Mechanical Theorem Proving in Computational Geometry

Laura I. Meikle and Jacques D. Fleuriot

School of Informatics, University of Edinburgh, Appleton Tower, Crichton Street, Edinburgh, EH8 9LE, UK
{lauram,jdf}@dai.ed.ac.uk

Abstract. Algorithms for solving geometric problems are widely used in many scientific disciplines. Applications range from computer vision and robotics to molecular biology and astrophysics. Proving the correctness and efficiency of these algorithms is vital in order to boost confidence in them. By specifying the algorithms formally in a theorem prover like Isabelle, it is hoped that rigorous proofs showing their correctness will be obtained. This paper outlines our current framework for reasoning about geometric algorithms in Isabelle. It focuses on our case study of the convex hull problem and shows how Hoare logic can be used to prove the correctness of such algorithms.

1 Introduction

Computational geometry is the branch of computer science that studies algorithms for solving geometric problems [15]. It has applications in, among other fields, computer graphics, robotics, molecular biology, astrophysics and statistics. Verifying that these algorithms do indeed produce the correct output is important, particularly where they are used in mission-critical instances. Formal verification by computer would boost confidence in the algorithms and would also provide a valuable insight into the how they work. However, little has been achieved in this field to date. Our aim is to build a framework for reasoning about geometric algorithms in the theorem prover Isabelle.

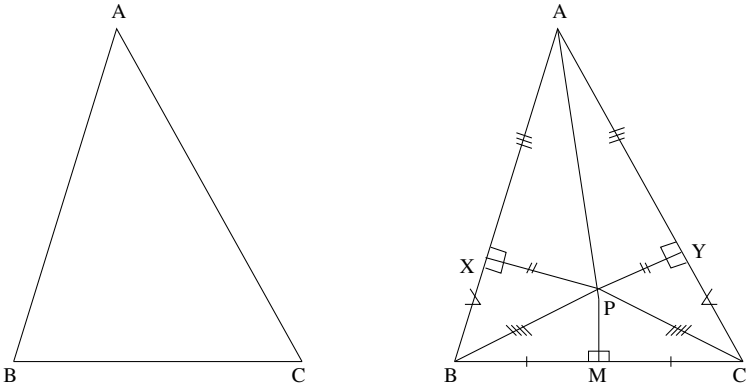
As convex hulls are used widely in computational geometry, we look at an algorithm for computing them in two dimensions known as Graham's Scan [6]. We chose to carry out our formal verification in Hoare logic [8], as it provides a mechanisable formal system on which one can reason about imperative programs.

The next section outlines a few of the issues with geometric reasoning. After that we introduce the theorem prover Isabelle and describe the logic and calculus we use. Next we focus on our case study, Graham's Scan, and in particular the formal specification and verification. Finally, we conclude by discussing some related work and our goals for the future.

2 All Triangles Are Equilateral

Are all triangles equilateral? The following diagram and steps of reasoning give quite a convincing argument in favour of this statement: take any arbitrary

triangle $\triangle ABC$ and let the perpendicular bisector of BC and the internal angle bisector of $\angle A$ meet at some point P. If we then construct points X and Y such that $PX \perp AB$ and $PY \perp AC$, then $AX = AY$ (by the angle-side-angle congruence test). Using the hypotenuse-side congruence test it can be shown that $\triangle PXB \cong \triangle PYC$, since $PX = PY$, $PB = PC$, and $\angle PXB$ is a right angle. Thus, $XB = YC$. Therefore, $AX + XB = AY + YC$. This means $AB = AC$. If we now rotate the triangle and apply the same argument on sides AC and BC, we get the conclusion that $AC = BC$. Therefore, all 3 sides of the triangle must be equal and we have proved that all triangles are equilateral.



What is the flaw in this proof? Our intuition tells us the proof is wrong, but if the result was not so obviously false we might be convinced by this spurious proof. Intuition is an important sanity check for proofs that appear logically sound.

On the flip side, if our intuition agrees with a result it may allow us to overlook missing steps or even flaws in proofs. Even Hilbert's rigorous axiomatisation of Euclidean geometry suffered from this human tendency. He argued that his proofs were free of intuition and only required the rules of logic and formal reasoning [7]. However, after formalising his work in Isabelle, we noted that Hilbert's proofs do in fact rely on intuition, through the use of diagrams and the exclusion of certain case splits [10]. We believe that if the commonly accepted "formalisation" of Euclidean geometry relies so unwittingly on intuition, then our confidence in geometric algorithms is suspect.

Diagrams in particular appeal to our intuition. They give a quasi-formal reassurance for geometric reasoning and geometric algorithms, but they can be misleading. Even if our hypotheses are tested through the use of diagrams, we still do not have complete validation of our results. In fact diagrams can be a minefield for making mistakes with undue confidence. The diagrammatic proof that all triangles are equilateral illustrates this. Although the design and verification of geometric algorithms can be aided through diagrams, clearly more rigorous reasoning is required.

A common approach to verification is simulating the algorithm on certain examples. Typically these examples are either human-selected or randomly generated. Although this method is useful, it is often subject to developer bias and

can rarely verify every possible case. We believe formal verification is the surest way to have confidence in theorems, proofs, and algorithms. Furthermore, our understanding of a geometric algorithm is increased by formal verification, as it reveals inconsistencies, ambiguities and incompleteness that might otherwise go undetected.

