

# Andrew's Monotone Chain Convex Hull Algorithm

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

May 12, 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

2

1.1	Convex-Hull Correctness Interface . . . . .	6
1.2	Endpoints of the Scans . . . . .	11
1.3	Support-Function Invariants . . . . .	13
1.4	Top-Level Correctness Statement . . . . .	21
<b>2</b>	<b>Examples</b>	<b>21</b>
2.1	Convex-hull Examples over the Reals . . . . .	22

# 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* ::

$$('a::linordered-idom) \textit{ point} \Rightarrow 'a \textit{ point} \Rightarrow 'a \textit{ point} \Rightarrow 'a$$

**where**

$$\begin{aligned} \textit{cross } p \ q \ r = \\ (fst \ q - fst \ p) * (snd \ r - snd \ p) - \\ (snd \ q - snd \ p) * (fst \ r - fst \ p) \end{aligned}$$

**definition** *left-turn* ::

$(!a::\text{linordered-idom}) \text{ point} \Rightarrow !a \text{ point} \Rightarrow !a \text{ point} \Rightarrow \text{bool}$   
**where**

$\text{left-turn } p \ q \ r \longleftrightarrow \text{cross } p \ q \ r > 0$

**definition** *collinear* ::

$(!a::\text{linordered-idom}) \text{ point} \Rightarrow !a \text{ point} \Rightarrow !a \text{ point} \Rightarrow \text{bool}$   
**where**

$\text{collinear } p \ q \ r \longleftrightarrow \text{cross } p \ q \ r = 0$

**lemma** *cross-same-left* [simp]:  $\text{cross } p \ p \ q = 0$   
(proof)

**lemma** *cross-same-right* [simp]:  $\text{cross } p \ q \ q = 0$   
(proof)

**lemma** *cross-same-outer* [simp]:  $\text{cross } p \ q \ p = 0$   
(proof)

**lemma** *cross-swap-outer*:  
 $\text{cross } p \ q \ r = - \text{cross } r \ q \ p$   
(proof)

**lemma** *cross-swap-last*:  
 $\text{cross } p \ r \ q = - \text{cross } p \ q \ r$   
(proof)

**lemma** *cross-cycle*:  
 $\text{cross } p \ q \ r = \text{cross } q \ r \ p$   
(proof)

**lemma** *two-cross-zero-imp-eq-middle*:  
**fixes**  $a \ p \ c \ q :: \text{real point}$   
**assumes**  $\text{cross } a \ p \ q = 0$   
**and**  $\text{cross } p \ c \ q = 0$   
**and**  $\text{cross } a \ p \ c \neq 0$   
**shows**  $q = p$   
(proof)

**lemma** *cross-pos-trans-coords*:  
**fixes**  $ax \ ay \ bx \ b-y \ cx \ cy \ dx \ d-y :: \text{real}$   
**assumes**  $(ax, ay) < (bx, b-y)$   
**and**  $(bx, b-y) < (cx, cy)$   
**and**  $(cx, cy) < (dx, d-y)$   
**and**  $0 < \text{cross } (ax, ay) (bx, b-y) (cx, cy)$   
**and**  $0 < \text{cross } (bx, b-y) (cx, cy) (dx, d-y)$   
**shows**  $0 < \text{cross } (ax, ay) (bx, b-y) (dx, d-y)$   
**and**  $0 < \text{cross } (ax, ay) (cx, cy) (dx, d-y)$   
(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**  $0 < cross\ (ax, ay)\ (bx, b-y)\ (dx, d-y)$   
 $\langle proof \rangle$

**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**  $0 < cross\ a\ b\ d$   
 $\langle proof \rangle$

**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**  $0 < cross\ (ax, ay)\ (cx, cy)\ (dx, d-y)$   
 $\langle proof \rangle$

**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**  $0 < cross\ a\ c\ d$   
 $\langle proof \rangle$

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

**definition** *chain-turns* ::  $(a::linordered-idom)\ point\ list \Rightarrow bool$   
**where**  
 $chain-turns\ ps \longleftrightarrow stack-turns\ (rev\ ps)$

**fun** *scan-push* ::  
 $(a::linordered-idom)\ point\ list \Rightarrow 'a\ point \Rightarrow 'a\ point\ list$   
**where**

