An automated geometry theorem prover using complex numbers and unit circle

Octavian-Eugen Ganea
Department of Computer Science
Ecole Polytehnique Federale de Lausanne, Switzerland
e-mail: octavian-eugen.ganea@epfl.ch
supervisor : **prof. Victor Kuncak**

June 10, 2011

Abstract

We present a method for automated theorem provers in geometry based on complex numbers approach. This method covers a large class of constructive geometric statements and was succesfully used to solve a big number of theorems and difficult geometry problems that deal with lines and circles. This approach is limited for the moment to problems where a single circle appears. The main idea is to eliminate the affix of each constructed point from the conclusion of the problem based on the equations that define it. These equations are expressed as homogenous polynomials in affixes and conjugates of affixes of the arbitrary and semiarbitrary points. This method has been implemented in Scala language using the SBT tool and the program has been tested with 20 theorems and problems, including difficult problems from International Mathematical Olympiad for High-School(IMO) students and IMO selection tests from Romania.

1 Introduction

In the past decade, highly successful algebraic methods for automated proving geometry problems and theorems have been developed (see references [1],[2],[3]). Computer programs based on these methods have proved many difficult geometry facts. However, most of them use a double coordinate approach within a cartesian axis system (classical analytical geometry). Few of them reason with circles.

A different approach would be that of using complex numbers to represent the coordinates of each point. Instead of having equations in real numbers x and y representing cartesian coordinates of a point, we can translate them in equations in z and $\overline{z}$ with the advantage that they can become linear (e.g. quadratic equation of the circle in standard real coordinates) and easier to manipulate. For example, the condition that points A and B are different translates from a 2 equations $x_A - x_B \neq 0 \ \wedge y_A - y_B \neq 0$ into a one equation $z_A - z_B \neq 0$ which can be used later to decompose a complicated expression of form $P(z_A) - P(z_B)$. Moreover, one of the biggest advantages is the equation of a circle, in particular unit circle (with center in the center of coordinate system and with radix equal to 1) because from quadratic $x^2 + y^2 = 1$ it becomes $z \cdot \overline{z} = 1$. This equation is the simples possible, so the main idea of my approach is to use points from the unit circle as much as possible. These facts suggest that it is possible to express all conjugates of non-arbitrary points in our configuration as a function of their corresponding affixes - $\overline{z} = f(z)$. As we will see, this function will be a fraction of homogenous polynomials with complex variables. Using these type of formulas, we can elliminate the affixes of all determinstic points (that are described as the intersection of two geometric loci

$\overline{z} = f1(z)$ and $\overline{z} = f2(z)$) and also the conjugates of all semiarbitrary points (described by only one locus) from the final conclusion. After that, it will be the job of the program to try to simplify as much as possible the resulting polynomial from the conclusion in order to verify that the problem is true (the polynomial is equal to 0).

The above method was successfully used by myself during many national and international mathematical contests in high-school. At this moment, this program has the advantage of automatizing this method and can be a good tool for verifying properties of geometric configurations and, maybe, for teaching complex numbers. However, it still misses important facts like dealing with geometric inequalities, differences between interior and exterior of convex figures, areas, segments lengths, and quantifiers. Some of those (e.g. semiplanes) are impossible to integrate as complex numbers restrict their use (for example an equation of a semiplane reduces to inequalities with complex numbers which are very difficult to reason with).

Outline The remainder of this article is organized as follows. Section 2 gives some theoretical basis of complex numbers in geometry using circles (unit circle). The main characteristics of my program and the algorithm behind it are described in Section 3. The tests and results are in Section 4. Section 5 give possible future extensions as well as limitations. Finally, Section 6 gives the conclusions.

2 Complex numbers and the unit circle in geometry

When we are unable to solve some problems in plane geometry, it is recommended to try to do calculus. There are several techniques for doing calculations instead of geometry. The next text is devoted to one of them the application of complex numbers.