3 The Theorem Prover Isabelle

Isabelle is a generic theorem prover, written in ML, which can be used as a specification and verification system [13]. There are a number of object-level logics that Isabelle allows the user to encode particular problems. Of specific interest to this work is the capacity for proofs in higher order logic (HOL). This provides a framework powerful enough to reason about algorithms and sophisticated mathematical notions. Isabelle/HOL, is influenced by Gordon's HOL theorem prover [5] which itself originates from a classic paper by Church [3]. It provides an extensive library of theories, a powerful simplifier which accomplishes conditional and unconditional rewritings and several generic proof methods, including `blast` and `auto` which attempts to prove all subgoals by a combination of simplification and classical reasoning. These tools greatly help mechanisation.

In particular, the development of Floyd-Hoare logic within Isabelle/HOL is highly relevant [11]. With this logic, the formal specifications of geometric algorithms can closely resemble their implementations.

4 Floyd-Hoare Logic

Floyd-Hoare logic is widely viewed as a way of reasoning mathematically about imperative programs [8]. The logic can not only allow verification of programs but can also aid their constructions from their specification. Hence, it should be beneficial to reasoning about geometric algorithms in a sound and rigorous manner.

Hoare introduced a notation, called a *partial correctness specification*, for specifying what a program does: $\{P\}\ C\ \{Q\}$ is said to be true if whenever C is executed in a state satisfying P and C terminates, then the state in which C terminates satisfies Q. Total correctness is what ultimately needs to be proved when verifying a program. Informally *total correctness* = *termination* + *partial correctness.*

Although Hoare developed this logic as a means of reasoning about imperative programs, he never fully mechanised it. This was first done by Gordon [4] in the theorem prover HOL using an embedding of an annotated WHILE language in higher order logic and a verification conditions generator. In Gordon's approach a program can be annotated, to show the relationships between the variables, by inserting statements called assertions. These assertions express conditions that are meant to hold at various intermediate points.

In particular, in order to formally verify programs involving loops, the partial correctness specification is annotated with mathematical statements known as

invariants. An invariant R must satisfy the following conditions: it must hold initially, it must establish the result with the negated test and the body of the program must leave it unchanged. In Gordon's system, a set of purely mathematical statements called *verification conditions* (or VCs) are then generated. A program is partially correct if the following VCs can be proven:

1. $P \rightarrow R$
2. $R \wedge$ Loop Test $\rightarrow$ body of loop preserves R
3. $R \wedge$ Negated Loop Test $\rightarrow Q$

In Isabelle, work on Hoare logic has been formalised by Nipkow [11] and will be of interest to us. In this work, the notation for representing a theorem about an imperative program with a while loop is of the general form:

```
theorem
 "|- .{ P }.
     variable assignments;;
     WHILE Loop test
     INV .{ Loop invariant }.
     DO
        program
     OD
     .{ Q }."
```

5 Geometric Preliminaries

Our mathematical framework for reasoning formally about geometric algorithms builds upon a 2D real vector theory developed in Isabelle. In this theory a real vector is a pair of real numbers, represented by (a, b). This can be interpreted in two ways; either as the point with coordinates (a, b) or as the position vector $\overrightarrow{(0,0)(a,b)}$. The vector theory defines the notions of vector addition, subtraction, dot product and scalar product. It also formalises the outer product ($\times$), defined in terms of the coordinates of the points A and B:

$$A \times B \equiv A_x * B_y \text{ - } A_y * B_x$$

Using the outer product definition, it is possible to capture the notion of the signed area of three points A, B and C:

$$\texttt{area}\ A\ B\ C \equiv (B \text{ - } A) \times (C \text{ - } A)$$

Our theory uses signed areas to identify where three points lie in relation to each other. The property of collinearity can be captured as follows:

$$\texttt{coll}\ A\ B\ C \equiv \text{area}\ A\ B\ C = 0$$

Another property which can be formalised is the notion of a *left turn*. If a point C lies to the left of the directed line from A to B, we write:

$$\texttt{Left\_turn}\ A\ B\ C \equiv \text{area}\ A\ B\ C > 0$$

As shall be seen in the following sections, testing for left turns is very useful in formalising and constructing convex hulls. Another useful property is whether a point lies between two others. We say that B lies between A and C if the following holds:

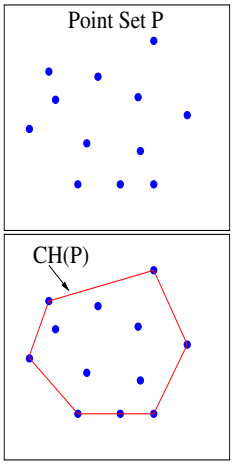
$$\begin{aligned} B\ \texttt{isBetween}\ A\ C \equiv \texttt{coll}\ A\ B\ C \quad \wedge \quad A \neq C \quad \wedge \\ \big(\forall\, \text{D.} \quad \texttt{area}\ A\ C\ D \neq 0 \quad \longrightarrow \\ (0 < (\texttt{area}\ A\ B\ D \ / \ \texttt{area}\ A\ C\ D) < 1)\ \big) \end{aligned}$$

Note that we do not need to state explicitly that all the points are distinct because the *area* tests would not hold simultaneously if the points were identical.

6 Defining Convex Hulls

In this section we describe what the convex hull of a set of points is and show how we formally define it in the theorem prover Isabelle.

6.1 What is a Convex Hull?



Many definitions exist for *convex hulls*. Intuitively, one may think of a set of points P in 2D as being nails sticking upwards from a board. Now imagine stretching a rubber band around them and letting go so its length is minimised. The area enclosed by the rubber band is known as the convex hull of P. The convex hull of P can also be described as the smallest convex set containing P. Despite the simplicity of this definition, it is not conducive to algorithm development as it is not constructive. For our formal specification we were inspired by a definition Knuth gave in his book *Axioms and Hulls* [9]. This is described in the next section.

