Andrew's Monotone Chain Convex Hull Algorithm

Arthur Freitas Ramos
David Barros Hulak
Ruy J. G. B. de Queiroz

June 25, 2026

Abstract

This development formalizes the executable core of Andrew's monotone chain convex hull algorithm [1]. The algorithm sorts planar points lexicographically, removes duplicates, computes lower and upper hull chains by a stack scan that removes non-left turns, and concatenates the two chains with their repeated endpoints removed. The formalization proves the stack-scan turn invariant, subset and distinctness properties for the computed chains, ordered-chain facts for the lower and upper scans, a distinctness criterion for the final concatenation, length bounds, support-function invariants for the lower and upper scans, and the real-coordinate theorem that the computed output has the same convex hull as the input. The final correctness theorem states the specification explicitly: the returned vertices are input points, every input point lies in the convex hull of the returned vertices, and the returned vertices have exactly the same convex hull as the input. It also proves irredundancy of the returned vertex set: deleting any returned point changes the convex hull of the returned set. The executable algorithm is polymorphic over ordered integral domains because the scan only uses lexicographic ordering and signs of orientation determinants; the geometric theorem is stated for real coordinates because the convexity and separating-hyperplane libraries use real vector spaces. Thus the integer examples exercise the executable code, while the real examples instantiate the geometric correctness theorem. The support-function argument shows that points removed by a scan are dominated in the relevant support directions; a separating hyperplane theorem then yields convex-hull coverage, and strict supporting directions expose each returned vertex for the irredundancy result. The development proves functional correctness, but does not formalize the asymptotic running time. AI assistance was used for proof engineering. The final definitions, statements, and proofs are checked by Isabelle.

Contents

1 Andrew's Monotone Chain Convex Hull Algorithm

theory *Andrew-Monotone-Chain*
imports
HOL−Analysis.Convex
HOL−Analysis.Convex-Euclidean-Space
HOL−Library.Product-Lexorder
HOL−Library.Sublist
begin

This theory formalizes the executable core of Andrew's monotone chain convex hull algorithm. Points are sorted lexicographically, duplicate points are removed, and two stack scans compute the lower and upper chains. The final hull is the usual concatenation of the two chains with their repeated endpoints removed.

The scan is stored internally with the top of the stack at the head of the list. Reversing the stack gives the geometric left-to-right order of a lower scan, or the right-to-left order of an upper scan.

The main correctness theorem combines executable invariants with a real-coordinate support-function argument. Since the algorithm only returns input points, convex-hull equality follows once every input point is shown to lie in the convex hull of the computed output.

The executable definitions are deliberately polymorphic over ordered integral domains: the scan only needs lexicographic ordering and signs of orientation determinants. The geometric specification below is stated for real coordinates, because Isabelle's convex-hull and separating-hyperplane theorems live in real vector spaces.

type-synonym *'a point* = *'a* × *'a*

definition *cross* ::
(*'a*::*linordered-idom*) *point* ⇒ *'a point* ⇒ *'a point* ⇒ *'a*
where
cross p q r =
(*fst q* − *fst p*) ∗ (*snd r* − *snd p*) −
(*snd q* − *snd p*) ∗ (*fst r* − *fst p*)

definition *left-turn* ::

(*'a*::*linordered-idom*) *point* ⇒ *'a point* ⇒ *'a point* ⇒ *bool*
where
left-turn p q r ⟷ *cross p q r* > *0*

definition *collinear* ::
(*'a*::*linordered-idom*) *point* ⇒ *'a point* ⇒ *'a point* ⇒ *bool*
where
collinear p q r ⟷ *cross p q r* = *0*

lemma *cross-same-left* [*simp*]: *cross p p q* = *0*
⟨*proof*⟩

lemma *cross-same-right* [*simp*]: *cross p q q* = *0*
⟨*proof*⟩

lemma *cross-same-outer* [*simp*]: *cross p q p* = *0*
⟨*proof*⟩

lemma *cross-swap-outer*:
cross p q r = − *cross r q p*
⟨*proof*⟩

lemma *cross-swap-last*:
cross p r q = − *cross p q r*
⟨*proof*⟩

lemma *cross-cycle*:
cross p q r = *cross q r p*
⟨*proof*⟩

lemma *two-cross-zero-imp-eq-middle*:
fixes *a p c q* :: *real point*
assumes *cross a p q* = *0*
and *cross p c q* = *0*
and *cross a p c* ≠ *0*
shows *q* = *p*
⟨*proof*⟩

lemma *cross-pos-trans-coords*:
fixes *ax ay bx b-y cx cy dx d-y* :: *real*
assumes (*ax*, *ay*) < (*bx*, *b-y*)
and (*bx*, *b-y*) < (*cx*, *cy*)
and (*cx*, *cy*) < (*dx*, *d-y*)
and *0* < *cross* (*ax*, *ay*) (*bx*, *b-y*) (*cx*, *cy*)
and *0* < *cross* (*bx*, *b-y*) (*cx*, *cy*) (*dx*, *d-y*)
shows *0* < *cross* (*ax*, *ay*) (*bx*, *b-y*) (*dx*, *d-y*)
and *0* < *cross* (*ax*, *ay*) (*cx*, *cy*) (*dx*, *d-y*)
⟨*proof*⟩

lemma *cross-pos-trans-left-coords*:
fixes *ax ay bx b-y cx cy dx d-y* :: *real*
assumes (*ax*, *ay*) < (*bx*, *b-y*)
and (*bx*, *b-y*) < (*cx*, *cy*)
and (*cx*, *cy*) < (*dx*, *d-y*)
and *0* < *cross* (*ax*, *ay*) (*bx*, *b-y*) (*cx*, *cy*)
and *0* < *cross* (*bx*, *b-y*) (*cx*, *cy*) (*dx*, *d-y*)
shows *0* < *cross* (*ax*, *ay*) (*bx*, *b-y*) (*dx*, *d-y*)
⟨*proof*⟩

lemma *cross-pos-trans-left*:
fixes *a b c d* :: *real point*
assumes *a* < *b* **and** *b* < *c* **and** *c* < *d*
and *0* < *cross a b c* **and** *0* < *cross b c d*
shows *0* < *cross a b d*
⟨*proof*⟩

