightarrow$
$odd\ (ix\ p\ d) \wedge$
$lv\ b\ d \leq lv\ p\ d \wedge$
$ix\ p\ d < (ix\ b\ d + 1) * 2\hat{}(lv\ p\ d - lv\ b\ d) \wedge$
$ix\ p\ d > (ix\ b\ d - 1) * 2\hat{}(lv\ p\ d - lv\ b\ d)$
(**is** $\bigwedge\ d.\ d < dm \Longrightarrow ?bounded\ p\ d$)
and $\bigwedge\ d.\ [\![\ dm \leq d\ ;\ d < length\ b\ ]\!] \Longrightarrow p\ !\ d = b\ !\ d$
shows $p \in grid\ b\ \{0..<dm\}$
⟨*proof*⟩

lemma *sparsegrid*:
$sparsegrid\ dm\ lm = \{p.$
$length\ p = dm \wedge level\ p < lm \wedge$
$(\forall\ d < dm.\ odd\ (ix\ p\ d) \wedge 0 < ix\ p\ d \wedge ix\ p\ d < 2\hat{}(lv\ p\ d + 1))\}$
(**is** $- = ?set$)
⟨*proof*⟩

end

3 Hat Functions

theory *Triangular-Function*
imports
HOL−Analysis.Equivalence-Lebesgue-Henstock-Integration
Grid
begin

lemma *continuous-on-max*[*continuous-intros*]:
fixes $f :: - \Rightarrow 'a::linorder\text{-}topology$
shows $continuous\text{-}on\ S\ f \Longrightarrow continuous\text{-}on\ S\ g \Longrightarrow continuous\text{-}on\ S\ (\lambda x.\ max\ (f\ x)\ (g\ x))$

⟨*proof*⟩

definition $\varphi :: (nat \times int) \Rightarrow real \Rightarrow real$ **where**
$\varphi \equiv (\lambda(l,i)\ x.\ max\ 0\ (1 - |\ x * 2\hat{}(l + 1) - real\text{-}of\text{-}int\ i\ |))$

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-φ p1 p2 = $(\int x.\ \varphi\ p1\ x * \varphi\ p2\ x\ \partial lborel)$

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

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

lemma *φ-nonneg*: $0 \le \varphi\ p\ x$
⟨*proof*⟩

lemma *φ-zero-iff*:
$\varphi\ (l,i)\ x = 0 \longleftrightarrow x \notin \{real\text{-}of\text{-}int\ (i - 1)\ /\ 2\hat{}(l + 1) <..< real\text{-}of\text{-}int\ (i + 1)\ /\ 2\hat{}(l + 1)\}$
⟨*proof*⟩

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

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

lemma *ix-lt*: $p \in sparsegrid\ dm\ lm \Longrightarrow d < dm \Longrightarrow ix\ p\ d < 2\hat{}(lv\ p\ d + 1)$
⟨*proof*⟩

lemma *ix-gt*: $p \in sparsegrid\ dm\ lm \Longrightarrow d < dm \Longrightarrow 0 < ix\ p\ d$
⟨*proof*⟩

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

lemma *φ-left-support′*:
$x \in \{real\text{-}of\text{-}int\ (i - 1)\ /\ 2\hat{}(l + 1)\ ..\ real\text{-}of\text{-}int\ i\ /\ 2\hat{}(l + 1)\} \Longrightarrow \varphi\ (l,i)\ x = 1 + x * 2\hat{}(l + 1) - real\text{-}of\text{-}int\ i$
⟨*proof*⟩

lemma *φ-left-support*: $x \in \{-1\ ..\ 0{::}real\} \Longrightarrow \varphi\ (l,i)\ ((x + real\text{-}of\text{-}int\ i)\ /\ 2\hat{}(l$

