

Proving Computational Geometry Algorithms in TLA+2

Hui Kong^{*§¶}, Hehua Zhang^{†§¶}, Xiaoyu Song[‡], Ming Gu^{†§¶}, Jianguang Sun^{†§¶}

^{*}Dept.CST, Tsinghua University, Beijing, China

Email: kong-h08@mails.tsinghua.edu.cn

[†]School of Software, Tsinghua University, Beijing, China

[‡]Dept.ECE, Portland State University, Oregon, U.S.A

[§]Key Laboratory for Information System Security, Ministry of Education

[¶]Tsinghua National Laboratory for Information Science and Technology

Abstract—Geometric algorithms are widely used in many scientific fields like computer vision, computer graphics. To guarantee the correctness of these algorithms, it's important to apply formal method to them. In this paper, we propose an approach to proving the correctness of geometric algorithms. The main contribution of the paper is that a set of proof decomposition rules is proposed which can help improve the automation of the proof of geometric algorithms. We choose TLA+2, a structural specification and proof language, as our experiment environment. The case study on a classical convex hull algorithm shows the usability of the method.

Keywords—loop invariant; algorithm verification; theorem proving; TLA+2; geometry algorithm;

I. INTRODUCTION

Computational geometry is an important discipline in computer science. It mainly focuses on studying algorithms for solving geometric problems [1]. The applications of geometric algorithms range over a variety of scientific fields such as computer graphics, computer vision, robotics, astrophysics. Therefore, guaranteeing the correctness of these algorithms is crucial, especially when they are used in safety-critical cases.

One common feature of computational geometry algorithms is their involvement in plenty of operations on large numbers of points, lines, or planes. Consequently, array is the most commonly-used data structure to store these geometric elements. This usually means that the loop invariants of these algorithms contain some predicates with quantifiers. Therefore, to prove the correctness of these algorithms, we need some approaches to proving the loop invariants of this form.

However, currently, there are no effective automatic methods for this purpose and it mainly depends on sophisticated researchers to do this work manually in interactive theorem provers like Coq etc [2]. This may work well for those illustrative simple examples, but for those complex realistic problems such as array-based geometric algorithms, the work will become error-prone and uncontrollable.

In this paper, we propose an approach to improving the automation of the proof of the loop invariants of geometric

algorithms. Our interest lies in those whose loop invariants are predicates with quantifiers. The basic idea is as follows:

- 1) structural decomposition. Given a geometric algorithm G and its candidate loop invariant I as well as the proof obligation $G \wedge I \Rightarrow I'$, by transforming G and I into DNF(Disjunctive Normal Form) and I' into CNF(Conjunctive Normal Form) respectively, the proof obligation is decomposed into a number of simpler proof obligations such that each of them is an implication whose precondition and consequence are a conjunction and a disjunction respectively.
- 2) quantifier decomposition. Based on the result of the previous decomposition, the consequence part of every sub-proof obligation should be a PNF(Prenex Normal Form) of the form $Q_1x_1 \dots Q_nx_n : x_1 \in D'_1 \wedge \dots \wedge x_n \in D'_n \Rightarrow P'(x_1, \dots, x_n)$, where Q_i is a universal or existential quantifier. According to the change to domains of quantified variables, each domain D'_i is divided into two parts: the unchanged part and the added part with respect to its corresponding domain D_i before the loop. Based on this division, every sub-proof obligation is further decomposed into lots of much simpler cases which is often obvious to an interactive theorem prover or need a little human intervention.

As convex hulls are widely used in computational geometry, we choose a typical two dimension convex hull algorithm—Graham's Algorithm for our case study. We choose TLA+2 as the algorithm specification and proof language and TLA+ toolbox as the experiment environment.

The remainder of this paper is organized as follows. Section II briefly introduces the preliminaries of our method. Section III introduces the formal specification of convex hull and the Graham's algorithm in TLA+2. Section IV demonstrates on how to prove a loop invariant for a geometric algorithm. Section V explain briefly our experiment result. Section VI introduces related work. Finally, we conclude our work in Section VII.

II. PRELIMINARIES

TLA+2 is a formal specification and proof language based on TLA(Temporal Logic of Actions), a logic developed by Leslie Lamport, which combines temporal logic with a logic of actions[3].