lemma *cross-pos-trans-right-coords*:
fixes *ax ay bx b-y cx cy dx d-y* :: *real*
assumes (*ax*, *ay*) < (*bx*, *b-y*)
and (*bx*, *b-y*) < (*cx*, *cy*)
and (*cx*, *cy*) < (*dx*, *d-y*)
and *0* < *cross* (*ax*, *ay*) (*bx*, *b-y*) (*cx*, *cy*)
and *0* < *cross* (*bx*, *b-y*) (*cx*, *cy*) (*dx*, *d-y*)
shows *0* < *cross* (*ax*, *ay*) (*cx*, *cy*) (*dx*, *d-y*)
⟨*proof*⟩

lemma *cross-pos-trans-right*:
fixes *a b c d* :: *real point*
assumes *a* < *b* **and** *b* < *c* **and** *c* < *d*
and *0* < *cross a b c* **and** *0* < *cross b c d*
shows *0* < *cross a c d*
⟨*proof*⟩

fun *stack-turns* :: ('*a*::*linordered-idom*) *point list* ⇒ *bool*
where
stack-turns [] = *True*
| *stack-turns* [*p*] = *True*
| *stack-turns* [*p*, *q*] = *True*
| *stack-turns* (*p* # *q* # *r* # *ps*) =
(*left-turn r q p* ∧ *stack-turns* (*q* # *r* # *ps*))

definition *chain-turns* :: ('*a*::*linordered-idom*) *point list* ⇒ *bool*
where
chain-turns ps ⟷ *stack-turns* (*rev ps*)

fun *scan-push* ::
('*a*::*linordered-idom*) *point list* ⇒ '*a point* ⇒ '*a point list*
where

```
  scan-push [] p = [p]
| scan-push [q] p = [p, q]
| scan-push (q # r # st) p =
    (if cross r q p ≤ 0 then scan-push (r # st) p else p # q # r # st)

definition scan-stack ::
  ('a::linordered-idom) point list ⇒ 'a point list
where
  scan-stack ps = foldl scan-push [] ps

definition scan-chain ::
  ('a::linordered-idom) point list ⇒ 'a point list
where
  scan-chain ps = rev (scan-stack ps)

definition sorted-unique ::
  ('a::linorder) list ⇒ 'a list
where
  sorted-unique xs = sorted-list-of-set (set xs)

definition lower-hull ::
  ('a::{linorder,linordered-idom}) point list ⇒ 'a point list
where
  lower-hull ps = scan-chain (sorted-unique ps)

definition upper-hull ::
  ('a::{linorder,linordered-idom}) point list ⇒ 'a point list
where
  upper-hull ps = scan-chain (rev (sorted-unique ps))

definition andrew-hull ::
  ('a::{linorder,linordered-idom}) point list ⇒ 'a point list
where
  andrew-hull ps =
    (case sorted-unique ps of
      [] ⇒ []
    | [p] ⇒ [p]
    | - ⇒ butlast (lower-hull ps) @ butlast (upper-hull ps))

lemma sorted-unique-set [simp]:
  set (sorted-unique xs) = set xs
  ⟨proof⟩

lemma sorted-unique-distinct [simp]:
  distinct (sorted-unique xs)
  ⟨proof⟩

lemma sorted-unique-sorted [simp]:
  sorted (sorted-unique xs)
```

⟨*proof*⟩

lemma *sorted-unique-Nil-iff* [*simp*]:
sorted-unique xs = [] ⟷ *xs* = []
⟨*proof*⟩

lemma *sorted-unique-singleton-iff*:
sorted-unique xs = [*p*] ⟷ *set xs* = {*p*}
⟨*proof*⟩

lemma *set-scan-push-subset*:
set (*scan-push st p*) ⊆ *insert p* (*set st*)
⟨*proof*⟩

lemma *set-scan-stack-subset*:
set (*scan-stack ps*) ⊆ *set ps*
⟨*proof*⟩

lemma *set-scan-chain-subset*:
set (*scan-chain ps*) ⊆ *set ps*
⟨*proof*⟩

lemma *lower-hull-subset*:
set (*lower-hull ps*) ⊆ *set ps*
⟨*proof*⟩

lemma *upper-hull-subset*:
set (*upper-hull ps*) ⊆ *set ps*
⟨*proof*⟩

theorem *andrew-hull-subset*:
set (*andrew-hull ps*) ⊆ *set ps*
⟨*proof*⟩

1.1 Convex-Hull Correctness Interface

The executable algorithm is generic over ordered integral domains. The geometric convex-hull specification is stated here for real coordinates, where Isabelle's convexity library supplies the ambient real vector-space structure on products.

The predicate is the envelope specification proved for the algorithm. The first conjunct says that every returned vertex is an input point. The second conjunct is equality of convex hulls, which includes both coverage of the input by the returned envelope and the absence of any hull area outside the input hull. Thus the predicate is stronger than one-sided soundness.

definition *convex-hull-correct* ::
real point list ⇒ *real point list* ⇒ *bool*
where

convex-hull-correct ps hs $\longleftrightarrow$
set hs $\subseteq$ *set ps* $\wedge$ *convex hull set hs* $=$ *convex hull set ps*

definition *convex-hull-irredundant* :: *real point list* $\Rightarrow$ *bool*
where
convex-hull-irredundant hs $\longleftrightarrow$
$(\forall\, p{\in}$*set hs*. *convex hull* (*set hs* $-$ $\{p\}$) $\neq$ *convex hull set hs*)

lemma *strict-support-notin-convex-hull*:
fixes *p v* :: *real point*
assumes *strict*: $\bigwedge q.\ q \in S \Longrightarrow$ *inner v q* $<$ *inner v p*
shows *p* $\notin$ *convex hull S*
⟨*proof*⟩

lemma *strict-support-hull-delete-ne*:
fixes *p v* :: *real point*
assumes *p*: *p* $\in$ *S*
and *strict*: $\bigwedge q.\ q \in S - \{p\} \Longrightarrow$ *inner v q* $<$ *inner v p*
shows *convex hull* (*S* $-$ $\{p\}$) $\neq$ *convex hull S*
⟨*proof*⟩

theorem *andrew-hull-convex-hull-subset*:
fixes *ps* :: *real point list*
shows *convex hull set* (*andrew-hull ps*) $\subseteq$ *convex hull set ps*
⟨*proof*⟩