6.2 Formal Specification of Convex Hulls

In his book, Knuth describes an orientation predicate which is equivalent to our previously defined `Left_turn`. Using this, he defines a counter-clockwise (CC) system as one which satisfies five axioms that capture the minimal properties of the orientation predicate:

1 (cyclic symmetry). $\texttt{Left\_turn}\ p\ q\ r \implies \texttt{Left\_turn}\ q\ r\ p$

2 (antisymmetry). $\texttt{Left\_turn}\ p\ q\ r \implies \neg\ \texttt{Left\_turn}\ p\ r\ q$

3 (nondegeneracy). $p{\neq}q \wedge p{\neq}r \wedge q{\neq}r \Longrightarrow$
$\texttt{Left\_turn}\ p\ q\ r \vee \texttt{Left\_turn}\ p\ r\ q$

4 (interiority). $\texttt{Left\_turn}\ t\ q\ r \wedge \texttt{Left\_turn}\ p\ t\ r \wedge$
$\texttt{Left\_turn}\ p\ q\ t \Longrightarrow \texttt{Left\_turn}\ p\ q\ r$

5 (transitivity). $\texttt{Left\_turn}\ t\ s\ p \wedge \texttt{Left\_turn}\ t\ s\ q \wedge$
$\texttt{Left\_turn}\ t\ s\ r \wedge \texttt{Left\_turn}\ t\ p\ q \wedge$
$\texttt{Left\_turn}\ t\ q\ r \Longrightarrow \texttt{Left\_turn}\ t\ p\ r$

Although our system bears much resemblance to Knuth's, we have not adopted his axiomatic approach. We have followed Isabelle's methodology of maintaining consistency by developing new theories on top of old ones through conservative extensions only; in our case we have built upon our theory of vectors.

One drawback of Knuth's CC system for our purposes is that it disallows collinear points. This leads to many elegant results in his framework, but for real-world applications this restriction is not practical. In our formalisation, which permits collinear points, four of Knuth's axioms remain true and have been proven from first principles in Isabelle. The remaining axiom, Knuth's third is altered and proven in our system with the obvious modification as the following theorem:

$$p{\neq}q \wedge p{\neq}r \wedge q{\neq}r \Longrightarrow \texttt{Left\_turn}\ p\ q\ r \vee \texttt{Left\_turn}\ p\ r\ q \vee \texttt{coll}\ p\ q\ r$$

Knuth presents an alternate version of axiom 5:

5b. $\texttt{Left\_turn}\ s\ t\ p \wedge \texttt{Left\_turn}\ s\ t\ q \wedge \texttt{Left\_turn}\ s\ t\ r \wedge$
$\texttt{Left\_turn}\ t\ p\ q \wedge \texttt{Left\_turn}\ t\ q\ r \Longrightarrow \texttt{Left\_turn}\ t\ p\ r$

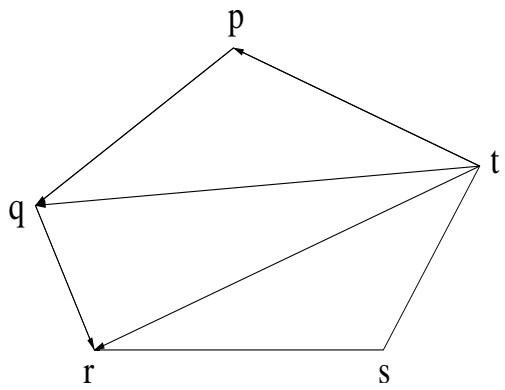


Fig. 1. Knuth's Axiom 5b

Axiom 5b is illuistrated in (see Figure 1) and has also been proven from first principles in Isabelle. Our mechanical proof followed the outline Knuth sketched

using three applications of Axiom 5. However, allowing collinear points in our system dramatically increased the number of cases which needed to be considered. In fact we had 13 additional configurations of the points to reason about.

Knuth then describes the convex hull C of a set of points P, which satisfy his CC system, as the set of all consecutive vertices ts of C such that `Left_turn` t s p holds for all $p \notin \{s, t\}$ (see Figure 2). Clearly this definition only holds when we are traversing the hull in a counter-clockwise direction.

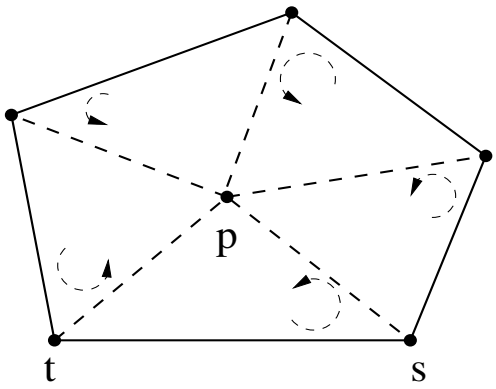


Fig. 2. Formal description of convex hull

Our formalisation of a convex hull also has to allow the possibility that any point in P could lie between two consecutive vertices. In Isabelle, we check that C is the convex hull of the points P using the infix predicate `isConvexHull`.

```
C isConvexHull P ≡ ¬all_collinear P ∧ distinct C ∧ set C ⊆ set P ∧
                   (∀ n < length P. ∀ i < (length C - 1).
                    (Left_turn C_{i+1} C_i P_n ∨
                     P_n mem [C_{i+1}, C_i] ∨
                     P_n isBetween C_{i+1} C_i) ∧
                    (Left_turn (hd C) (last C) P_n ∨
                     P_n mem [hd C, last C] ∨
                     P_n isBetween (hd C) (last C)))
```

Note that we have represented the point sets C and P using lists and the nth member of a list C is denoted by C_n. We assume that the points in C are ordered clockwise and the predicate `isConvexHull` ensures that every point in C is distinct and belongs to the original, non-collinear point set P. We then check that every point in P either makes a left turn with respect to consecutive vertices in C or is a vertex of the hull or lies between consecutive vertices in C. The last case in the definition simply closes the convex polygon and ensures that the first and last vertices in C satisfy the same tests carried on on consecutive vertices.