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

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

**definition** *scan-chain* ::  
 $('a::linordered-idom)\ point\ list \Rightarrow 'a\ point\ list$   
**where**  
 $scan-chain\ ps = rev\ (scan-stack\ ps)$

**definition** *sorted-unique* ::  
 $('a::linorder)\ list \Rightarrow 'a\ list$   
**where**  
 $sorted-unique\ xs = sorted-list-of-set\ (set\ xs)$

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

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

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

**lemma** *sorted-unique-set* [*simp*]:  
 $set\ (sorted-unique\ xs) = set\ xs$   
 $\langle proof \rangle$

**lemma** *sorted-unique-distinct* [*simp*]:  
 $distinct\ (sorted-unique\ xs)$   
 $\langle proof \rangle$

**lemma** *sorted-unique-sorted* [*simp*]:  
 $sorted\ (sorted-unique\ xs)$

*<proof>*

**lemma** *sorted-unique-Nil-iff* [*simp*]:

*sorted-unique xs = []  $\longleftrightarrow$  xs = []*

*<proof>*

**lemma** *sorted-unique-singleton-iff*:

*sorted-unique xs = [p]  $\longleftrightarrow$  set xs = {p}*

*<proof>*

**lemma** *set-scan-push-subset*:

*set (scan-push st p)  $\subseteq$  insert p (set st)*

*<proof>*

**lemma** *set-scan-stack-subset*:

*set (scan-stack ps)  $\subseteq$  set ps*

*<proof>*

**lemma** *set-scan-chain-subset*:

*set (scan-chain ps)  $\subseteq$  set ps*

*<proof>*

**lemma** *lower-hull-subset*:

*set (lower-hull ps)  $\subseteq$  set ps*

*<proof>*

**lemma** *upper-hull-subset*:

*set (upper-hull ps)  $\subseteq$  set ps*

*<proof>*

**theorem** *andrew-hull-subset*:

*set (andrew-hull ps)  $\subseteq$  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  $\Rightarrow$  real point list  $\Rightarrow$  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*  $\longleftrightarrow$   
 $(\forall p \in \text{set } hs. \text{convex hull } (\text{set } hs - \{p\}) \neq \text{convex hull set } hs)$

**lemma** *strict-support-notin-convex-hull*:

**fixes** *p v* :: *real point*  
**assumes** *strict*:  $\bigwedge q. q \in S \implies \text{inner } v \ q < \text{inner } v \ p$   
**shows**  $p \notin \text{convex hull } S$   
 $\langle \text{proof} \rangle$

**lemma** *strict-support-hull-delete-ne*:

**fixes** *p v* :: *real point*  
**assumes** *p*:  $p \in S$   
**and** *strict*:  $\bigwedge q. q \in S - \{p\} \implies \text{inner } v \ q < \text{inner } v \ p$   
**shows**  $\text{convex hull } (S - \{p\}) \neq \text{convex hull } S$   
 $\langle \text{proof} \rangle$

**theorem** *andrew-hull-convex-hull-subset*:

**fixes** *ps* :: *real point list*  
**shows**  $\text{convex hull set } (\text{andrew-hull } ps) \subseteq \text{convex hull set } ps$   
 $\langle \text{proof} \rangle$

**lemma** *convex-hull-eq-from-mutual-inclusion*:

**fixes** *xs ys* ::  $(\text{'a}::\text{real-vector})$  *list*  
**assumes**  $\text{set } xs \subseteq \text{convex hull set } ys$   
**and**  $\text{set } ys \subseteq \text{convex hull set } xs$   
**shows**  $\text{convex hull set } xs = \text{convex hull set } ys$   
 $\langle \text{proof} \rangle$

**theorem** *andrew-hull-convex-hull-eqI*:

**fixes** *ps* :: *real point list*  
**assumes**  $\text{set } ps \subseteq \text{convex hull set } (\text{andrew-hull } ps)$   
**shows**  $\text{convex hull set } (\text{andrew-hull } ps) = \text{convex hull set } ps$   
 $\langle \text{proof} \rangle$

**theorem** *andrew-hull-convex-hull-eq-iff*:

**fixes** *ps* :: *real point list*  
**shows**  $\text{convex hull set } (\text{andrew-hull } ps) = \text{convex hull set } ps \longleftrightarrow$   
 $\text{set } ps \subseteq \text{convex hull set } (\text{andrew-hull } ps)$   
 $\langle \text{proof} \rangle$