$+ 1)) = 1 + x$
⟨*proof*⟩

lemma *φ-right-support'*:
$x \in \{real\text{-}of\text{-}int\ i\ /\ 2\hat{}(l + 1)\ ..\ real\text{-}of\text{-}int\ (i + 1)\ /\ 2\hat{}(l + 1)\} \Longrightarrow \varphi\ (l,i)\ x = 1 - x * 2\hat{}(l + 1) + real\text{-}of\text{-}int\ i$
⟨*proof*⟩

lemma *φ-right-support*:
$x \in \{0\ ..\ 1::real\} \Longrightarrow \varphi\ (l,i)\ ((x + real\ i)\ /\ 2\hat{}(l + 1)) = 1 - x$
⟨*proof*⟩

lemma *integrable-φ*: *integrable lborel* $(\varphi\ p)$
⟨*proof*⟩

lemma *integrable-φ2*: *integrable lborel* $(\lambda x.\ \varphi\ p\ x * \varphi\ q\ x)$
⟨*proof*⟩

lemma *l2-φI-DERIV*:
assumes *n*: $\bigwedge x.\ x \in \{\ (real\text{-}of\text{-}int\ i' - 1)\ /\ 2\hat{}(l' + 1)\ ..\ real\text{-}of\text{-}int\ i'\ /\ 2\hat{}(l' + 1)\ \} \Longrightarrow$
DERIV Φ-*n x* :> $(\varphi\ (l',\ i')\ x * \varphi\ (l,\ i)\ x)$ (**is** $\bigwedge x.\ x \in \{?a..?b\} \Longrightarrow$ *DERIV* - - :> *?P x*)
and *p*: $\bigwedge x.\ x \in \{\ real\text{-}of\text{-}int\ i'\ /\ 2\hat{}(l' + 1)\ ..\ (real\text{-}of\text{-}int\ i' + 1)\ /\ 2\hat{}(l' + 1)\ \} \Longrightarrow$
DERIV Φ-*p x* :> $(\varphi\ (l',\ i')\ x * \varphi\ (l,\ i)\ x)$ (**is** $\bigwedge x.\ x \in \{?b..?c\} \Longrightarrow$ -)
shows *l2-φ* $(l',\ i')\ (l,\ i) =$ (Φ-*n ?b* − Φ-*n ?a*) + (Φ-*p ?c* − Φ-*p ?b*)
⟨*proof*⟩

lemma *l2-eq*: *length a* = *length b* ⟹ *l2 a b* = $(\prod d<length\ a.\ l2\text{-}\varphi\ (a!d)\ (b!d))$
⟨*proof*⟩

lemma *l2-when-disjoint*:
assumes $l \le l'$
defines $d == l' - l$
assumes $(i + 1) * 2\hat{}d < i' \vee i' < (i - 1) * 2\hat{}d$ (**is** *?right* ∨ *?left*)
shows *l2-φ* $(l',\ i')\ (l,\ i) = 0$
⟨*proof*⟩

lemma *l2-commutative*: *l2-φ p q* = *l2-φ q p*
⟨*proof*⟩

lemma *l2-when-same*: *l2-φ* $(l,\ i)\ (l,\ i) = 1/3\ /\ 2\hat{}l$
⟨*proof*⟩

lemma *l2-when-left-child*:
assumes $l < l'$
and *i'-bot*: $i' > (i - 1) * 2\hat{}(l' - l)$
and *i'-top*: $i' < i * 2\hat{}(l' - l)$

shows *l2-φ (l′, i′) (l, i) = (1 + real-of-int i′ / 2^(l′ − l) − real-of-int i) / 2^(l′ + 1)*
⟨*proof*⟩

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*⟩

lemma *level-shift*: $lc > l \Longrightarrow (x :: real) / 2 \hat{} (lc - Suc\ l) = x * 2 / 2 \hat{} (lc - l)$
⟨*proof*⟩

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

lemma *l2-same*: *l2-φ (p!d) (p!d) = 1/3 / 2^(lv p d)*
⟨*proof*⟩

lemma *l2-disjoint*: **assumes** $d <$ *length b* **and** $p \in$ *grid b {d}* **and** $p' \in$ *grid b {d}*
and $p' \notin$ *grid p {d}* **and** $lv\ p'\ d \geq lv\ p\ d$
shows *l2-φ (p′ ! d) (p ! d) = 0*
⟨*proof*⟩

lemma *l2-down2*:
fixes *pc pd p*
assumes $d <$ *length pd*
assumes *pc-in-grid*: $pc \in$ *grid (child pd dir d) {d}*
assumes *pd-is-child*: *pd = child p dir d* (**is** *pd = ?pd*)
shows *l2-φ (pc ! d) (pd ! d) / 2 = l2-φ (pc ! d) (p ! d)*
⟨*proof*⟩