In the next section we describe Graham's Scan algorithm for computing convex hulls, which was chosen for formalisation in Isabelle.

7 Graham's Scan Algorithm

Computing the convex hull of a set of points is a problem which has been greatly studied. As a result there exists an abundance of algorithms. Graham's Scan is just one which computes 2D convex hulls. It is described in the following section, followed by its formalisation in Isabelle's Hoare logic.

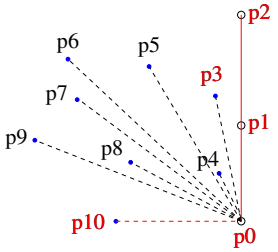
7.1 How it Works

Graham's Scan uses a method known as rotational sweep and solves the problem by maintaining a stack C of candidate points. Each point of the input set P is pushed once onto the stack, and the points that are not vertices of the convex hull are eventually popped. When the algorithm terminates, stack C contains exactly the vertices of the hull, in counterclockwise order of their appearance on the boundary (see pseudo-code below, from [12]).

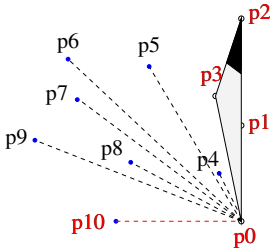
GRAHAM'S SCAN ALGORITHM

```
Find rightmost lowest point; label it P_0.
Sort all other points angularly about P_0,
     break ties in favour of closeness to P_0;
     label P_1,...,P_{n-1}.
Stack C=(P_{n-1},P_0)=(P_{t-1},P_t); t indexes top.
i = 1
while i < n do
    if P_i is strictly left of (P_{t-1},P_t)
          then Push(C,i) and set i ← i + 1
          else Pop(C)
```

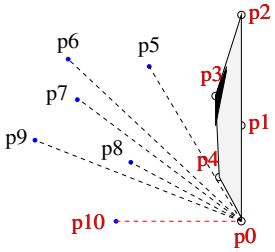
Notice that point P_{n-1} ends up twice on C, so a final pop is required in the algorithm. The following diagrams illustrate some of the behaviour of Graham's Scan. The hollow points are the vertices of the hull (shaded region) C at each stage.



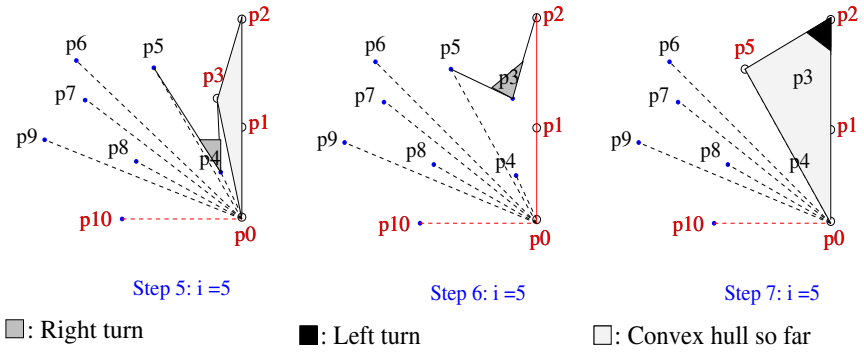
Step 2: i =2



Step 3: i =3



Step 4: i =4



7.2 Formal Specification of Graham's Scan

This section describes how we formally specify Graham's Scan using Hoare logic in Isabelle. We use lists to represent the point sets P and C, as Isabelle already has this data structure defined and many of its properties proved. Due to the way C is updated, the vertices of the final hull are returned in clockwise order, not counter-clockwise like the pseudo-code of Section 7.1. The theorem that needs to be proven is specified as follows:

```
theorem
 "⊢ .{ ordered P & 3 ≤ length P & distinct P & ¬all_collinear P }.
     'i := 0;;
     'C := [hd P, last P];;
     WHILE 'i < length P
     INV .{ Loop invariant }.
     DO
        IF Left_turn C₁ C₀ P'ᵢ
        THEN 'C := P'ᵢ # 'C;;
             'i := 'i+1
        ELSE 'C := tl 'C
        FI
     OD
     .{(butlast 'C) isConvexHull P }."
```

We have not shown the loop invariant of Graham's Scan here as it is made up of numerous components which will be explained in Section 8. Note that the variables C and i are written with $'$ before them. This is to signify that they change as the program evolves. Also note that the post-condition has to check that all the points, excluding the last one in C (hence the use of the `butlast` function), are indeed vertices of the convex hull of P. We exclude the last point of C as it would appear twice in the list otherwise.

The last three preconditions, in the statement of Graham's Scan in Isabelle, prevent us from trying to prove the algorithm is correct for degenerate cases: we must start with at least three points, they must all be distinct and the points cannot all lie in the same line. The preconditions also state that the point set

P must be *ordered*. Next, we review briefly how this notion is mechanised in Isabelle.