**theorem** *andrew-hull-correctI*:

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

*<proof>*

**theorem** *andrew-hull-correct-iff:*

**fixes** *ps* :: *real point list*

**shows** *convex-hull-correct ps (andrew-hull ps)  $\longleftrightarrow$*

*set ps  $\subseteq$  convex hull set (andrew-hull ps)*

*<proof>*

**lemma** *distinct-scan-push:*

**assumes** *distinct st* **and** *p  $\notin$  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  $x\ y\ z :: \text{real point}$   
assumes *sorted: sorted-wrt (<) (x # y # zs)*  
and *turns: chain-turns (x # y # zs)*  
and  $z: z \in \text{set } zs$   
shows  $0 < \text{cross } x\ y\ z$   
⟨proof⟩

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

**lemma** *uminus-less-point-iff [simp]*:  
fixes  $p\ q :: \text{real point}$   
shows  $-p < -q \longleftrightarrow q < p$

*<proof>*

**lemma** *cross-uminus* [*simp*]:  
 **fixes**  $p\ q\ r :: \text{real point}$   
 **shows**  $\text{cross } (-p) (-q) (-r) = \text{cross } p\ q\ r$   
 *<proof>*

**lemma** *left-turn-uminus* [*simp*]:  
 **fixes**  $p\ q\ r :: \text{real point}$   
 **shows**  $\text{left-turn } (-p) (-q) (-r) \longleftrightarrow \text{left-turn } p\ q\ r$   
 *<proof>*

**lemma** *stack-turns-map-uminus* [*simp*]:  
 **fixes**  $xs :: \text{real point list}$   
 **shows**  $\text{stack-turns } (\text{map } (\lambda p. -p) xs) \longleftrightarrow \text{stack-turns } xs$   
 *<proof>*

**lemma** *chain-turns-map-uminus* [*simp*]:  
 **fixes**  $xs :: \text{real point list}$   
 **shows**  $\text{chain-turns } (\text{map } (\lambda p. -p) xs) \longleftrightarrow \text{chain-turns } xs$   
 *<proof>*

**lemma** *sorted-wrt-less-map-uminus*:  
 **fixes**  $xs :: \text{real point list}$   
 **assumes**  $\text{sorted-wrt } (>) xs$   
 **shows**  $\text{sorted-wrt } (<) (\text{map } (\lambda p. -p) xs)$   
 *<proof>*

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

**lemma** *sorted-chain-edge-cross-nonneg-increasing*:  
 **fixes**  $xs :: \text{real point list}$   
 **assumes**  $\text{sorted: sorted-wrt } (<) xs$   
 **and**  $\text{turns: chain-turns } xs$   
 **and**  $\text{edge: } \text{Suc } i < \text{length } xs$   
 **and**  $q: q \in \text{set } xs$   
 **shows**  $0 \leq \text{cross } (xs ! i) (xs ! \text{Suc } i) q$   
 *<proof>*

**lemma** *sorted-chain-edge-cross-nonneg-decreasing*:  
 **fixes**  $xs :: \text{real point list}$

**assumes** *sorted*: *sorted-wrt* ( $>$ ) *xs*  
**and** *turns*: *chain-turns* *xs*  
**and** *edge*: *Suc* *i*  $<$  *length* *xs*  
**and** *q*:  $q \in \text{set } xs$   
**shows**  $0 \leq \text{cross } (xs \ ! \ i) (xs \ ! \ \text{Suc } i) \ q$   
*<proof>*

**lemma** *scan-push-nonempty* [*simp*]:  
*scan-push* *st*  $p \neq []$   
*<proof>*

**lemma** *scan-stack-nonempty*:  
 $ps \neq [] \implies \text{scan-stack } ps \neq []$   
*<proof>*

**lemma** *scan-chain-nonempty*:  
 $ps \neq [] \implies \text{scan-chain } ps \neq []$   
*<proof>*

**lemma** *lower-hull-nonempty*:  
 $ps \neq [] \implies \text{lower-hull } ps \neq []$   
*<proof>*

**lemma** *upper-hull-nonempty*:  
 $ps \neq [] \implies \text{upper-hull } ps \neq []$   
*<proof>*

## 1.2 Endpoints of the Scans

**lemma** *hd-scan-push* [*simp*]:  
 $\text{hd } (\text{scan-push } st \ p) = p$   
*<proof>*

**lemma** *last-scan-push*:  
**assumes**  $st \neq []$   
**shows**  $\text{last } (\text{scan-push } st \ p) = \text{last } st$   
*<proof>*