lemma *convex-hull-eq-from-mutual-inclusion*:
fixes *xs ys* :: (*'a*::*real-vector*) *list*
assumes *set xs* $\subseteq$ *convex hull set ys*
and *set ys* $\subseteq$ *convex hull set xs*
shows *convex hull set xs* $=$ *convex hull set ys*
⟨*proof*⟩

theorem *andrew-hull-convex-hull-eqI*:
fixes *ps* :: *real point list*
assumes *set ps* $\subseteq$ *convex hull set* (*andrew-hull ps*)
shows *convex hull set* (*andrew-hull ps*) $=$ *convex hull set ps*
⟨*proof*⟩

theorem *andrew-hull-convex-hull-eq-iff*:
fixes *ps* :: *real point list*
shows *convex hull set* (*andrew-hull ps*) $=$ *convex hull set ps* $\longleftrightarrow$
set ps $\subseteq$ *convex hull set* (*andrew-hull ps*)
⟨*proof*⟩

theorem *andrew-hull-correctI*:
fixes *ps* :: *real point list*
assumes *set ps* $\subseteq$ *convex hull set* (*andrew-hull ps*)
shows *convex-hull-correct ps* (*andrew-hull ps*)

⟨*proof*⟩

theorem *andrew-hull-correct-iff*:
fixes *ps* :: *real point list*
shows *convex-hull-correct ps* (*andrew-hull ps*) ⟷
set ps ⊆ *convex hull set* (*andrew-hull ps*)
⟨*proof*⟩

lemma *distinct-scan-push*:
assumes *distinct st* **and** *p* ∉ *set st*
shows *distinct* (*scan-push st p*)
⟨*proof*⟩

lemma *distinct-scan-stack*:
assumes *distinct ps*
shows *distinct* (*scan-stack ps*)
⟨*proof*⟩

lemma *distinct-scan-chain*:
assumes *distinct ps*
shows *distinct* (*scan-chain ps*)
⟨*proof*⟩

lemma *distinct-lower-hull*:
distinct (*lower-hull ps*)
⟨*proof*⟩

lemma *distinct-upper-hull*:
distinct (*upper-hull ps*)
⟨*proof*⟩

lemma *stack-turns-scan-push*:
assumes *stack-turns st*
shows *stack-turns* (*scan-push st p*)
⟨*proof*⟩

lemma *stack-turns-scan-stack*:
stack-turns (*scan-stack ps*)
⟨*proof*⟩

theorem *chain-turns-scan-chain*:
chain-turns (*scan-chain ps*)
⟨*proof*⟩

theorem *lower-hull-turns*:
chain-turns (*lower-hull ps*)
⟨*proof*⟩

theorem *upper-hull-turns*:

chain-turns (*upper-hull ps*)
⟨*proof*⟩

lemma *stack-turns-tl*:
assumes *stack-turns xs*
shows *stack-turns* (*tl xs*)
⟨*proof*⟩

lemma *stack-turns-butlast*:
assumes *stack-turns xs*
shows *stack-turns* (*butlast xs*)
⟨*proof*⟩

lemma *stack-turns-append-last3*:
assumes *stack-turns* (*xs* @ [*z*, *y*, *x*])
shows *left-turn x y z*
⟨*proof*⟩

lemma *chain-turns-tl*:
assumes *chain-turns xs*
shows *chain-turns* (*tl xs*)
⟨*proof*⟩

lemma *chain-turns-first*:
assumes *chain-turns* (*x # y # z # xs*)
shows *left-turn x y z*
⟨*proof*⟩

lemma *sorted-chain-cross-first-two*:
fixes *x y z* :: *real point*
assumes *sorted*: *sorted-wrt* (<) (*x # y # zs*)
and *turns*: *chain-turns* (*x # y # zs*)
and *z*: *z* ∈ *set zs*
shows *0* < *cross x y z*
⟨*proof*⟩

lemma *sorted-chain-cross-nth-increasing*:
fixes *xs* :: *real point list*
assumes *sorted*: *sorted-wrt* (<) *xs*
and *turns*: *chain-turns xs*
and *ij*: *i* < *j*
and *jk*: *j* < *k*
and *k-len*: *k* < *length xs*
shows *0* < *cross* (*xs* ! *i*) (*xs* ! *j*) (*xs* ! *k*)
⟨*proof*⟩

lemma *uminus-less-point-iff* [*simp*]:
fixes *p q* :: *real point*
shows − *p* < − *q* ⟷ *q* < *p*

⟨*proof*⟩

lemma *cross-uminus* [*simp*]:
fixes *p q r* :: *real point*
shows *cross* (− *p*) (− *q*) (− *r*) = *cross p q r*
⟨*proof*⟩

lemma *left-turn-uminus* [*simp*]:
fixes *p q r* :: *real point*
shows *left-turn* (− *p*) (− *q*) (− *r*) ⟷ *left-turn p q r*
⟨*proof*⟩

lemma *stack-turns-map-uminus* [*simp*]:
fixes *xs* :: *real point list*
shows *stack-turns* (*map* (λ*p*. − *p*) *xs*) ⟷ *stack-turns xs*
⟨*proof*⟩

lemma *chain-turns-map-uminus* [*simp*]:
fixes *xs* :: *real point list*
shows *chain-turns* (*map* (λ*p*. − *p*) *xs*) ⟷ *chain-turns xs*
⟨*proof*⟩

lemma *sorted-wrt-less-map-uminus*:
fixes *xs* :: *real point list*
assumes *sorted-wrt* (>) *xs*
shows *sorted-wrt* (<) (*map* (λ*p*. − *p*) *xs*)
⟨*proof*⟩

lemma *sorted-chain-cross-nth-decreasing*:
fixes *xs* :: *real point list*
assumes *sorted*: *sorted-wrt* (>) *xs*
and *turns*: *chain-turns xs*
and *ij*: *i* < *j*
and *jk*: *j* < *k*
and *k-len*: *k* < *length xs*
shows *0* < *cross* (*xs* ! *i*) (*xs* ! *j*) (*xs* ! *k*)
⟨*proof*⟩

lemma *sorted-chain-edge-cross-nonneg-increasing*:
fixes *xs* :: *real point list*
assumes *sorted*: *sorted-wrt* (<) *xs*
and *turns*: *chain-turns xs*
and *edge*: *Suc i* < *length xs*
and *q*: *q* ∈ *set xs*
shows *0* ≤ *cross* (*xs* ! *i*) (*xs* ! *Suc i*) *q*
⟨*proof*⟩