7.3 Ordering Points

In our formalisation of an ordered point set, we refer to a predicate `Lowest_pt`, which represents the point with the minimum y-coordinate in P. Similarly to the algorithm, we take the rightmost point in case of a tie. Instead of using trigonometric functions to sort the points by polar angle, we decided to take advantage of the properties of signed areas. We can say the points in P are ordered if:

```
ordered P ≡ P₀ = Lowest_pt P ∧
            ∀m<length P. ∀n<length P. 0 < m ∧ m < n ⟶
                 before P₀ Pₘ Pₙ
```

where the predicate `before` is defined as:

```
before L u v ≡ (u isBetween L v) ∨ (Left_turn L u v)
```

This can be read as: "if L is the lowest point in P and u and v are any other two points in P, then u comes before v if it lies between L and v or if the point v lies to the left of the directed line from L to u".

Many properties of the ordered predicate have been proven in Isabelle, one of which is:

```
ordered P ∧ distinct P ∧ P ≠ [] ∧
A mem (take i P) ∧ i < length P ∧ A ≠ hd P
                                ⟹ before (hd P) A Pᵢ
```

where (take i P) represents a list of the first i elements in P.

8 Loop Invariants for Graham's Scan

Formulating the correct loop invariant for Graham's Scan is the most difficult task in this particular verification problem. This is because it requires identifying multiple components, which are sufficient to give the desired result on termination, and which are all true each time the loop is iterated. Discovering the correct components is an iterative process, where we first start with the facts we think are necessary in the proof. We then attempt to prove the VCs using these components. If our attempt is unsuccessful it will usually suggest additional information which should be added to the loop invariant.

Our first attempt at formalising the loop invariant of Graham's Scan was drawn from the written proof given by O'Rourke [12]. This written proof states that there must always be two points on the stack in order to determine whether a new point is a vertex of the hull. More specifically, the two points which the stack is initialised with, (`hd` P) and (`last` P), must be vertices of the final hull. In our formalisation, we have the corresponding loop invariant components:

1. $2 \leq \texttt{length}\ C$
2. $\texttt{last}\ C = \texttt{last}\ P$
3. $\texttt{last}\ (\texttt{butlast}\ C) = \texttt{hd}\ P$

The written proof assumes that whenever multiple points are collinear with P_0, all but the furthest are removed in a pre-processing step. With this assumption, and the ordering by polar angle, P_1 must be on the hull. Since the algorithm does not strictly require the removal of these collinear points, and because this removal in fact complicates the verification process, we have not made this assumption. As a result, we introduce a new predicate called `furthest_fst_pol_ang`. This predicate is defined as:

$$\begin{aligned}\texttt{furthest\_fst\_pol\_ang P f} \equiv\ & \texttt{ordered P} \wedge 2 \leq \texttt{length P} \wedge \\ & \texttt{f} < \texttt{length P} \wedge \texttt{collinear P}_0\ \texttt{P}_1\ \texttt{P}_f \wedge \\ & \neg\texttt{collinear P}_0\ \texttt{P}_1\ \texttt{P}_{f+1}\end{aligned}$$

In words, the point P_f is the *furthest first polar angle* point in P, if P is ordered and P_f is the point furthest from P_0 on the directed line from P_0 through P_1.[1] In the mechanical proof, we also need to keep track of when this vertex appears on the hull. The written proof's claim that P_1 is on the hull becomes, in our system, the following component:

4. $\forall f.\ (\texttt{furthest\_fst\_pol\_ang}\ P\ f \wedge f < i) \longrightarrow P_f\ \texttt{mem}\ C$

The key step in the algorithm, according to the written proof, is the *strict* left turn test, where C_0 is removed from C if it is collinear with P_i (the point under consideration) and C_1. The written proof neglects the important case where C_0 is removed because of a *right* turn with respect to P_i. Our corresponding loop invariant component is:

5. $\forall k\ j.\ (\neg\ P_k\ \texttt{mem butlast}\ C \wedge k < i \wedge j < \texttt{length}\ C\ \texttt{-}\ 2) \longrightarrow$
   $(\texttt{Left\_turn}\ C_{j+1}\ C_j\ P_k \vee P_k\ \texttt{isBetween}\ C_{j+1}\ C_j)$

This component notes that if we have examined a point P_k which does not belong to the list of vertices C, then it must have been removed from C at some time. Thus it will not be a vertex of the final hull, and must therefore lie to the left of all directed lines from C_{j+1} through C_j for all $j < \texttt{length}\ C - 2$ or between two adjacent vertices in the hull so far.

With only these five components drawn from the written proof, it is impossible to mechanically verify Graham's Scan. The following facts are also needed:

6. $i \leq \texttt{length}\ P$
7. $1 \leq i$
8. $3 \leq \texttt{length}\ P$
9. $\neg\texttt{all\_collinear}\ P$
10. $\texttt{ordered}\ P$

[1] In many cases P_f will be P_1. The predicate is introduced to simplify the case where $P_0, P_1, ..., P_f$ are collinear and $1 < f$.

