Convex Hull of a Set of Points

For a given finite set of points in the plane, its *convex hull* is the smallest polyhedron that contains all the points (`https://en.wikipedia.org/wiki/Convex_hull`). Figure 1 shows an example of such a convex hull, where the polyhedron is characterized by an ordered sequence of points, represented by the arrows. The goal of this project is to formally specify, implement, and formally prove correct an algorithm for computing convex hulls.

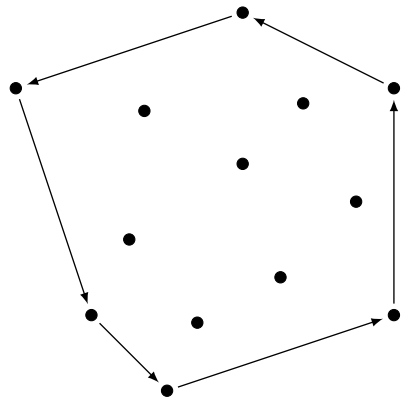

Figure 1: The convex Hull of a set of points

This project is to be carried out using the Why3 tool (version 1.5.x), in combination with automated provers (Alt-Ergo 2.4.x, CVC4 1.8, CVC5 1.0 and Z3 4.8.x or higher). You can use other automatic provers or versions if you want, if they are freely available and recognized by Why3. You may use Coq for discharging particular proof obligations, although the project can be completed without it. The installation procedure may be found on the web page of the course at URL `https://marche.gitlabpages.inria.fr/lecture-deductive-verif/install.html`.

The project must be done individually: team work is not allowed. In order to obtain a grade for the project, you must send an e-mail to `Claude.Marche@inria.fr` and `Jean-Marie.Madiot@inria.fr`, no later than **Thursday, February 23rd, 2023** at 23:00 UTC+1. This e-mail should be entitled "MPRI project 2-36-1", be signed with your name, and have as attachment an archive (`zip` or `tar.gz`) storing the following items:

- The proposed canvas file `convex_hull.mlw` completed by yours.
- The content of the sub-directory `convex_hull` generated by Why3. In particular, this directory should contain session files `why3session.xml` and `why3shapes.gz`, and Coq proof scripts, if any.
- A PDF document named `report.pdf` in which you report on your work. The contents of this report counts for at least half of your grade for the project.

The report must be written in French or English, and should typically consist of 3 to 6 pages. The structure should follow the sections and the questions of the present document. For each question, detail your approach, focusing in particular on the design choices that you made regarding the implementations and specifications. In particular, loop invariants and assertions that you added should be explained in your report: what they mean and how they help to complete the proof.

A typical answer to a question or step would be: *"For this function, I propose the following implementation: [give pseudo-code]. The contract of this function is [give a copy-paste of the contract]. It captures the fact that [rephrase the contract in natural language]. To prove this code correct, I need to add extra annotations [give the loop invariants, etc.] capturing that [rephrase the annotations in english]. This invariant is initially true because [explain]. It is preserved at each iteration because [explain]. The post-condition then follows because [explain].*"

The reader of your report should be convinced at each step that the contracts are the right ones, and should be able to understand why your program is correct, *e.g.*, why a loop invariant is initially true, why it is preserved, and why it suffices to establish the post-condition. It is legitimate to copy-paste parts of

your Why3 code in the report, yet you should only copy the most relevant parts, not all of your code. In case you are not able to fully complete a definition or a proof, you should carefully describe which parts are missing and explain the problems that you faced.

In addition, your report should contain a conclusion, providing general feedback about your work: how easy or how hard it was, what were the major difficulties, was there any unexpected result, and any other information that you think is important to consider for the evaluation of the work you did.

1 Formal Specifications of Points and Convex Hull

The proposed canvas file `convex_hull.mlw` already contains a module `Point` providing definitions for the type `pt` for points, as records of two real numbers. It also contains the definition of the type `pt_set` for sets of points, represented as arrays of points.

To start with, we want to be able to compute extremal points: points whose coordinates are minimal or maximal in a given set. More precisely we want to compute the lowest-leftmost point: the one whose y-coordinate is minimal, and the rightmost among all the points that have a minimal y-coordinate. For that purpose the predicate `is_lower_pt` is provided: `is_lower_pt`$((x_1, y_1), (x_2, y_2))$ is true when either $y_1 < y_2$ or $y_1 = y_2$ and $x_1 \leq x_2$.