lemma *sorted-chain-edge-cross-nonneg-decreasing*:
fixes *xs* :: *real point list*

assumes *sorted*: *sorted-wrt* (>) *xs*
and *turns*: *chain-turns xs*
and *edge*: *Suc i* < *length xs*
and *q*: *q* ∈ *set xs*
shows *0* ≤ *cross* (*xs* ! *i*) (*xs* ! *Suc i*) *q*
⟨*proof*⟩

lemma *scan-push-nonempty* [*simp*]:
scan-push st p ≠ []
⟨*proof*⟩

lemma *scan-stack-nonempty*:
ps ≠ [] ⟹ *scan-stack ps* ≠ []
⟨*proof*⟩

lemma *scan-chain-nonempty*:
ps ≠ [] ⟹ *scan-chain ps* ≠ []
⟨*proof*⟩

lemma *lower-hull-nonempty*:
ps ≠ [] ⟹ *lower-hull ps* ≠ []
⟨*proof*⟩

lemma *upper-hull-nonempty*:
ps ≠ [] ⟹ *upper-hull ps* ≠ []
⟨*proof*⟩

1.2 Endpoints of the Scans

lemma *hd-scan-push* [*simp*]:
hd (*scan-push st p*) = *p*
⟨*proof*⟩

lemma *last-scan-push*:
assumes *st* ≠ []
shows *last* (*scan-push st p*) = *last st*
⟨*proof*⟩

lemma *hd-scan-stack*:
assumes *ps* ≠ []
shows *hd* (*scan-stack ps*) = *last ps*
⟨*proof*⟩

lemma *last-scan-stack*:
assumes *ps* ≠ []
shows *last* (*scan-stack ps*) = *hd ps*
⟨*proof*⟩

lemma *hd-scan-chain*:

assumes $ps \neq []$
shows $hd\ (scan\text{-}chain\ ps) = hd\ ps$
⟨*proof*⟩

lemma *last-scan-chain*:
assumes $ps \neq []$
shows $last\ (scan\text{-}chain\ ps) = last\ ps$
⟨*proof*⟩

lemma *hd-lower-hull*:
assumes $ps \neq []$
shows $hd\ (lower\text{-}hull\ ps) = hd\ (sorted\text{-}unique\ ps)$
⟨*proof*⟩

lemma *last-lower-hull*:
assumes $ps \neq []$
shows $last\ (lower\text{-}hull\ ps) = last\ (sorted\text{-}unique\ ps)$
⟨*proof*⟩

lemma *hd-upper-hull*:
assumes $ps \neq []$
shows $hd\ (upper\text{-}hull\ ps) = last\ (sorted\text{-}unique\ ps)$
⟨*proof*⟩

lemma *last-upper-hull*:
assumes $ps \neq []$
shows $last\ (upper\text{-}hull\ ps) = hd\ (sorted\text{-}unique\ ps)$
⟨*proof*⟩

lemma *length-scan-chain-ge2*:
assumes $xs \neq []$ **and** $hd\ xs \neq last\ xs$
shows $2 \leq length\ (scan\text{-}chain\ xs)$
⟨*proof*⟩

lemma *hd-ne-last-sorted-unique-if-card-ge2*:
assumes $2 \leq card\ (set\ ps)$
shows $hd\ (sorted\text{-}unique\ ps) \neq last\ (sorted\text{-}unique\ ps)$
⟨*proof*⟩

lemma *length-lower-hull-ge2*:
assumes $2 \leq card\ (set\ ps)$
shows $2 \leq length\ (lower\text{-}hull\ ps)$
⟨*proof*⟩

lemma *length-upper-hull-ge2*:
assumes $2 \leq card\ (set\ ps)$
shows $2 \leq length\ (upper\text{-}hull\ ps)$
⟨*proof*⟩

lemma *set-butlast-last*:
assumes $xs \neq []$
shows $set\ xs = set\ (butlast\ xs) \cup \{last\ xs\}$
⟨*proof*⟩

lemma *hd-mem-set-butlast*:
assumes $2 \leq length\ xs$
shows $hd\ xs \in set\ (butlast\ xs)$
⟨*proof*⟩

theorem *set-andrew-hull*:
$set\ (andrew\text{-}hull\ ps) = set\ (lower\text{-}hull\ ps) \cup set\ (upper\text{-}hull\ ps)$
⟨*proof*⟩

1.3 Support-Function Invariants

definition *support-value* :: *real point* ⇒ *real point* ⇒ *real*
where
$support\text{-}value\ v\ p = fst\ v * fst\ p + snd\ v * snd\ p$

lemma *support-value-eq-inner*:
$support\text{-}value\ v\ p = inner\ v\ p$
⟨*proof*⟩

lemma *support-value-edge-normal*:
fixes *a b x* :: *real point*
shows $support\text{-}value\ (snd\ b - snd\ a, fst\ a - fst\ b)\ x =$
$support\text{-}value\ (snd\ b - snd\ a, fst\ a - fst\ b)\ a - cross\ a\ b\ x$
⟨*proof*⟩

lemma *support-middle-le-max-increasing*:
fixes *r q p v* :: *real point*
assumes *rq*: $r < q$ **and** *qp*: $q < p$
and *turn*: $cross\ r\ q\ p \leq 0$
and *neg*: $snd\ v < 0$
shows $support\text{-}value\ v\ q \leq max\ (support\text{-}value\ v\ r)\ (support\text{-}value\ v\ p)$
⟨*proof*⟩

lemma *support-middle-le-max-decreasing*:
fixes *r q p v* :: *real point*
assumes *pq*: $p < q$ **and** *qr*: $q < r$
and *turn*: $cross\ r\ q\ p \leq 0$
and *pos*: $0 < snd\ v$
shows $support\text{-}value\ v\ q \leq max\ (support\text{-}value\ v\ r)\ (support\text{-}value\ v\ p)$
⟨*proof*⟩

lemma *scan-push-strict-sorted-increasing*:
fixes *st* :: *real point list*
assumes *sorted*: $sorted\text{-}wrt\ (<)\ (rev\ st)$

and *below*: $\bigwedge q.\ q \in set\ st \Longrightarrow q < p$
shows *sorted-wrt* $(<)$ $(rev\ (scan\text{-}push\ st\ p))$
⟨*proof*⟩