**lemma** *hd-scan-stack*:  
**assumes**  $ps \neq []$   
**shows**  $\text{hd } (\text{scan-stack } ps) = \text{last } ps$   
*<proof>*

**lemma** *last-scan-stack*:  
**assumes**  $ps \neq []$   
**shows**  $\text{last } (\text{scan-stack } ps) = \text{hd } ps$   
*<proof>*

**lemma** *hd-scan-chain*:

**assumes**  $ps \neq []$   
**shows**  $hd (scan-chain ps) = hd ps$   
(proof)

**lemma** *last-scan-chain*:  
**assumes**  $ps \neq []$   
**shows**  $last (scan-chain ps) = last ps$   
(proof)

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

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

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

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

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

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

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

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

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

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

**theorem** *set-andrew-hull*:  
 $set\ (andrew-hull\ ps) = set\ (lower-hull\ ps) \cup set\ (upper-hull\ ps)$   
 $\langle proof \rangle$

### 1.3 Support-Function Invariants

**definition** *support-value* :: *real point*  $\Rightarrow$  *real point*  $\Rightarrow$  *real*  
**where**  
 $support-value\ v\ p = fst\ v * fst\ p + snd\ v * snd\ p$

**lemma** *support-value-eq-inner*:  
 $support-value\ v\ p = inner\ v\ p$   
 $\langle proof \rangle$

**lemma** *support-value-edge-normal*:  
**fixes**  $a\ b\ x :: real\ point$   
**shows**  $support-value\ (snd\ b - snd\ a,\ fst\ a - fst\ b)\ x =$   
 $support-value\ (snd\ b - snd\ a,\ fst\ a - fst\ b)\ a - cross\ a\ b\ x$   
 $\langle proof \rangle$

**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**  $support-value\ v\ q \leq max\ (support-value\ v\ r)\ (support-value\ v\ p)$   
 $\langle proof \rangle$

**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**  $support-value\ v\ q \leq max\ (support-value\ v\ r)\ (support-value\ v\ p)$   
 $\langle proof \rangle$

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

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

**lemma** *scan-stack-strict-sorted-increasing:*

**fixes**  $xs :: \text{real point list}$   
**assumes**  $\text{sorted-wrt } (<) \ xs$   
**shows**  $\text{sorted-wrt } (<) (\text{rev } (\text{scan-stack } xs))$   
 $\langle \text{proof} \rangle$

**lemma** *scan-push-support-dominates-increasing-step:*

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

**lemma** *scan-push-support-dominates-increasing:*

**fixes**  $st :: \text{real point list}$   
**shows**  $\text{sorted-wrt } (<) (\text{rev } st) \implies$   
 $(\bigwedge q. q \in \text{set } st \implies q < p) \implies$   
 $\text{snd } v < 0 \implies$   
 $x \in \text{insert } p (\text{set } st) \implies$   
 $\exists y \in \text{set } (\text{scan-push } st \ p). \text{support-value } v \ x \leq \text{support-value } v \ y$   
 $\langle \text{proof} \rangle$

**lemma** *scan-stack-support-dominates-increasing:*

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

**lemma** *scan-chain-support-dominates-increasing:*

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

**lemma** *sorted-wrt-less-sorted-unique* [*simp*]:

$\text{sorted-wrt } (<) (\text{sorted-unique } (xs :: 'a::\text{linorder list}))$   
 $\langle \text{proof} \rangle$

**lemma** *lower-hull-supports-negative:*

**assumes**  $snd\ v < 0$  **and**  $x \in set\ ps$

**shows**  $\exists y \in set\ (lower-hull\ ps). support-value\ v\ x \leq support-value\ v\ y$   
(*proof*)

**lemma** *scan-push-strict-sorted-decreasing:*

**fixes**  $st :: real\ point\ list$

**assumes** *sorted:*  $sorted-wrt\ (>)\ (rev\ st)$

**and** *above:*  $\bigwedge q. q \in set\ st \implies p < q$

**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:*

$cross\ r\ q\ p \leq 0 \implies$

$(\bigwedge x. x \in insert\ p\ (set\ (r\ \# \ st))) \implies$

$\exists y \in set\ (scan-push\ (r\ \# \ st)\ p). support-value\ v\ x \leq support-value\ v\ y$