lemma *l2-zigzag*:
assumes $d <$ *length p* **and** *p-child*: *p = child p-p dir d*
and *p′-grid*: $p' \in$ *grid (child p (inv dir) d) {d}*
and *ps-intro*: *child p (inv dir) d = child ps dir d* (**is** *?c-p = ?c-ps*)
shows *l2-φ (p′ ! d) (p-p ! d) = l2-φ (p′ ! d) (ps ! d) + l2-φ (p′ ! d) (p ! d) / 2*
⟨*proof*⟩

end

4 UpDown Scheme

theory *UpDown-Scheme*
imports *Grid*
begin

fun *down′* :: *nat* ⇒ *nat* ⇒ *grid-point* ⇒ *real* ⇒ *real* ⇒ *vector* ⇒ *vector*
where
down′ d 0 p f_l f_r α = α
| *down′ d (Suc l) p f_l f_r α = (let*
f_m = (f_l + f_r) / 2 + (α p);
α = α(p := ((f_l + f_r) / 4 + (1 / 3) ∗ (α p)) / 2 ^ (lv p d));
α = down′ d l (child p left d) f_l f_m α;
α = down′ d l (child p right d) f_m f_r α
in α)

definition *down* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *vector* ⇒ *vector* **where**
down = lift (λ d l p. down′ d l p 0 0)

fun *up′* :: *nat* ⇒ *nat* ⇒ *grid-point* ⇒ *vector* ⇒ (*real* ∗ *real*) ∗ *vector* **where**
up′ d 0 p α = ((0, 0), α)
| *up′ d (Suc l) p α = (let*
((f_l, f_ml), α) = up′ d l (child p left d) α;
((f_mr, f_r), α) = up′ d l (child p right d) α;
result = (f_ml + f_mr + (α p) / 2 ^ (lv p d) / 2) / 2
in ((f_l + result, f_r + result), α(p := f_ml + f_mr)))

definition *up* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *vector* ⇒ *vector* **where**
up = lift (λ d lm p α. snd (up′ d lm p α))

fun *updown′* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *vector* ⇒ *vector* **where**
updown′ dm lm 0 α = α
| *updown′ dm lm (Suc d) α =*
(sum-vector (updown′ dm lm d (up dm lm d α)) (down dm lm d (updown′ dm lm d α)))

definition *updown* :: *nat* ⇒ *nat* ⇒ *vector* ⇒ *vector* **where**
updown dm lm α = updown′ dm lm dm α

end

5 Up Part

theory *Up*
imports *UpDown-Scheme Triangular-Function*
begin

lemma *up′-inplace*:

assumes *p'-in*: $p' \notin grid\ p\ ds$ **and** $d \in ds$
shows $snd\ (up'\ d\ l\ p\ \alpha)\ p' = \alpha\ p'$
⟨*proof*⟩

lemma *up'-fl-fr*:
$[\![\ d < length\ p\ ;\ p = (child\ p\text{-}r\ right\ d)\ ;\ p = (child\ p\text{-}l\ left\ d)\ ]\!]$
$\Longrightarrow fst\ (up'\ d\ lm\ p\ \alpha) =$
$(\sum\ p' \in lgrid\ p\ \{d\}\ (lm + level\ p).\ (\alpha\ p') * l2\text{-}\varphi\ (p'\ !\ d)\ (p\text{-}r\ !\ d),$
$\sum\ p' \in lgrid\ p\ \{d\}\ (lm + level\ p).\ (\alpha\ p') * l2\text{-}\varphi\ (p'\ !\ d)\ (p\text{-}l\ !\ d))$
⟨*proof*⟩

lemma *up'-β*:
$[\![\ d < length\ b\ ;\ l + level\ b = lm\ ;\ b \in sparsegrid'\ dm\ ;\ p \in sparsegrid'\ dm\ ]\!]$
$\Longrightarrow$
$(snd\ (up'\ d\ l\ b\ \alpha))\ p =$
$(if\ p \in lgrid\ b\ \{d\}\ lm$
$then \sum\ p' \in (lgrid\ p\ \{d\}\ lm) - \{p\}.\ \alpha\ p' * l2\text{-}\varphi\ (p'\ !\ d)\ (p\ !\ d)$
$else\ \alpha\ p)$
(**is** $[\![\ \text{-}\ ;\ \text{-}\ ;\ \text{-}\ ;\ \text{-}\ ]\!] \Longrightarrow (?goal\ l\ b\ p\ \alpha)$)
⟨*proof*⟩