TLA+2 is used to describe behaviors and properties of sequential or concurrent systems. System behaviors are usually specified with a standard TLA+ specification:

$$Init \wedge \square[Next]_{vars} \wedge L \quad (\text{II.1})$$

where *Init* is the initial predicate, *Next* is the next state action, *vars* is the tuple of all variables, *L* is a temporal formula that usually specifies a liveness constraint. *Next* is a disjunction of multiple actions. An action is an expression containing primed and non-primed variables. The meaning of the non-primed variables is the variable's value in current state. The meaning of primed variables is the variable's value in the next state. An action expresses a transition relation.

Generally, important properties of a specification can often be expressed as invariants. Proving that a predicate *P* is an invariant of a specification means proving the formula

$$Init \wedge \square[Next]_v \Rightarrow \square P \quad (\text{II.2})$$

It is proved by finding an appropriate state predicate *Inv* and proving the following three formulae [3]:

$$Init \Rightarrow Inv \quad (\text{II.3})$$

$$Inv \wedge [Next]_v \Rightarrow Inv' \quad (\text{II.4})$$

$$Inv \Rightarrow P \quad (\text{II.5})$$

III. SPECIFYING CONVEX HULL AND GEOMETRIC ALGORITHM

A. Specifying Convex Hull

Intuitively, the convex hull of a set of points in the plane is the shape taken by a rubber band stretched around nails pounded into the plane at each point. Formally, we give a slightly different definition from [2] in TLA+2: .

Definition 3.1: Let *P* and *S* be two arrays of points, we say that *S* is a convex hull of *P* if the following predicate holds,

$$\begin{aligned} IsCnvxHull(S, P) \triangleq & SubArray(S, P) \wedge Len(S) \geq 2 \\ & \wedge Different(S) \wedge Different(P) \\ & \wedge Inside(P, S) \end{aligned} \quad (\text{III.1})$$

where *SubArray(S, P)* is an array-based alternative representation to subset, which is defined as follows

$$\begin{aligned} SubArray(S, P) \triangleq & \forall i : \exists j : i \geq 1 \wedge i \leq Len(S) \wedge j \geq 1 \\ & \wedge j \leq Len(P) \Rightarrow S[i] = P[j] \end{aligned}$$

Different(P) decides whether all the points in *P* are different, *Inside(P, S)* is used to decide whether all the points of *P* lies inside of the polygon formed by *S*,

$$\begin{aligned} Inside(P, S) \triangleq & \forall i, j : i \geq 1 \wedge i \leq Len(S) - 1 \\ & \wedge j \geq 1 \wedge j \leq Len(P) \\ & \Rightarrow NotAtRight(S[i + 1], S[i], P[j]) \\ & \wedge NotAtRight(S[1], S[Len(S)], P[j]) \end{aligned}$$

where *NotAtRight(A, B, C)* means *C* does not lies to the right of the directed line from *A* to *B*.

This formal definition will be used to define the postcondition of a convex hull algorithm. Obviously, the postcondition will consist of many predicates with quantifiers which is exactly what we are interested in.

B. Specifying Graham's Algorithm

The pseudo code of Graham's Scan can be found in [4]. We specify it in TLA+2 as follows. Remark that we omit the sort operation in this specification and assume that *P* is already sorted, which means that all the points appear strictly counterclockwise around the point *P[1]* in the order in which they appear in *P*. *S* is a stack used to store the resulting convex hull.

```
VARIABLES i, P, S, pc
vars == << i, P, S, pc >>
Init == /\ i = 3 /\ S = <<P[2], P[1]>> /\ pc = "Lbl_1"
Lbl_1 == /\ pc = "Lbl_1"
        /\ IF (i <= Len(P))
            THEN /\ IF (LeftOf(S[2], S[1], P[i]))
                    THEN /\ S' = Push(S, P[i])
                          /\ i' = i + 1
                    ELSE /\ S' = Pop(S)
                          /\ UNCHANGED i
                          /\ pc' = "Lbl_1"
            ELSE /\ pc' = "Done"
                 /\ UNCHANGED << i, S >>
        /\ UNCHANGED << P >>
Next == Lbl_1 \/ (pc = "Done" /\ UNCHANGED vars)
Spec == Init /\ [] [Next]_vars
PostCondition == IsCnvxHull(S, P)
```