**and** *sorted:*  $sorted-wrt\ (>)\ (rev\ (q\ \# \ r\ \# \ st))$

**and** *above:*  $\bigwedge z. z \in set\ (q\ \# \ r\ \# \ st) \implies p < z$

**and** *pos:*  $0 < snd\ v$

**and** *x-in:*  $x \in insert\ p\ (set\ (q\ \# \ r\ \# \ st))$

**shows**  $\exists y \in set\ (scan-push\ (q\ \# \ r\ \# \ st)\ p). support-value\ v\ x \leq support-value\ v\ y$   
(*proof*)

**lemma** *scan-push-support-dominates-decreasing:*

**fixes**  $st :: real\ point\ list$

**shows** *sorted-wrt*  $(>)\ (rev\ st) \implies$

$(\bigwedge q. q \in set\ st \implies p < q) \implies$

$0 < snd\ v \implies$

$x \in insert\ p\ (set\ st) \implies$

$\exists y \in set\ (scan-push\ st\ p). support-value\ v\ x \leq support-value\ v\ y$   
(*proof*)

**lemma** *scan-stack-support-dominates-decreasing:*

**fixes**  $xs :: real\ point\ list$

**assumes** *sorted:*  $sorted-wrt\ (>)\ xs$

**and** *pos:*  $0 < snd\ v$

**and**  $x \in set\ xs$

**shows**  $\exists y \in set\ (scan-stack\ xs). support-value\ v\ x \leq support-value\ v\ y$   
(*proof*)

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

**lemma** *sorted-wrt-greater-rev-sorted-unique* [*simp*]:  
*sorted-wrt* ( $>$ ) ( $\text{rev } (\text{sorted-unique } (xs :: 'a::\text{linorder list})))$ )  
 $\langle \text{proof} \rangle$

**lemma** *upper-hull-supports-positive*:  
**assumes**  $0 < \text{snd } v$  **and**  $x \in \text{set } ps$   
**shows**  $\exists y \in \text{set } (\text{upper-hull } ps). \text{support-value } v \ x \leq \text{support-value } v \ y$   
 $\langle \text{proof} \rangle$

**theorem** *lower-hull-sorted*:  
**fixes**  $ps :: \text{real point list}$   
**shows** *sorted-wrt* ( $<$ ) (*lower-hull*  $ps$ )  
 $\langle \text{proof} \rangle$

**theorem** *upper-hull-sorted*:  
**fixes**  $ps :: \text{real point list}$   
**shows** *sorted-wrt* ( $>$ ) (*upper-hull*  $ps$ )  
 $\langle \text{proof} \rangle$

**lemma** *fst-hd-sorted-unique-le*:  
**fixes**  $x :: \text{real point}$   
**assumes**  $x \in \text{set } ps$   
**shows**  $\text{fst } (\text{hd } (\text{sorted-unique } ps)) \leq \text{fst } x$   
 $\langle \text{proof} \rangle$

**lemma** *fst-le-last-sorted-unique*:  
**fixes**  $x :: \text{real point}$   
**assumes**  $x \in \text{set } ps$   
**shows**  $\text{fst } x \leq \text{fst } (\text{last } (\text{sorted-unique } ps))$   
 $\langle \text{proof} \rangle$

**lemma** *hd-sorted-unique-less*:  
**fixes**  $x :: \text{real point}$   
**assumes**  $x: x \in \text{set } ps$   
**and**  $ne: x \neq \text{hd } (\text{sorted-unique } ps)$   
**shows**  $\text{hd } (\text{sorted-unique } ps) < x$   
 $\langle \text{proof} \rangle$

**lemma** *less-last-sorted-unique*:  
**fixes**  $x :: \text{real point}$   
**assumes**  $x: x \in \text{set } ps$   
**and**  $ne: x \neq \text{last } (\text{sorted-unique } ps)$   
**shows**  $x < \text{last } (\text{sorted-unique } ps)$