lemma *scan-stack-strict-sorted-increasing*:
fixes *xs* :: *real point list*
assumes *sorted-wrt* $(<)$ *xs*
shows *sorted-wrt* $(<)$ $(rev\ (scan\text{-}stack\ xs))$
⟨*proof*⟩

lemma *scan-push-support-dominates-increasing-step*:
fixes *q r p x* :: *real point* **and** *st* :: *real point list*
assumes *tail-dom*:
$cross\ r\ q\ p \leq 0 \Longrightarrow$
$(\bigwedge x.\ x \in insert\ p\ (set\ (r\ \#\ st)) \Longrightarrow$
$\exists\, y{\in}set\ (scan\text{-}push\ (r\ \#\ st)\ p).\ support\text{-}value\ v\ x \leq support\text{-}value\ v\ y)$
and *sorted*: *sorted-wrt* $(<)$ $(rev\ (q\ \#\ r\ \#\ st))$
and *below*: $\bigwedge z.\ z \in set\ (q\ \#\ r\ \#\ st) \Longrightarrow z < p$
and *neg*: $snd\ v < 0$
and *x-in*: $x \in insert\ p\ (set\ (q\ \#\ r\ \#\ st))$
shows $\exists\, y{\in}set\ (scan\text{-}push\ (q\ \#\ r\ \#\ st)\ p).\ support\text{-}value\ v\ x \leq support\text{-}value\ v\ y$
⟨*proof*⟩

lemma *scan-push-support-dominates-increasing*:
fixes *st* :: *real point list*
shows *sorted-wrt* $(<)$ $(rev\ st) \Longrightarrow$
$(\bigwedge q.\ q \in set\ st \Longrightarrow q < p) \Longrightarrow$
$snd\ v < 0 \Longrightarrow$
$x \in insert\ p\ (set\ st) \Longrightarrow$
$\exists\, y{\in}set\ (scan\text{-}push\ st\ p).\ support\text{-}value\ v\ x \leq support\text{-}value\ v\ y$
⟨*proof*⟩

lemma *scan-stack-support-dominates-increasing*:
fixes *xs* :: *real point list*
assumes *sorted*: *sorted-wrt* $(<)$ *xs*
and *neg*: $snd\ v < 0$
and *x*: $x \in set\ xs$
shows $\exists\, y{\in}set\ (scan\text{-}stack\ xs).\ support\text{-}value\ v\ x \leq support\text{-}value\ v\ y$
⟨*proof*⟩

lemma *scan-chain-support-dominates-increasing*:
fixes *xs* :: *real point list*
assumes *sorted-wrt* $(<)$ *xs* **and** $snd\ v < 0$ **and** $x \in set\ xs$
shows $\exists\, y{\in}set\ (scan\text{-}chain\ xs).\ support\text{-}value\ v\ x \leq support\text{-}value\ v\ y$
⟨*proof*⟩

lemma *sorted-wrt-less-sorted-unique* [*simp*]:
sorted-wrt $(<)$ $(sorted\text{-}unique\ (xs :: 'a{::}linorder\ list))$
⟨*proof*⟩

lemma *lower-hull-supports-negative*:
assumes *snd v* < *0* **and** *x* ∈ *set ps*
shows ∃ *y*∈*set* (*lower-hull ps*). *support-value v x* ≤ *support-value v y*
⟨*proof*⟩

lemma *scan-push-strict-sorted-decreasing*:
fixes *st* :: *real point list*
assumes *sorted*: *sorted-wrt* (>) (*rev st*)
and *above*: ⋀*q*. *q* ∈ *set st* ⟹ *p* < *q*
shows *sorted-wrt* (>) (*rev* (*scan-push st p*))
⟨*proof*⟩

lemma *scan-stack-strict-sorted-decreasing*:
fixes *xs* :: *real point list*
assumes *sorted-wrt* (>) *xs*
shows *sorted-wrt* (>) (*rev* (*scan-stack xs*))
⟨*proof*⟩

lemma *scan-push-support-dominates-decreasing-step*:
fixes *q r p x* :: *real point* **and** *st* :: *real point list*
assumes *tail-dom*:
cross r q p ≤ *0* ⟹
(⋀*x*. *x* ∈ *insert p* (*set* (*r* # *st*)) ⟹
∃ *y*∈*set* (*scan-push* (*r* # *st*) *p*). *support-value v x* ≤ *support-value v y*)
and *sorted*: *sorted-wrt* (>) (*rev* (*q* # *r* # *st*))
and *above*: ⋀*z*. *z* ∈ *set* (*q* # *r* # *st*) ⟹ *p* < *z*
and *pos*: *0* < *snd v*
and *x-in*: *x* ∈ *insert p* (*set* (*q* # *r* # *st*))
shows ∃ *y*∈*set* (*scan-push* (*q* # *r* # *st*) *p*). *support-value v x* ≤ *support-value v y*
⟨*proof*⟩

lemma *scan-push-support-dominates-decreasing*:
fixes *st* :: *real point list*
shows *sorted-wrt* (>) (*rev st*) ⟹
(⋀*q*. *q* ∈ *set st* ⟹ *p* < *q*) ⟹
0 < *snd v* ⟹
x ∈ *insert p* (*set st*) ⟹
∃ *y*∈*set* (*scan-push st p*). *support-value v x* ≤ *support-value v y*
⟨*proof*⟩

lemma *scan-stack-support-dominates-decreasing*:
fixes *xs* :: *real point list*
assumes *sorted*: *sorted-wrt* (>) *xs*
and *pos*: *0* < *snd v*
and *x*: *x* ∈ *set xs*
shows ∃ *y*∈*set* (*scan-stack xs*). *support-value v x* ≤ *support-value v y*
⟨*proof*⟩

lemma *scan-chain-support-dominates-decreasing*:
fixes *xs* :: *real point list*
assumes *sorted-wrt* (>) *xs* **and** *0* < *snd v* **and** *x* ∈ *set xs*
shows ∃ *y*∈*set* (*scan-chain xs*). *support-value v x* ≤ *support-value v y*
⟨*proof*⟩

lemma *sorted-wrt-greater-rev-sorted-unique* [*simp*]:
sorted-wrt (>) (*rev* (*sorted-unique* (*xs* :: *'a*::*linorder list*)))
⟨*proof*⟩

lemma *upper-hull-supports-positive*:
assumes *0* < *snd v* **and** *x* ∈ *set ps*
shows ∃ *y*∈*set* (*upper-hull ps*). *support-value v x* ≤ *support-value v y*
⟨*proof*⟩