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

end

6 Down part

theory *Down*
imports *Triangular-Function UpDown-Scheme*
begin

lemma *sparsegrid'-parents*:
assumes *b*: $b \in sparsegrid'\ dm$ **and** *p'*: $p' \in parents\ d\ b\ p$
shows $p' \in sparsegrid'\ dm$
⟨*proof*⟩

lemma *down'-β*: $[\![\ d < length\ b\ ;\ l + level\ b = lm\ ;\ b \in sparsegrid'\ dm\ ;\ p \in sparsegrid'\ dm\ ]\!] \Longrightarrow$
$down'\ d\ l\ b\ fl\ fr\ \alpha\ p = (if\ p \in lgrid\ b\ \{d\}\ lm$
$then$
$(fl + (fr - fl)\ /\ 2 * (real\text{-}of\text{-}int\ (ix\ p\ d)\ /\ 2\hat{}(lv\ p\ d - lv\ b\ d) - real\text{-}of\text{-}int\ (ix\ b\ d) + 1))\ /\ 2\ \hat{}\ (lv\ p\ d + 1) +$
$(\sum\ p' \in parents\ d\ b\ p.\ (\alpha\ p') * l2\text{-}\varphi\ (p\ !\ d)\ (p'\ !\ d))$
$else\ \alpha\ p)$
⟨*proof*⟩

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

end

7 UpDown

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

lemma *updown'*: $[\![\ d \leq dm;\ p \in sparsegrid\ dm\ lm\ ]\!]$
$\Longrightarrow (updown'\ dm\ lm\ d\ \alpha)\ p = (\sum\ p' \in lgrid\ (base\ \{0\ ..<\ d\}\ p)\ \{0\ ..<\ d\}\ lm.\ \alpha\ p' * (\prod\ d' \in \{0\ ..<\ d\}.\ l2\text{-}\varphi\ (p'\ !\ d')\ (p\ !\ d')))$
(**is** $[\![\ \text{-}\ ;\ \text{-}\ ]\!] \Longrightarrow \text{-} = (\sum\ p' \in ?subgrid\ d\ p.\ \alpha\ p' * ?prod\ d\ p'\ p)$)
⟨*proof*⟩

theorem *updown*:
assumes *p-spg*: $p \in sparsegrid\ dm\ lm$
shows $updown\ dm\ lm\ \alpha\ p = (\sum\ p' \in sparsegrid\ dm\ lm.\ \alpha\ p' * l2\ p'\ p)$
⟨*proof*⟩

corollary
fixes α
assumes p: $p \in sparsegrid\ dm\ lm$
defines $f_\alpha \equiv \lambda x.\ (\sum p \in sparsegrid\ dm\ lm.\ \alpha\ p * \Phi\ p\ x)$
shows $updown\ dm\ lm\ \alpha\ p = (\int x.\ f_\alpha\ x * \Phi\ p\ x\ \partial(\Pi_M\ d \in \{..<dm\}.\ lborel))$
⟨*proof*⟩

end

8 Imperative Version

theory *Imperative*
imports *UpDown-Scheme Separation-Logic-Imperative-HOL.Sep-Main*
begin

type-synonym *pointmap* = *grid-point* ⇒ *nat*
type-synonym *impgrid* = *rat array*

instance *rat* :: *heap* ⟨*proof*⟩

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

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

definition
zipWithA :: (*'a*::*heap* ⇒ *'b*::*heap* ⇒ *'a*::*heap*) ⇒ *'a array* ⇒ *'b array* ⇒ *'a array Heap*
where
zipWithA f a b = *do* {
n ← *Array.len a*;
Heap-Monad.fold-map (λ*n*. *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 list*
assumes *length xs* = *length ys*
shows < *a* ↦$_a$ *xs* ∗ *b* ↦$_a$ *ys* > *zipWithA f a b* < λ*r*. (*a* ↦$_a$ *map* (*case-prod f*) (*zip xs ys*)) ∗ *b* ↦$_a$ *ys* ∗ ↑(*a* = *r*) >
⟨*proof*⟩