*<proof>*

**lemma** *lex-min-strict-support*:

**fixes**  $p :: \text{real point}$

**assumes**  $\text{fin}: \text{finite } S$

**and**  $\text{lex}: \bigwedge q. q \in S - \{p\} \implies p < q$

**shows**  $\exists v. \forall q \in S - \{p\}. \text{inner } v \ q < \text{inner } v \ p$

*<proof>*

**lemma** *lex-max-strict-support*:

**fixes**  $p :: \text{real point}$

**assumes**  $\text{fin}: \text{finite } S$

**and**  $\text{lex}: \bigwedge q. q \in S - \{p\} \implies q < p$

**shows**  $\exists v. \forall q \in S - \{p\}. \text{inner } v \ q < \text{inner } v \ p$

*<proof>*

**lemma** *lower-hull-edge-cross-nonneg-input-if-fst-less*:

**fixes**  $ps :: \text{real point list}$

**defines**  $L \equiv \text{lower-hull } ps$

**assumes**  $\text{edge}: \text{Suc } i < \text{length } L$

**and**  $\text{fst-less}: \text{fst } (L \ ! \ i) < \text{fst } (L \ ! \ \text{Suc } i)$

**and**  $x: x \in \text{set } ps$

**shows**  $0 \leq \text{cross } (L \ ! \ i) \ (L \ ! \ \text{Suc } i) \ x$

*<proof>*

**lemma** *upper-hull-edge-cross-nonneg-input-if-fst-greater*:

**fixes**  $ps :: \text{real point list}$

**defines**  $U \equiv \text{upper-hull } ps$

**assumes**  $\text{edge}: \text{Suc } i < \text{length } U$

**and**  $\text{fst-greater}: \text{fst } (U \ ! \ \text{Suc } i) < \text{fst } (U \ ! \ i)$

**and**  $x: x \in \text{set } ps$

**shows**  $0 \leq \text{cross } (U \ ! \ i) \ (U \ ! \ \text{Suc } i) \ x$

*<proof>*

**lemma** *lower-hull-vertical-edge-last*:

**fixes**  $ps :: \text{real point list}$

**defines**  $L \equiv \text{lower-hull } ps$

**assumes**  $\text{edge}: \text{Suc } i < \text{length } L$

**and**  $\text{same-x}: \text{fst } (L \ ! \ i) = \text{fst } (L \ ! \ \text{Suc } i)$

**shows**  $\text{Suc } i = \text{length } L - 1$

*<proof>*

**lemma** *upper-hull-vertical-edge-last*:

**fixes**  $ps :: \text{real point list}$

**defines**  $U \equiv \text{upper-hull } ps$

**assumes**  $\text{edge}: \text{Suc } i < \text{length } U$

**and**  $\text{same-x}: \text{fst } (U \ ! \ i) = \text{fst } (U \ ! \ \text{Suc } i)$

**shows**  $\text{Suc } i = \text{length } U - 1$

*<proof>*

**lemma** *lower-hull-edge-cross-nonneg-input:*

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

**lemma** *upper-hull-edge-cross-nonneg-input:*

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

**lemma** *strict-support-from-two-edges:*

**fixes**  $a p c :: \text{real point}$   
**assumes**  $\text{left}: \bigwedge q. q \in S \implies 0 \leq \text{cross } a p q$   
**and**  $\text{right}: \bigwedge q. q \in S \implies 0 \leq \text{cross } p c q$   
**and**  $\text{noncol}: \text{cross } a p c \neq 0$   
**shows**  $\exists v. \forall q \in S - \{p\}. \text{inner } v q < \text{inner } v p$   
(*proof*)

**lemma** *lower-hull-interior-strict-support:*

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

**lemma** *upper-hull-interior-strict-support:*

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

**lemma** *lower-hull-interior-strict-support-if-fst-less:*

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

*<proof>*

**lemma** *hd-sorted-unique-mem-andrew-hull:*

**assumes**  $ps \neq []$

**shows**  $hd (sorted\_unique\ ps) \in set (andrew\_hull\ ps)$

*<proof>*

**lemma** *last-sorted-unique-mem-andrew-hull:*

**assumes**  $ps \neq []$

**shows**  $last (sorted\_unique\ ps) \in set (andrew\_hull\ ps)$

*<proof>*

**lemma** *andrew-hull-supports-horizontal:*

**assumes**  $x: x \in set\ ps$  **and** *horizontal:*  $snd\ v = 0$

**shows**  $\exists y \in set (andrew\_hull\ ps). support\_value\ v\ x \leq support\_value\ v\ y$

*<proof>*

**lemma** *andrew-hull-supports:*

**assumes**  $x: x \in set\ ps$

**shows**  $\exists y \in set (andrew\_hull\ ps). support\_value\ v\ x \leq support\_value\ v\ y$

*<proof>*

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

**theorem** *andrew-hull-covers-input:*

**fixes**  $ps :: real\ point\ list$

**shows**  $set\ ps \subseteq convex\ hull\ set (andrew\_hull\ ps)$

*<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**  $\exists v. \forall q \in set (andrew\_hull\ ps) - \{p\}. inner\ v\ q < inner\ v\ p$