According to the specification, the verification goal "when the algorithm terminates, the point array *S* is the convex hull of *P*" can be formalized in TLA+2 as:

$$Prop \triangleq pc = "Done" \Rightarrow IsCnvxHull(S, P)$$

It is regarded as an invariant that should always hold for the specification. As a result, we obtain the final **Verification Condition(VC)** of the algorithm that corresponds to formula (II.2).

$$VC \triangleq Init \wedge \square[Next]_{vars} \Rightarrow \square Prop \quad (\text{III.2})$$

As explained in Section II, to prove this formula, we need to find and prove an inductive loop invariant that implies this one. Finding a loop invariant is out of the scope of this paper. In the following section, we focus on how to prove a complex inductive loop invariant by the proposed decomposition rules.

IV. PROVING GEOMETRIC ALGORITHMS

In this section, we take Graham's algorithm as example to demonstrate on how to prove the correctness of a geometric algorithm. Given a geometric algorithm, we follow the paradigm described by the formulae (II.3), (II.4), (II.5) to perform the proof procedure.

A. Establishing the Loop Invariant

A standard method to obtain a candidate loop invariant is replace constants occurring in the postcondition by some variables involved in the loop and then revise the candidate loop invariant according to the failed proof attempt [5]. Inspired by the standard method, we obtain heuristically the initial candidate loop invariant $IsCnvxHull(S, First(P, i - 1))$ from the postcondition $IsCnvxHull(S, P)$, where $First(P, i - 1)$ means $\langle P[1], \dots, P[i-1] \rangle$. Through multiple times of proof attempt and revision, we reach the final loop invariant.

$$\begin{aligned}
I &\triangleq \forall j : j \geq 2 \wedge j \leq i - 1 \\
&\Rightarrow (P[j] = S[1] \Rightarrow IsCnvxHull(S, First(P, j))) \\
&\wedge \forall j, k : j \geq 2 \wedge j \leq i - 1 \wedge k \geq j + 1 \wedge k \leq i - 1 \\
&\Rightarrow (P[j] = S[1] \Rightarrow \neg LeftOf(S[1], P[k], P[i])) \quad (IV.1) \\
&\wedge i \leq Len(P) + 1 \\
&\wedge i = Len(P) + 1 \Rightarrow S[1] = P[Len(P)] \\
&\wedge pc = "Done" \Rightarrow i = Len(P) + 1
\end{aligned}$$

According to Definition 3.1, the expansion of the formula (IV.1) consists of a number of predicates with quantifiers. This is a typical form for most geometric algorithms. In the following subsection, we introduce how to prove a loop invariant of this form.

B. Decompose Proof Obligation

According to the paradigm introduced in Section II, the proof of the formula (III.2) is transformed into the proof of the following three formulae (II.3), (II.4) and (II.5).

$$Init \Rightarrow I \quad (IV.2)$$

$$[Next]_{vars} \wedge I \Rightarrow I' \quad (IV.3)$$

$$I \Rightarrow Prop \quad (IV.4)$$

where $Init$ and $Next$ are defined in the Graham's algorithm specification. The proofs of formula (IV.2) and (IV.4) are relatively intuitive, we don't bother to detail on them. We lay the emphasis on the proof of the formula (IV.3) which is much more complicated due to the existence of numbers of predicates with quantifiers. Regarding this kind of proof obligation, we propose a structure decomposition rule and a quantifier decomposition rule correspondingly for the overall proof structure and the predicates, both of which can be implemented on computer to break a monolithic proof obligation into many simpler cases.