theorem *lower-hull-sorted*:
fixes *ps* :: *real point list*
shows *sorted-wrt* (<) (*lower-hull ps*)
⟨*proof*⟩

theorem *upper-hull-sorted*:
fixes *ps* :: *real point list*
shows *sorted-wrt* (>) (*upper-hull ps*)
⟨*proof*⟩

lemma *fst-hd-sorted-unique-le*:
fixes *x* :: *real point*
assumes *x* ∈ *set ps*
shows *fst* (*hd* (*sorted-unique ps*)) ≤ *fst x*
⟨*proof*⟩

lemma *fst-le-last-sorted-unique*:
fixes *x* :: *real point*
assumes *x* ∈ *set ps*
shows *fst x* ≤ *fst* (*last* (*sorted-unique ps*))
⟨*proof*⟩

lemma *hd-sorted-unique-less*:
fixes *x* :: *real point*
assumes *x*: *x* ∈ *set ps*
and *ne*: *x* ≠ *hd* (*sorted-unique ps*)
shows *hd* (*sorted-unique ps*) < *x*
⟨*proof*⟩

lemma *less-last-sorted-unique*:
fixes *x* :: *real point*
assumes *x*: *x* ∈ *set ps*
and *ne*: *x* ≠ *last* (*sorted-unique ps*)
shows *x* < *last* (*sorted-unique ps*)

⟨*proof*⟩

lemma *lex-min-strict-support*:
  fixes p :: *real point*
  assumes *fin*: *finite* S
    and *lex*: $\bigwedge q.\ q \in S - \{p\} \Longrightarrow p < q$
  shows $\exists\, v.\ \forall\, q{\in}S - \{p\}.\ \mathit{inner}\ v\ q < \mathit{inner}\ v\ p$
⟨*proof*⟩

lemma *lex-max-strict-support*:
  fixes p :: *real point*
  assumes *fin*: *finite* S
    and *lex*: $\bigwedge q.\ q \in S - \{p\} \Longrightarrow q < p$
  shows $\exists\, v.\ \forall\, q{\in}S - \{p\}.\ \mathit{inner}\ v\ q < \mathit{inner}\ v\ p$
⟨*proof*⟩

lemma *lower-hull-edge-cross-nonneg-input-if-fst-less*:
  fixes ps :: *real point list*
  defines $L \equiv \mathit{lower\text{-}hull}\ ps$
  assumes *edge*: $\mathit{Suc}\ i < \mathit{length}\ L$
    and *fst-less*: $\mathit{fst}\ (L\ !\ i) < \mathit{fst}\ (L\ !\ \mathit{Suc}\ i)$
    and x: $x \in \mathit{set}\ ps$
  shows $0 \le \mathit{cross}\ (L\ !\ i)\ (L\ !\ \mathit{Suc}\ i)\ x$
⟨*proof*⟩

lemma *upper-hull-edge-cross-nonneg-input-if-fst-greater*:
  fixes ps :: *real point list*
  defines $U \equiv \mathit{upper\text{-}hull}\ ps$
  assumes *edge*: $\mathit{Suc}\ i < \mathit{length}\ U$
    and *fst-greater*: $\mathit{fst}\ (U\ !\ \mathit{Suc}\ i) < \mathit{fst}\ (U\ !\ i)$
    and x: $x \in \mathit{set}\ ps$
  shows $0 \le \mathit{cross}\ (U\ !\ i)\ (U\ !\ \mathit{Suc}\ i)\ x$
⟨*proof*⟩

lemma *lower-hull-vertical-edge-last*:
  fixes ps :: *real point list*
  defines $L \equiv \mathit{lower\text{-}hull}\ ps$
  assumes *edge*: $\mathit{Suc}\ i < \mathit{length}\ L$
    and *same-x*: $\mathit{fst}\ (L\ !\ i) = \mathit{fst}\ (L\ !\ \mathit{Suc}\ i)$
  shows $\mathit{Suc}\ i = \mathit{length}\ L - 1$
⟨*proof*⟩

lemma *upper-hull-vertical-edge-last*:
  fixes ps :: *real point list*
  defines $U \equiv \mathit{upper\text{-}hull}\ ps$
  assumes *edge*: $\mathit{Suc}\ i < \mathit{length}\ U$
    and *same-x*: $\mathit{fst}\ (U\ !\ i) = \mathit{fst}\ (U\ !\ \mathit{Suc}\ i)$
  shows $\mathit{Suc}\ i = \mathit{length}\ U - 1$
⟨*proof*⟩

lemma *lower-hull-edge-cross-nonneg-input*:
fixes *ps* :: *real point list*
defines $L \equiv$ *lower-hull ps*
assumes *edge*: *Suc i* $<$ *length L*
and *x*: $x \in$ *set ps*
shows $0 \leq$ *cross* (*L* ! *i*) (*L* ! *Suc i*) *x*
⟨*proof*⟩

lemma *upper-hull-edge-cross-nonneg-input*:
fixes *ps* :: *real point list*
defines $U \equiv$ *upper-hull ps*
assumes *edge*: *Suc i* $<$ *length U*
and *x*: $x \in$ *set ps*
shows $0 \leq$ *cross* (*U* ! *i*) (*U* ! *Suc i*) *x*
⟨*proof*⟩

lemma *strict-support-from-two-edges*:
fixes *a p c* :: *real point*
assumes *left*: $\bigwedge q.\ q \in S \Longrightarrow 0 \leq$ *cross a p q*
and *right*: $\bigwedge q.\ q \in S \Longrightarrow 0 \leq$ *cross p c q*
and *noncol*: *cross a p c* $\neq 0$
shows $\exists v.\ \forall q{\in}S - \{p\}.$ *inner v q* $<$ *inner v p*
⟨*proof*⟩

lemma *lower-hull-interior-strict-support*:
fixes *ps* :: *real point list*
defines $L \equiv$ *lower-hull ps*
assumes *i-pos*: $0 < i$
and *i-len*: *Suc i* $<$ *length L*
shows $\exists v.\ \forall q{\in}$*set ps* $- \{L\ !\ i\}.$ *inner v q* $<$ *inner v* (*L* ! *i*)
⟨*proof*⟩