definition *copy-array* :: *'a*::*heap array* ⇒ (*'a*::*heap array*) *Heap* **where**
copy-array a = *Array.freeze a* ⤜ *Array.of-list*

theorem *copy-array* [*sep-heap-rules*]:
< *a* ↦$_a$ *xs* > *copy-array a* < λ*r*. *a* ↦$_a$ *xs* ∗ *r* ↦$_a$ *xs* >
⟨*proof*⟩

definition *sum-array* :: *rat array* ⇒ *rat array* ⇒ *unit Heap* **where**
sum-array a b = *zipWithA* (+) *a b* ⤜ *return* ()

theorem *sum-array* [*sep-heap-rules*]:
fixes *xs ys* :: *rat list*
shows *length xs* = *length ys* ⟹ < *a* ↦$_a$ *xs* ∗ *b* ↦$_a$ *ys* > *sum-array a b* < λ*r*. (*a* ↦$_a$ *map* (λ(*a*, *b*). *a* + *b*) (*zip xs ys*)) ∗ *b* ↦$_a$ *ys* >
⟨*proof*⟩

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

lemma *linearizationD*:
p ∈ *sparsegrid dm lm* ⟹ *pm p* < *card* (*sparsegrid dm lm*)
⟨*proof*⟩

definition *gridI* :: *impgrid* ⇒ (*grid-point* ⇒ *real*) ⇒ *assn* **where**

gridI a v =
(∃ A xs. a ↦a xs ∗ ↑((∀ p∈sparsegrid dm lm. v p = of-rat (xs ! pm p)) ∧ length xs = card (sparsegrid dm lm)))

lemma *gridI-nth-rule* [*sep-heap-rules*]:
g ∈ sparsegrid dm lm ⟹ < gridI a v > Array.nth a (pm g) <λr. gridI a v ∗ ↑ (of-rat r = v g)>
⟨*proof*⟩

lemma *gridI-upd-rule* [*sep-heap-rules*]:
g ∈ sparsegrid dm lm ⟹
< gridI a v > Array.upd (pm g) x a <λa′. gridI a (fun-upd v g (of-rat x)) ∗ ↑(a′ = a)>
⟨*proof*⟩

primrec *upI′* :: *nat ⇒ nat ⇒ grid-point ⇒ impgrid ⇒ (rat ∗ rat) Heap* **where**
upI′ d 0 p a = return (0, 0) |
upI′ d (Suc l) p a = do {
(fl, fml) ← upI′ d l (child p left d) a ;
(fmr, fr) ← upI′ d l (child p right d) a ;
val ← Array.nth a (pm p) ;
Array.upd (pm p) (fml + fmr) a ;
let result = ((fml + fmr + val / 2 ^ (lv p d)) / 2) / 2) ;
return (fl + result, fr + result)
}

lemma *upI′* [*sep-heap-rules*]:
assumes *lin*[*simp*]: *d < dm*
and *l*: *level p + l = lm l = 0 ∨ p ∈ sparsegrid dm lm*
shows *< gridI a v > upI′ d l p a <λr. let (r′, v′) = up′ d l p v in gridI a v′ ∗ ↑(rat-pair r = r′) >*
⟨*proof*⟩

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) >*

⟨*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* <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*⟩

definition *upI* **where** *upI* = *liftI* (λ*d 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* <λ*r*. *gridI a* (*up dm lm d v*) >
⟨*proof*⟩

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*) >
⟨*proof*⟩

theorem *copy-array-gridI* [*sep-heap-rules*]:
< *gridI a v* > *copy-array a* < λ*r*. *gridI a v* ∗ *gridI r v* >
⟨*proof*⟩

theorem *sum-array-gridI* [*sep-heap-rules*]:
< *gridI a v* ∗ *gridI b w* > *sum-array a b* < λ*r*. *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*]:
$d \leq dm \Longrightarrow < gridI\ a\ v > updownI'\ d\ a\ <\lambda r.\ gridI\ a\ (updown'\ dm\ lm\ d\ v) >_t$
⟨*proof*⟩

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

theorem *updownI* [*sep-heap-rules*]:
$< gridI\ a\ v > updownI\ a\ <\lambda r.\ gridI\ a\ (updown\ dm\ lm\ v) >_t$
⟨*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.