11. $\texttt{distinct}\ P$
12. $\texttt{distinct}\ (\texttt{butlast}\ C)$
13. $i = \texttt{length}\ P \longrightarrow \texttt{last}\ P = \texttt{hd}\ C$
14. $\forall\ V.\ V\ \texttt{mem}\ (\texttt{butlast}\ C) \longrightarrow V\ \texttt{mem}\ (\texttt{take}\ (i + 1)\ P)$
15. $\forall k.\ (\neg\ P_k\ \texttt{mem butlast}\ C \wedge k < i) \longrightarrow$
    $(\forall m.\ C_0 = P_m \wedge m < k \longrightarrow \neg\ \texttt{Left\_turn}\ P_m\ P_k\ P_i)$
16. $\forall j\ k\ l.\ (j < \texttt{length}\ C\ \text{-}\ 2 \wedge k < j \wedge k < l) \longrightarrow \texttt{Left\_turn}\ C_j\ C_k\ C_l$
17. $\forall\ k < \texttt{length}\ C\ \text{-}\ 1.\ \exists\ n < \texttt{length}\ P.\ C_k = P_n \wedge$
    $((\texttt{drop}\ k\ (\texttt{butlast}\ C))\ \texttt{isConvexHull}\ (\texttt{take}\ (n + 1)\ P) \vee$
    $(\texttt{all\_collinear}\ (\texttt{take}\ (n + 1)\ P) \wedge$
    $(\texttt{length}\ C\ \text{-}\ k = 2 \vee \texttt{length}\ C\ \text{-}\ k = 3)))$

Here i corresponds to the ith point in the ordered list for which the convex hull is being constructed.

It is fairly clear that components 6 to 12 hold true on each iteration of the loop. Note that component 6 states that $i \leq$ length P and not $i <$ length P, as it has to hold true on the termination of the loop.

Component 13 states that on termination of the loop, the first vertex in C is the last point in P and component 14 states that the points in C must belong to the set of points we have already examined in P.

The final three components are a little harder to understand. Component 15 states that the current point we are examining, P_i, cannot lie to the left of the directed line from C_0 through P_k if P_k is a point we have examined after C_0 and before P_i. This is because if C_0, P_k, P_i did make a left turn then we would not have popped P_k.

The fact that the list C has an clockwise ordering is contained in component 16. It states that if we travel along an edge of the hull so far, say C_{j+1} to C_j, then we must make a left turn with respect to vertices of the hull added after C_j.

The final component of the loop invariant reasons about the construction of the convex hull. It says that during every iteration of the loop, the first vertex in C must be one of the points in P, say P_k. Up to P_k we have constructed a convex hull or all the points are collinear. In the case where all the points up to P_k are collinear, C must have length 2 or 3.

It is understandable why a written proof may omit some of the above components, as they seem obvious. However, the process of formal verification highlights the assumptions people make in written proofs. Several of the above components are important and non-trivial and the mechanical verification makes these explicit.

9 Proving the Verification Conditions

Three verification conditions have to be proved in order to show the partial correctness of Graham's Scan. The first one shows that the preconditions imply

the loop invariant with $i = 0$ and C being [hd P, last P]. This is a fairly trivial proof and will not be described here. The third verification condition to prove is:

$$\texttt{Loop invariant} \wedge \neg\ \texttt{i < length P} \implies \texttt{(butlast C) isConvexHull P}$$

The proof of this VC involves deriving the fact $i = \texttt{length}\ P$ from the assumptions. From this and components 6, 13 and 17 of the loop invariant we know the following facts:

a) $\texttt{last}\ P = P_{\texttt{length}\ P-1} = P_k = \texttt{hd}\ C$
b) $(\texttt{butlast}\ C)\ \texttt{isConvexHull}\ (\texttt{take}\ (k+1)\ P) \vee$
$(\texttt{all\_collinear}\ (\texttt{take}\ (k+1)\ P) \wedge$
$(\texttt{length}\ C\ \texttt{-}\ k = 2 \vee \texttt{length}\ C\ \texttt{-}\ k = 3))$

From the assumption that all points in P are distinct and fact *a* above, we show $k = \texttt{length}\ P - 1$. This allows *b* to reduce to $(\texttt{butlast}\ C)\ \texttt{isConvexHull}\ P \vee \texttt{all\_collinear}\ P \wedge \ldots$. From the loop invariant we know that the list of points in P are not all collinear, hence we are left with the desired result.

The second verification condition is the most difficult to prove. We will not go into all the details here but simply mention that it involves showing that the loop invariant and loop test preserve the body of the loop. The proof splits into two cases: one where we turn left with respect to the new point being examined in P, and one where we do not turn left. Both cases require several lemmas to be proven. One of the lemmas required in the case we where we do not turn left is:

$$\texttt{Loop invariant} \wedge \texttt{i < length P} \wedge \neg\ \texttt{Left\_turn}\ \texttt{C}_1\ \texttt{C}_0\ \texttt{P}_i \implies 2 \leq \texttt{length (tl C)}$$

The proof of this lemma follows from component 1 of the loop invariant, which states that $2 \leq \texttt{length}\ C$. For the case where C has more than two elements the above lemma is easily proved. When C has exactly two elements the proof is more complex. In this case, C=[$\texttt{hd}\ P$, $\texttt{last}\ P$]. Furthermore, it can be shown that all points in P, excluding the first and last, either lie to the left of the directed line from $\texttt{last}\ P$ to $\texttt{hd}\ P$, or lie between the first and last points of P. In the case where $i < \texttt{length}\ P - 1$, we can use this fact together with the assumption $\neg\ \texttt{Left\_turn}\ C_1\ C_0\ P_i$, to show that the point P_i lies between $\texttt{hd}\ P$ and $\texttt{last}\ P$. If we have reached this case then it must be true that there exists an $f < i$ which is the *furthest first polar angle* point. Using this and component 13 of the loop invariant, it can be shown that P_f is a member of C. Since we can deduce that P_f is distinct from $\texttt{hd}\ P$ and $\texttt{last}\ P$, it can be shown that C must have three elements. However, this contradicts with the assumption that C has exactly two elements. Thus, the lemma is proved for the case where i is less than $\texttt{length}\ P-1$. For the case where $i = \texttt{length}\ P-1$, we follow a similar argument.