1. *Fill in the definition of the program function* `lowest_leftmost` *of module* `Point`*, that compute the index of the lowest-leftmost point of a set.*

2 Counterclockwise Triangles

A key ingredient of this project is the notion of *counterclockwise* triangles. The provided module `CCW` defines a predicate `ccw` which, given three points p_1, p_2 and p_3, captures the intuition that the triangle $p_1p_2p_3$ is oriented *counterclockwise* (and is not flat), as shown on Figure 2. A thesis developed in the book *Axioms and Hulls* [1] (for reference only, there is no need to read any part of this book for the project) is that all the properties needed to reason about convex hulls may be abstracted away into five facts about the predicate `ccw`. These facts are:

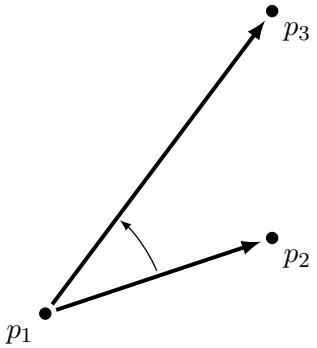

Figure 2: Counterclockwise triangle $p_1p_2p_3$

(Cyclic symmetry) if `ccw`(p_1, p_2, p_3) then `ccw`(p_2, p_3, p_1).

(Antisymmetry) if `ccw`(p_1, p_2, p_3) then not `ccw`(p_2, p_1, p_3).

(Non-degeneracy) if p_1, p_2, p_3 are not colinear, then either `ccw`(p_1, p_2, p_3) or `ccw`(p_2, p_1, p_3).

(Interiority) if `ccw`(q, p_1, p_2), `ccw`(q, p_2, p_3) and `ccw`(q, p_3, p_1) then `ccw`(p_1, p_2, p_3).

(Transitivity) assuming `ccw`(q_1, q_2, p_1), `ccw`(q_1, q_2, p_2) and `ccw`(q_1, q_2, p_3) (that is, all points p_1, p_2, p_3 are to the "left" of the segment q_1, q_2), if `ccw`(q_2, p_1, p_2) and `ccw`(q_2, p_2, p_3), then `ccw`(q_2, p_1, p_3). In other words, given two points q_1 and q_2, the binary relation $x \; R \; y :=$ `ccw`(q_2, x, y) is transitive, when considering points located to the left of the segment q_1, q_2.

The definition of the predicate `ccw` is provided, as well as the statement and the proof of each of the five properties described above. These five properties allow us to abstract the reasoning about angles.

2.1 Points to the Left of a Segment

We now consider the specification, implementation and verification of a program that checks whether all the points, from a given point set, are located to the left of the segment joining two particular points from this point set.

2. *Complete the definition of the predicate* `all_on_left`(s, i, j) *that states that all the points from the point set* s*, distinct from* $s[i]$ *and* $s[j]$*, are located to the left of the segment* $s[i], s[j]$.
3. *Complete the definition of the program function* `check_all_on_left` *that decides whether* `all_on_left`(c, i, j) *holds. Notice that it is on purpose that this function does not return a Boolean but instead raises the exception* `Exit` *when the* `all_on_left` *predicate does not hold.*

2.2 Checking Convex Hull

As already shown in Figure 1, a convex hull is represented as a sequence of points. To represent a sequence, that we call indeed a *path*, a module `Path` is provided. A path in a set of points is in fact a sequence of indices of points from this set. A path representing the convex hull should contain points of the hull in counterclockwise order, without any repetition, as in Figure 1. Remark that any of the points from the convex hull may be used as a starting point for the path.

4. *Define a predicate* `is_convex_hull`(s, p) *that states that the path* p *is a convex hull of the point set* s*. This definition should make use of the predicate* `all_on_left`*. Justify carefully in your report why you pretend that your definition captures faithfully the notion of convex hull. For simplicity, to avoid degenerated cases, you may assume that considered sets of points have a minimal number of elements, say 2 or 3 (explain your choice in your report).*
5. *Complete the definition of the program function* `check_is_convex_hull` *that, given a set of points and a path, decides whether the path describes the convex hull of the point set. This function must return a Boolean value, and it should make use of the function* `check_all_on_left`.