lemma *upper-hull-interior-strict-support*:
fixes *ps* :: *real point list*
defines $U \equiv$ *upper-hull ps*
assumes *i-pos*: $0 < i$
and *i-len*: *Suc i* $<$ *length U*
shows $\exists v.\ \forall q{\in}$*set ps* $- \{U\ !\ i\}.$ *inner v q* $<$ *inner v* (*U* ! *i*)
⟨*proof*⟩

lemma *lower-hull-interior-strict-support-if-fst-less*:
fixes *ps* :: *real point list*
defines $L \equiv$ *lower-hull ps*
assumes *i-pos*: $0 < i$
and *i-len*: *Suc i* $<$ *length L*
and *fst-left*: *fst* (*L* ! (*i* − *1*)) $<$ *fst* (*L* ! *i*)
and *fst-right*: *fst* (*L* ! *i*) $<$ *fst* (*L* ! *Suc i*)
shows $\exists v.\ \forall q{\in}$*set ps* $- \{L\ !\ i\}.$ *inner v q* $<$ *inner v* (*L* ! *i*)

⟨*proof*⟩

lemma *hd-sorted-unique-mem-andrew-hull*:
assumes $ps \neq []$
shows *hd* (*sorted-unique ps*) ∈ *set* (*andrew-hull ps*)
⟨*proof*⟩

lemma *last-sorted-unique-mem-andrew-hull*:
assumes $ps \neq []$
shows *last* (*sorted-unique ps*) ∈ *set* (*andrew-hull ps*)
⟨*proof*⟩

lemma *andrew-hull-supports-horizontal*:
assumes *x*: $x \in set\ ps$ **and** *horizontal*: *snd v* = *0*
shows ∃ *y*∈*set* (*andrew-hull ps*). *support-value v x* ≤ *support-value v y*
⟨*proof*⟩

lemma *andrew-hull-supports*:
assumes *x*: $x \in set\ ps$
shows ∃ *y*∈*set* (*andrew-hull ps*). *support-value v x* ≤ *support-value v y*
⟨*proof*⟩

declare *support-value-eq-inner* [*simp*]

theorem *andrew-hull-covers-input*:
fixes *ps* :: *real point list*
shows *set ps* ⊆ *convex hull set* (*andrew-hull ps*)
⟨*proof*⟩

theorem *andrew-hull-correct*:
fixes *ps* :: *real point list*
shows *convex-hull-correct ps* (*andrew-hull ps*)
⟨*proof*⟩

lemma *andrew-hull-point-strict-support*:
fixes *ps* :: *real point list*
assumes *p*: $p \in set$ (*andrew-hull ps*)
shows ∃ *v*. ∀ *q*∈*set* (*andrew-hull ps*) − {*p*}. *inner v q* < *inner v p*
⟨*proof*⟩

theorem *andrew-hull-delete-changes-convex-hull*:
fixes *ps* :: *real point list*
assumes *p*: $p \in set$ (*andrew-hull ps*)
shows *convex hull* (*set* (*andrew-hull ps*) − {*p*}) ≠ *convex hull set* (*andrew-hull ps*)
⟨*proof*⟩

theorem *andrew-hull-irredundant*:
fixes *ps* :: *real point list*

shows *convex-hull-irredundant* (*andrew-hull ps*)
⟨*proof*⟩

lemma *length-scan-push-le*:
length (*scan-push st p*) ≤ *Suc* (*length st*)
⟨*proof*⟩

lemma *length-scan-stack-le*:
length (*scan-stack ps*) ≤ *length ps*
⟨*proof*⟩

lemma *length-scan-chain-le*:
length (*scan-chain ps*) ≤ *length ps*
⟨*proof*⟩

lemma *length-lower-hull-le*:
length (*lower-hull ps*) ≤ *card* (*set ps*)
⟨*proof*⟩

lemma *length-upper-hull-le*:
length (*upper-hull ps*) ≤ *card* (*set ps*)
⟨*proof*⟩

theorem *length-andrew-hull-le-twice-card*:
length (*andrew-hull ps*) ≤ *2* ∗ *card* (*set ps*)
⟨*proof*⟩

For inputs with at least two distinct points, the implementation returns exactly the standard Andrew concatenation of lower and upper scans. This theorem is often the most convenient way to unfold the top-level algorithm without exposing the special empty and singleton cases.

theorem *andrew-hull-ge2*:
assumes *card* (*set ps*) ≥ *2*
shows *andrew-hull ps* = *butlast* (*lower-hull ps*) @ *butlast* (*upper-hull ps*)
⟨*proof*⟩

theorem *andrew-hull-ordered-chains*:
fixes *ps* :: *real point list*
assumes *2* ≤ *card* (*set ps*)
shows *andrew-hull ps* = *butlast* (*lower-hull ps*) @ *butlast* (*upper-hull ps*)
and *sorted-wrt* (<) (*lower-hull ps*)
and *sorted-wrt* (>) (*upper-hull ps*)
and *chain-turns* (*lower-hull ps*)
and *chain-turns* (*upper-hull ps*)
⟨*proof*⟩

theorem *distinct-andrew-hull-iff*:
fixes *ps* :: *real point list*
shows *distinct* (*andrew-hull ps*) ⟷

card (*set ps*) < *2* ∨
set (*butlast* (*lower-hull ps*)) ∩ *set* (*butlast* (*upper-hull ps*)) = {}
⟨*proof*⟩

theorem *distinct-andrew-hull-if-trimmed-chains-disjoint*:
fixes *ps* :: *real point list*
assumes *set* (*butlast* (*lower-hull ps*)) ∩ *set* (*butlast* (*upper-hull ps*)) = {}
shows *distinct* (*andrew-hull ps*)
⟨*proof*⟩

1.4 Top-Level Correctness Statement

The final theorem collects the specification in one place. For real-coordinate inputs, the returned list consists only of input points, covers every input point by its convex hull, has exactly the same convex hull as the input, and is irredundant: deleting any returned point changes the convex hull of the returned set. The preceding sortedness and left-turn theorems describe the order and shape of the lower and upper scans; they are supporting invariants, not a substitute for this envelope and irredundancy specification.