One lemma which needs to be proved for the case where we make a left turn with respect to P_i is:

```
Loop invariant ∧ i < length P ∧ Left_turn C1 C0 Pi
    ⟹ ∀j k l. j < length (Pi # C) ∧ k < j ∧ l < k ⟶
          Left_turn (Pi # C)j (Pi # C)k (Pi # C)l
```

This lemma shows that when we add a new vertex to the list C, the *ordering* property of vertices is maintained. The proof proceeds by eliminating the universal quantifiers and implication in the conclusion to give:

```
Loop invariant ∧ i < length P ∧ Left_turn C1 C0 Pi ∧
j < length (Pi # C) ∧ k < j ∧ l < k
   ⟹ Left_turn (Pi # C)j (Pi # C)k (Pi # C)l
```

For the case where $l{\neq}0$ we can instantiate the universially quantified variables in component 16 of the loop invariant to $j-1$, $k-1$ and $l-1$ respectively. The proof follows easily from these instantiations. When $l{=}0$, the proof is more complex. We are left to show:

$$\texttt{Left\_turn C}_{j-1}\ \texttt{C}_{k-1}\ \texttt{P}_i \qquad (\star)$$

We shall not discuss the proofs relating to the cases where $k-1$ is 0 or 1, as they are fairly trivial compared to the general case. Knuth's Axiom 5b, which was described earlier, and component 16 of the loop invariant were used to prove the general case. From component 16 we can derive the fact:

c) $\texttt{Left\_turn C}_{j-1}\ \texttt{C}_{k-1}\ \texttt{C}_0$

Then using Knuth's Axiom 5b with the instantiations $s{=}C_1$, $t{=}C_0$, $p{=}P_i$, $q{=}$`last butlast` C and r$=C_{j-1}$ we derive the fact:

d) $\texttt{Left\_turn C}_{j-1}\ \texttt{C}_0\ \texttt{P}_i$

Notice that in order to use Knuth's Axiom 5b, with these instantiations, we need the following five conditions to hold:

1. $\texttt{Left\_turn C}_1\ \texttt{C}_0\ \texttt{P}_i$
2. `Left_turn` $\texttt{C}_1$ $\texttt{C}_0$ `(last butlast C)`
3. $\texttt{Left\_turn C}_1\ \texttt{C}_0\ \texttt{C}_{j-1}$
4. $\texttt{Left\_turn C}_0\ \texttt{P}_i$ `(last butlast C)`
5. $\texttt{Left\_turn C}_0$ `(last butlast C)` $\texttt{C}_{j-1}$

Condition 1 is already in our assumptions and 2, 3 and 5 can be proved easily from component 16 of the loop invariant. The trickiest part is in showing condition 4 holds. Recall from the loop invariant that we know (`last butlast` C) is always `hd` P. We can also show that there must exist an m such that $C_0{=}P_m$ and $m{<}i$. From the fact P is ordered it can then be shown that C_0 comes *before* P_i. This implies that C_0 either lies between P_i and `hd` P or P_i lies to the left of the directed line from (`hd` P) to C_0. As the first case is impossible, condition 4 must hold true.

From facts (c) and (d) we can then use Knuth's Axiom 5b once again with the instantiations $s{=}$`last butlast` C, $t{=}C_{j-1}$, $p{=}C_{k-1}$, $q{=}C_0$ and r$=P_i$ to derive ($\star$). This completes the proof.

To illustrate another lemma which is required for the case where we make a left turn with respect to P_i, we give the following example:

```
Loop invariant ∧ i < length P ∧ Left_turn C₁ C₀ Pᵢ
 ⟹ ∀k<length (Pᵢ # C) - 1. ∃n<length P.
      (Pᵢ # C)ₖ = Pₙ ∧
      ((drop k (butlast (Pᵢ # C))) isConvexHull (take (n + 1) P) ∨
        all_collinear (take (n + 1) P) ∧
          (length (Pᵢ # C) - k = 2 ∨ length (Pᵢ # C) - k = 3))
```

We omit the proof of this lemma, as it is extremely intricate and very long. Parts of it are interesting, however, and we note that the key step is identifying three cases: either the stack contains only the initial two points (`last` P and P_0); or there are three points on the stack (`last` P, P_0, and P_h, where $h > 0$ and $P_0 \ldots P_h$ are collinear); or `butlast` C is the convex hull of all the points in P considered thus far. For each of these cases, it is necessary to show that if P_i makes a left turn with C_1 and C_0, the updated stack (P_i # C) still satisfies one of the above three cases.

10 On Automation

Based on our experience, automating the proof of Graham's Scan in Hoare logic is a challenging task. This is, in part, due to the formulation of the loop invariant, which is a complex process. Although several researchers have attempted to automate the proofs of imperative programs using Hoare logic, they have not had great success. The most promising results have come out of the work done by Stark and Ireland, who investigated the automatic discovery of loop invariants in the CLAM proof planning system [16]. They exploited the relationship between while loops and tail recursive functions and were inspired by previous work in CLAM which found generalisations in inductive proofs. A proof planning framework was used to express their heuristics and their technique was based on analysing failed proof attempts. In their approach, invariants were first guessed and then systematically refined every time a proof attempt failed. A heuristic known as rippling [1] was used in the proof discovery stage and if a proof failed then this could be attributed to a ripple being blocked. The blocking of a ripple can suggest some patch which will give a new loop invariant. Despite this work showing a good foundation to build upon, it is worth bearing in mind that Stark and Ireland did not test their approach on geometric problems. It is therefore unclear how applicable their approach would be in this domain.