3 Computing Convex Hulls

The goal is to implement and prove an algorithm for computing a convex hull of a given point set. We focus on the *gift wrapping* algorithm, also known as *Jarvis march* (`https://en.wikipedia.org/wiki/Gift_wrapping_algorithm`). The idea of this algorithm is to "wrap" around the set of point, starting from the lowest-leftmost point, as illustrated in Figure 3. At each step, we compute the next point that the wrapping will "touch". We find this point by enumerating the set of all the points from the point set in order to determine the one that is maximal for the relation $x\ R\ y :=$`ccw`(p, x, y), (recall that, by Knuth's fifth axiom, this relation is transitive) where p is the last point from the path in construction, called the *pivot*. The convex hull is completed when the next point that we find is equal to the point that we started with.

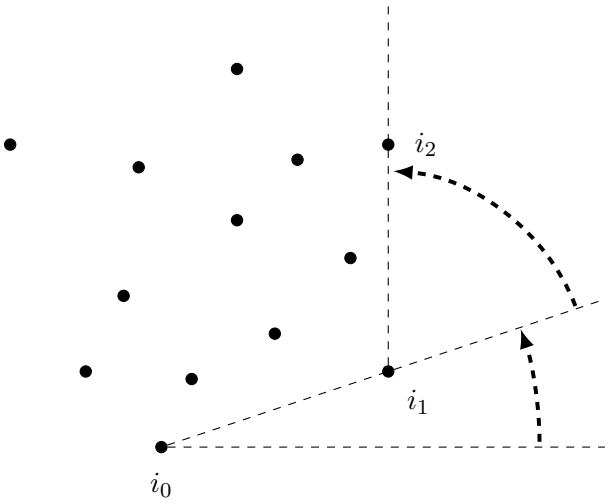

Figure 3: First steps of the gift wrapping algorithm

For simplicity, we rule out all degenerated cases. First, we assume that no three points are ever co-linear. Moreover, we also assume there is a unique point with minimal y-coordinate. Two predicates called `no_colinear_triple` and `unique_minimal_y` are provided for this purpose. They can be added in the preconditions of your functions when needed.

3.1 Correctness of Jarvis Algorithm

A canvas for the code of the convex hull algorithm is provided. It consists of a main function called `jarvis` and an auxiliary function called `largest`, which computes the maximal point for the relation R defined above.

Again, note that the program never needs to make use of any kind of computation of angles: all computations are performed in terms of the predicate `ccw`. Finding the appropriate specification for the auxiliary function `largest` requires a good understanding of the proof of the main loop of the `jarvis` function. We therefore advise to approach the problem as follows:

6. *Prove the function* `jarvis_no_termination` *(a version ignoring termination), in the same time as developing the appropriate specification for* `largest`.
7. *Prove correct the function* `largest`.

Note that you may need to state auxiliary lemmas to prove the functions above. In such a case, do not forget to explain these lemmas in your report, and how they are proved, *e.g.*, using lemma functions.

3.2 Termination

As a last step, we focus on how to prove termination of the Jarvis algorithm. The canvas file provides a function `jarvis` with the same implementation as `jarvis_no_termination` but without the **`diverges`** clause in its contract. The purpose of this new program function is now to prove termination of the algorithm. The idea of the proof is somehow simple: the number of elements in the computed path cannot exceed the number of points in the given set. Yet, to prove that fact it is necessarily to invoke some variation of the so-called *pigeon hole principle* saying that since the path does not contains any repeated points, it cannot contain more elements than the set of points.

You are free to proceed on your own to prove the termination, but you may make use of the hint provided by the ghost function `inverse`(l, f, v) which given a function f from the set of integers $\{0, \ldots, l-1\}$ into itself, assuming f is injective, and given a value v, returns an index i such $f(i) = v$. It thus somehow explicitate that f must also be surjective.

8. *Prove the termination of the* `jarvis` *function. Any lemma or ghost function used must be proved too, including the* `inverse` *function if you use it.*

4 Conclusions

Don't forget to end your report with a conclusion that summarizes your achievements, explain the issues you couldn't solve if any, and comment about what you learned when doing this project.

References

[1] Donald Knuth. *Axioms and Hulls*, volume 606 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.