Rule 4.1: Structure Decomposition Rule(SDR). For a given a geometric algorithm, let $Next$ be the next state action and Inv be the candidate loop invariant. Suppose

$Next$ has the following DNF and Inv has the following DNF and CNF respectively:

$$\begin{aligned}
Next &\triangleq A_1 \vee A_2 \vee \dots \vee A_p \\
Inv &\triangleq B_1 \vee B_2 \vee \dots \vee B_q \triangleq C_1 \wedge C_2 \wedge \dots \wedge C_r
\end{aligned}$$

then the proof obligation $[Next]_{vars} \wedge Inv \Rightarrow Inv'$ is equivalent to the following formula:

$$\begin{aligned}
\forall j, k, l : j \geq 1 \wedge j \leq p \wedge k \geq 1 \wedge k \leq q \\
\wedge l \geq 1 \wedge l \leq r \Rightarrow (A_j \wedge B_k \Rightarrow C'_l) \quad (IV.5)
\end{aligned}$$

According to the rule SDR, by computing the corresponding DNF and CNF of the specification and the loop invariant, a monolithic proof obligation can be broken into $p \times q \times r$ simpler proof obligations, where p , q and r are decided by the depth of the expansion of the predicates. In order to control the scale of the generated subproof obligations, the expansion should be performed case by case and layer by layer according to the result of each subproof attempt.

Based on this method, for example, the DNF of Graham's algorithm specification consists of 4 conjunctions (refer to the TLA+2 specification of the algorithm in Section III-B):

- 1) $A_1 \triangleq pc = "Lb1" \wedge i \leq Len(P) \wedge LeftOf(S[2], S[1], P[i]) \wedge S' = Push(S, P[i]) \wedge i' = i + 1 \wedge pc' = "Lb1"$
- 2) $A_2 \triangleq pc = "Lb1" \wedge i \leq Len(P) \wedge \neg LeftOf(S[2], S[1], P[i]) \wedge S' = Pop(S) \wedge i' = i \wedge pc' = "Lb1"$
- 3) $A_3 \triangleq pc = "Lb1" \wedge \neg(i \leq Len(P)) \wedge i' = i \wedge S' = S \wedge pc' = "Done"$
- 4) $A_4 \triangleq pc = "Done" \wedge UNCHANGED vars$

As for the loop invariant I , the DNF has 1 conjunctions and the CNF has 5 disjunctions in the case of non-expansion. According to the SDR rule, the number of the resulting sub-proof obligations is 20. Among of these subproof obligations, some are fairly trivial and hence can be proved automatically, some need to be further decomposed.

In the following part, we introduce another rule-quantifier decomposition rule. The idea is based on the observation that every combination of the partitions of the domains of the quantified variables reflects a different geometric situation that may require different geometric theorems to prove.

Suppose C is a formula with the following PNF:

$$C \triangleq \Gamma \vec{s} : \vec{s} \in \vec{S} \Rightarrow P(\vec{s}) \quad (IV.6)$$

where

$$\Gamma \triangleq (Q_1, Q_2, \dots, Q_n), \quad Q_i \in \{\forall, \exists\}, \quad \vec{s} \triangleq (s_1, s_2, \dots, s_n)$$

$$\Gamma \vec{s} \triangleq Q_1 s_1 : Q_2 s_2 : \dots : Q_n s_n$$

$$\vec{S} \triangleq S_1 \times S_2 \times \dots \times S_n \quad (i.e. = \prod_{i=1}^n S_i)$$

Under this definition, the n -dimension space \vec{S}' can be represented as the union of 2^n n -dimension subspaces, which can be written as:

$$\vec{S}' = \bigsqcup_{i=1 \dots n} \widehat{S}_i \in \{\widehat{S}_i, S_i^*\} \prod_{j=1}^n \widehat{S}_j \quad (IV.7)$$

where $\overline{S}_i \triangleq S_i \cap S'_i$, $S_i^* \triangleq S'_i - S_i \cap S'_i$.

Consider that the predicates with universal quantifiers have different decomposition paradigm from the predicates with existential quantifiers, we divide S_i into two groups by the quantifier Q_i , represented as:

$$S_{\forall} \triangleq \{i|Q_i = \forall\}, S_{\exists} \triangleq \{i|Q_i = \exists\}$$

Rule 4.2: Quantifier Decomposition Rule(QDR). Suppose A is an action of the algorithm specification, B is a conjunction of the DNF of the loop invariant, C is a disjunction of the CNF of the loop invariant, which has the PNF expressed as the formula (IV.6). For a given subproof obligation $A \wedge B \Rightarrow C'$, it's equivalent to the following formula:

$$\bigwedge_{i \in S_{\forall}} (\bigvee_{j \in S_{\exists}} \widehat{S}_j \in \{\overline{S}_j, S_j^*\}) A \wedge B \Rightarrow (\Gamma \vec{s} : \vec{s} \in \prod_{k=1}^n \widehat{S}_k \Rightarrow P'(\vec{s})) \quad (IV.8)$$

According to the rule QDR, a subproof obligation is decomposed into $2^{|S_{\forall}|}$ disjunctions.