The irredundancy conjunct is the semantic minimality property for the returned vertex set. It does not merely follow from convex-hull equality; it is proved separately by exposing every returned point with a strict supporting direction.

theorem *andrew-hull-correctness*:
fixes *ps* :: *real point list*
shows *set* (*andrew-hull ps*) ⊆ *set ps*
and *set ps* ⊆ *convex hull set* (*andrew-hull ps*)
and *convex hull set* (*andrew-hull ps*) = *convex hull set ps*
and *convex-hull-correct ps* (*andrew-hull ps*)
and *convex-hull-irredundant* (*andrew-hull ps*)
⟨*proof*⟩

end

2 Examples

theory *Andrew-Monotone-Chain-Examples*
imports
Andrew-Monotone-Chain
HOL−Analysis.Convex-Euclidean-Space
begin

The first examples use integer coordinates to exercise the polymorphic executable code path: the scan, sorting, and orientation tests work over any ordered integral domain. The geometric convex-hull specification is then instantiated in the following subsection over real coordinates, where Isabelle's convexity library provides the ambient vector-space structure.

abbreviation *square-points* :: (*int* × *int*) *list*
where
square-points ≡
[(*0*, *0*), (*1*, *0*), (*1*, *1*), (*0*, *1*), (*0*, *0*), (*1*, *1*)]

lemma *square-hull*:
andrew-hull square-points = [(*0*, *0*), (*1*, *0*), (*1*, *1*), (*0*, *1*)]
⟨*proof*⟩

lemma *square-lower-hull*:
lower-hull square-points = [(*0*, *0*), (*1*, *0*), (*1*, *1*)]
⟨*proof*⟩

lemma *square-upper-hull*:
upper-hull square-points = [(*1*, *1*), (*0*, *1*), (*0*, *0*)]
⟨*proof*⟩

abbreviation *collinear-points* :: (*int* × *int*) *list*
where
collinear-points ≡ [(*2*, *0*), (*0*, *0*), (*1*, *0*), (*3*, *0*), (*1*, *0*)]

lemma *collinear-hull*:
andrew-hull collinear-points = [(*0*, *0*), (*3*, *0*)]
⟨*proof*⟩

abbreviation *diamond-points* :: (*int* × *int*) *list*
where
diamond-points ≡
[(*0*, *0*), (*1*, *1*), (*2*, *0*), (*1*, −*1*), (*1*, *0*), (*0*, *0*), (*2*, *0*)]

lemma *diamond-hull*:
andrew-hull diamond-points = [(*0*, *0*), (*1*, −*1*), (*2*, *0*), (*1*, *1*)]
⟨*proof*⟩

lemma *diamond-hull-subset*:
set (*andrew-hull diamond-points*) ⊆ *set diamond-points*
⟨*proof*⟩

lemma *diamond-hull-turns-lower*:
chain-turns (*lower-hull diamond-points*)
⟨*proof*⟩

lemma *diamond-hull-turns-upper*:
chain-turns (*upper-hull diamond-points*)
⟨*proof*⟩

2.1 Convex-hull Examples over the Reals

definition *square-real* :: (*real* × *real*) *list*

where
square-real =
[(*0*, *0*), (*1*, *0*), (*1*, *1*), (*0*, *1*), (*0*, *0*), (*1*, *1*)]

definition *square-real-vertices* :: (*real* × *real*) *list*
where
square-real-vertices = [(*0*, *0*), (*1*, *0*), (*1*, *1*), (*0*, *1*)]

lemma *square-real-hull*:
andrew-hull square-real = *square-real-vertices*
⟨*proof*⟩

lemma *square-real-correct*:
convex-hull-correct square-real (*andrew-hull square-real*)
⟨*proof*⟩

lemma *square-real-irredundant*:
convex-hull-irredundant (*andrew-hull square-real*)
⟨*proof*⟩

lemma *square-real-convex-hull*:
convex hull set (*andrew-hull square-real*) = *convex hull set square-real*
⟨*proof*⟩

definition *collinear-real* :: (*real* × *real*) *list*
where
collinear-real = [(*2*, *0*), (*0*, *0*), (*1*, *0*), (*3*, *0*), (*1*, *0*)]

definition *collinear-real-vertices* :: (*real* × *real*) *list*
where
collinear-real-vertices = [(*0*, *0*), (*3*, *0*)]

lemma *collinear-real-hull*:
andrew-hull collinear-real = *collinear-real-vertices*
⟨*proof*⟩

lemma *one-zero-in-collinear-real-hull*:
(*1*::*real*, *0*) ∈ *convex hull set collinear-real-vertices*
⟨*proof*⟩

lemma *two-zero-in-collinear-real-hull*:
(*2*::*real*, *0*) ∈ *convex hull set collinear-real-vertices*
⟨*proof*⟩

lemma *collinear-real-correct*:
convex-hull-correct collinear-real (*andrew-hull collinear-real*)
⟨*proof*⟩

lemma *collinear-real-irredundant*:

convex-hull-irredundant (*andrew-hull collinear-real*)
⟨*proof*⟩

lemma *collinear-real-convex-hull*:
convex hull set (*andrew-hull collinear-real*) = *convex hull set collinear-real*
⟨*proof*⟩

definition *diamond-real* :: (*real* × *real*) *list*
where
diamond-real =
[(*0*, *0*), (*1*, *1*), (*2*, *0*), (*1*, *−1*), (*1*, *0*), (*0*, *0*), (*2*, *0*)]

definition *diamond-real-vertices* :: (*real* × *real*) *list*
where
diamond-real-vertices = [(*0*, *0*), (*1*, *−1*), (*2*, *0*), (*1*, *1*)]

lemma *diamond-real-hull*:
andrew-hull diamond-real = *diamond-real-vertices*
⟨*proof*⟩

lemma *diamond-center-in-computed-hull*:
(*1*::*real*, *0*) ∈ *convex hull set diamond-real-vertices*
⟨*proof*⟩

lemma *diamond-real-correct*:
convex-hull-correct diamond-real (*andrew-hull diamond-real*)
⟨*proof*⟩

lemma *diamond-real-irredundant*:
convex-hull-irredundant (*andrew-hull diamond-real*)
⟨*proof*⟩

lemma *diamond-real-convex-hull*:
convex hull set (*andrew-hull diamond-real*) = *convex hull set diamond-real*
⟨*proof*⟩

value *andrew-hull square-points*
value *andrew-hull collinear-points*
value *andrew-hull diamond-points*

end

References

[1] A. M. Andrew. Another efficient algorithm for convex hulls in two dimensions. *Information Processing Letters*, 9(5):216–219, 1979. DOI: 10.1016/0020-0190(79)90072-3.