*<proof>*

**theorem** *andrew-hull-delete-changes-convex-hull:*

**fixes**  $ps :: real\ point\ list$

**assumes**  $p: p \in set (andrew\_hull\ ps)$

**shows**  $convex\ hull (set (andrew\_hull\ ps) - \{p\}) \neq convex\ hull\ set (andrew\_hull\ ps)$

*<proof>*

**theorem** *andrew-hull-irredundant:*

**fixes**  $ps :: real\ point\ list$

**shows** *convex-hull-irredundant* (*andrew-hull ps*)  
<proof>

**lemma** *length-scan-push-le*:  
 $length\ (scan-push\ st\ p) \leq Suc\ (length\ st)$   
<proof>

**lemma** *length-scan-stack-le*:  
 $length\ (scan-stack\ ps) \leq length\ ps$   
<proof>

**lemma** *length-scan-chain-le*:  
 $length\ (scan-chain\ ps) \leq length\ ps$   
<proof>

**lemma** *length-lower-hull-le*:  
 $length\ (lower-hull\ ps) \leq card\ (set\ ps)$   
<proof>

**lemma** *length-upper-hull-le*:  
 $length\ (upper-hull\ ps) \leq card\ (set\ ps)$   
<proof>

**theorem** *length-andrew-hull-le-twice-card*:  
 $length\ (andrew-hull\ ps) \leq 2 * card\ (set\ ps)$   
<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**  $card\ (set\ ps) \geq 2$   
**shows**  $andrew-hull\ ps = butlast\ (lower-hull\ ps) @ butlast\ (upper-hull\ ps)$   
<proof>

**theorem** *andrew-hull-ordered-chains*:  
**fixes**  $ps :: real\ point\ list$   
**assumes**  $2 \leq card\ (set\ ps)$   
**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**  $distinct\ (andrew-hull\ ps) \longleftrightarrow$

```

    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**

$\text{square-real} =$   
[[0, 0), (1, 0), (1, 1), (0, 1), (0, 0), (1, 1)]

**definition**  $\text{square-real-vertices} :: (\text{real} \times \text{real}) \text{ list}$

**where**

$\text{square-real-vertices} = [(0, 0), (1, 0), (1, 1), (0, 1)]$

**lemma**  $\text{square-real-hull}$ :

$\text{andrew-hull square-real} = \text{square-real-vertices}$   
<proof>

**lemma**  $\text{square-real-correct}$ :

$\text{convex-hull-correct square-real (andrew-hull square-real)}$   
<proof>

**lemma**  $\text{square-real-irredundant}$ :

$\text{convex-hull-irredundant (andrew-hull square-real)}$   
<proof>

**lemma**  $\text{square-real-convex-hull}$ :

$\text{convex hull set (andrew-hull square-real)} = \text{convex hull set square-real}$   
<proof>

**definition**  $\text{collinear-real} :: (\text{real} \times \text{real}) \text{ list}$

**where**

$\text{collinear-real} = [(2, 0), (0, 0), (1, 0), (3, 0), (1, 0)]$

**definition**  $\text{collinear-real-vertices} :: (\text{real} \times \text{real}) \text{ list}$

**where**

$\text{collinear-real-vertices} = [(0, 0), (3, 0)]$

**lemma**  $\text{collinear-real-hull}$ :

$\text{andrew-hull collinear-real} = \text{collinear-real-vertices}$   
<proof>

**lemma**  $\text{one-zero-in-collinear-real-hull}$ :

$(1::\text{real}, 0) \in \text{convex hull set collinear-real-vertices}$   
<proof>

**lemma**  $\text{two-zero-in-collinear-real-hull}$ :

$(2::\text{real}, 0) \in \text{convex hull set collinear-real-vertices}$   
<proof>

**lemma**  $\text{collinear-real-correct}$ :

$\text{convex-hull-correct collinear-real (andrew-hull collinear-real)}$   
<proof>

**lemma**  $\text{collinear-real-irredundant}$ :

*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.