Usually, proving every conjunct of the above formula requires different proof strategy depending on the essence of the combination of the partitions. In fact, proving the loop invariant thoroughly involves several important geometric theorems and plenty of human intervention, decomposition provides an automatic mechanism to separate the geometric situations in which different geometric theorems apply.

V. EXPERIMENT

In this proving practice of Graham's algorithm, we wrote more than 800 lines of proof codes in the integrated development environment *TLA+ toolbox* with the version 1.2.1. We ran it on a laptop with an Intel Core I3-370 2.4G CPU and 4G memory space. It took the computer 4.5 minutes to finish all the proofs.

VI. RELATED WORK

The application of formal method to computational geometry problem is not pervasive. In 1992, Knuth designed a counter-clockwise(CC) system by an axiomatic approach in his book *Axioms and Hulls* [6] to formalize geometric problem. Based on the geometric axioms presented by Knuth, Pichardie and Bertot proved the correctness of an incremental algorithm and a package wrapping algorithm with the theorem prover Coq[2]. Chou [7] uses algebraic method to automatically prove theorems about figures like straight lines, triangles, and circles, geometric algorithms are out of their scope.

Verifying an algorithm by verifying its loop invariant by the theory of Floyd-Hoare logic is one of the main methods. Denmat et al and Ponsini et al proposed constraint-based reasoning approach to prove the loop invariant [8], [9]. Some heuristic methods about finding loop invariant are proposed in [5], [10], [9].

VII. CONCLUSION

In this paper, we propose an approach to proving the correctness of geometric algorithms. Focusing on the loop invariants consisting of some predicates with quantifiers, we propose two proof obligation decomposition rules to transform a monolithic proof obligation into numbers of simpler subproof obligations, which can implemented to improve the automation of the proof procedure. By taking Graham's algorithm as example, we demonstrate the application of the decomposition rules. The result shows that the approach is practical in computational geometry field.

In the future work, we will try to improve the automation extent of the approach and implement it in tools.

ACKNOWLEDGMENT

This research is supported in part by NSFC Programs (No.91018015, No.60811130468) and 973 Program (No.2010CB328003) of China and system@tic-Ile de France Region-CSDL.

REFERENCES

- [1] F. Preparata and M. Shamos, *Computational geometry: an introduction*. Springer, 1985.
- [2] D. Pichardie and Y. Bertot, "Formalizing convex hull algorithms," *Theorem Proving in Higher Order Logics*, pp. 346–361, 2001.
- [3] L. Lamport, *Specifying systems: The TLA+ language and tools for hardware and software engineers*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2002.
- [4] J. O'rourke, "Computational geometry," *ACM SIGACT News*, vol. 25, no. 1, pp. 31–33, 1994.
- [5] J. Stark and A. Ireland, "Invariant discovery via failed proof attempts," *Logic-Based Program Synthesis and Transformation*, pp. 271–288, 1999.
- [6] D. Knuth, "Axioms and Hulls, volume 606 of Lecture Notes Comput. Sci," 1992.
- [7] S. Chou, X. Gao, and J. Zhang, *Machine proofs in geometry*. World Scientific Singapore, 1994.
- [8] T. Denmat, A. Gotlieb, and M. Ducassé, "Proving or disproving likely invariants with constraint reasoning," *Arxiv preprint cs/0508108*, 2005.
- [9] O. Ponsini, H. Collavizza, C. Fédèle, C. Michel, and M. Rueher, "Automatic verification of loop invariants," in *Software Maintenance (ICSM), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–5.
- [10] S. Kauer and J. Winkler, "Mechanical inference of invariants for FOR-loops," *Journal of Symbolic Computation*, 2009.