Incorporating automation into the mechanisation process has wider scope than just discovering the correct loop invariant. In our case, the verification conditions generated in Isabelle are merely statements in higher order logic. If we could automate the proofs of these statements, our task would be greatly simplified. On inspection of our verification conditions, it seems that they could be translated into algebraic form. This hints that Wu's method [17] or some other algebraic methods could perhaps be applied to solve them automatically. However, the drawback of using this technique is that it would be extremely difficult to understand the proofs intuitively because the output would consist of complicated computations of polynomials. It is also worth noting that many

of our lemmas required reasoning about signed areas. This observation suggests that incorporating the Signed Area method [2] into Isabelle may automate many of our proofs.

11 Related Work

Although our work is the first to formally verify Graham's Scan in Hoare logic, it is not the first to mechanise the correctness proofs of convex hull algorithms. Similar to our research, Pichardie and Bertot were inspired by the work of Knuth [14]. However, unlike the current research, they followed Knuth's approach of using axioms. Using these axioms they proved the correctness of an incremental algorithm and a package wrapping algorithm in the theorem prover Coq. They adopted Knuth's method of disallowing collinear points in their CC system, but then this was modified using two different approaches. Their first approach merely added more axioms into their system. Their second one is more interesting as they formalised a perturbation technique for dealing with degenerate cases. This technique was flawed however, as points which lay between two adjacent vertices were sometimes returned as legitimate vertices.

12 Conclusion

In conclusion, we believe that formally proving geometric algorithms in a theorem prover like Isabelle adds confidence in their correctness and we believe it should become an important stage in the development process of such algorithms. Despite the difficulty of proving these algorithms at present, we believe that by building libraries of useful theories and gaining a better understanding of the field, this task will get easier. Our work demonstrates how successful Hoare logic can be for formalising geometric algorithms. Not only does it allow the formal specifications to resemble the algorithm, but it forces one to think carefully about algorithms involving loops. By formalising the facts that never change on each iteration of the loop, it is often possible to get a better understanding of the algorithm and reveal unforeseen situations that might otherwise go undetected.

Although confidence in geometric algorithms can be boosted by proofs such as ours, it is important to highlight that these proofs do not guarantee complete correctness. In some instances appropriate input data can unfortunately cause a seemingly correct program to fail. This is due to the fact that geometric algorithms commonly suffer from the problem of nonrobustness, which is caused by two factors: the use of real-world data, which may be degenerate, and the substitution of floating-point arithmetic for real arithmetic. For future work, we plan to incorporate the issues of nonrobustness into the mechanical proof.

References

1. D. Basin and T. Walsh. *A Calculus for and Termination of Rippling.* Journal of Automated Reasoning, vol. 16, no. 1-2, pp 147-180, 1996.

2. S. C. Chou, X. S. Gao, and J. Z. Zhang. Automated generation of readable proofs with geometric invariants, I. multiple and shortest proof generation. *Journal of Automated Reasoning*, 17:325-347, 1996.
3. A. Church. A formulation of the simple theory of type. *Journal of Symbolic Logic*, 5:56-68, 1940.
4. M. Gordon. Mechanizing Programming Logics in Higher Order Logic. *Current Trends in Hardware Verification and Automated Theorem Proving*, G. Birtwistle and P. A. Subrahmanyam, Springer, 1989.
5. M. Gordon and T. Melham. *Introduction to HOL: A theorem proving environment for Higher Order Logic*. Cambridge University Press, 1993.
6. R. L. Graham. *An efficient algorithm for determining the convex hull of a finite planar set*. Info. Proc. Lett. 1, pp 132-133, 1972.
7. D. Hilbert. *The Foundations of Geometry*. The Open Court Company, 2001, 11th edition. Translation by Leo Unger.
8. C. A. R. Hoare. *An axiomatic basis for computer programming*. Communications of the ACM, v. 12 n. 10, pages 576-580, 1969.
9. D. E. Knuth. *Axioms and Hulls*. Lecture Notes in Computer Science, Volume 606. Springer, 1992.
10. L. Meikle and J. Fleuriot. *Formalizing Hilbert's Grundlagen in Isabelle/Isar*. TPHOLs, vol. 2758, pp 319-334, Springer, 2003.
11. T. Nipkow. *Hoare Logics in Isabelle/HOL*. Proof and System Reliability, Kluwer, 2002.
12. J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1994.
13. L. C. Paulson. Isabelle: A Generic Theorem Prover. *Lecture Notes in Computer Science*, Volume 828. Springer, 1994.
14. D. Pichardie and Y. Bertot. *Formalizing Convex Hull Algorithms*. TPHOLs, 346-361, Springer, 2001.
15. F. P. Preparata and M. I. Shamos. *Computational Geomatry: An Introduction*. Springer, 1985.
16. J. Stark and A. Ireland. *Invariant Discovery via Failed Proof Attempts*. Lecture Notes in Computer Science, Volume 1559, page 271. Springer, 1998.
17. W. Wu. Basic principles of mechanical theorem proving in elementary geometries. *Journal of Automated Reasoning*, 2:221-252, 1986.