The plane will be the complex plane (cartesian system) and each point has its corresponding complex number. Because of that, affixes of points will be often denoted by lowercase letters $a, b, c, d, \ldots$, as complex numbers. Their conjugates will be $\overline{a}, \overline{b} \ldots$.

The unit circle will be a circle centered in the center of the plane (of affix 0) and of radix 1. Thus, any geometric configuration with at least one circle can be translated and scaled such that one of its circles becomes the unit circle; this trasformations would not affect the generality of the problem and will be used as much as possible in the following (because of the simple form of the unit circle equation : $z \cdot \overline{z} - 1 = 0$ or $\overline{z} = \frac{1}{z}$).

The equation of a line AB becomes here: $\frac{x-b}{x-a} \in R$ which will be expressed as :

$$\frac{x-b}{x-a} = \frac{\overline{x}-\overline{b}}{\overline{x}-\overline{a}}$$

or $x\overline{b} + b\overline{a} + a\overline{x} - x\overline{a} - a\overline{b} - b\overline{x} = 0$ which is linear in x and $\overline{x}$

If A and B are on the unit circle, this equation becomes (after substitutions of $\overline{a}$ and $\overline{b}$ and simplification with a - b):

$$x + \overline{x}ab = a + b$$

which is much simpler than the general one above. This suggests that using the unit circle as much as possible will simplify our computations and will have a greater rate of success. We will see that in practice this really happens (see section RESULTS).

The following formulas can be derived easily in the same way:

General facts:
• $AB \parallel CD$ iff $\frac{a-b}{c-d} = \frac{\overline{a}-\overline{b}}{\overline{c}-\overline{d}}$
• $AB \perp CD$ iff $\frac{a-b}{c-d} = -\frac{\overline{a}-\overline{b}}{\overline{c}-\overline{d}}$
• triangle ABC is equilateral iff $a^2 + b^2 + c^2 = ab + bc + ca$
• A,B,C,D concyclic iff $\frac{a-b}{a-d} \cdot \frac{c-d}{c-b} = \frac{\overline{a}-\overline{b}}{\overline{a}-\overline{d}} \cdot \frac{\overline{c}-\overline{d}}{\overline{c}-\overline{b}}$
• centroid of triangle ABC : $g = \frac{a+b+c}{3}$

Unit circle properties:
• equation of chord AB: $\overline{x} = \frac{a+b-x}{ab}$
• projection of point C on chord AB: $x = \frac{1}{2}(a + b + c - ab\overline{c})$

• equation of tangent at unit circle in point A : $\overline{x} = \frac{2a-x}{a^2}$

• the intersection of the tangents from A and B: $x = \frac{2ab}{a+b}$

• the intersection of chords AB and CD: $x = \frac{ab(c+d)-cd(a+b)}{ab-cd}$

• orthocenter of triangle ABC with vertices on unit circle: $h = a + b + c$

These last formulas were deduced automatically by the program (described next) using the general formulas and substitutions and simplifications. Refer to the next section for more details.

It can be seen that these last formulas are much simpler than the general ones and permit to obtain affixes of all deterministic points. Thus, the conclusion of each problem that uses these properties can be expressed as a function of affixes and conjugates of arbitrary points and also affixes of semiarbitrary points (described by a geometric locus).

3 Automated prover

The program implemented was implemented in Scala language and it is not a big one (1700 lines of code). It uses SBT tool. It is composed of 3 main modules: problem parser, polynomial reasoner and problem solver. The algorithm is implemented in the problem solver module (with the help of polynomial defined functions) and will be discussed in its subsection. We will briefly describe each module in the following.

In the problem can be defined 4 types of elements:

• One circle - will be interpreted as the unit circle

• Arbitrary points - points with no restriction. Their affixes and conjugates can not be replaced because are independent from the configuration. They will appear with both affix and conjugate in the conclusion.

• Semi-arbitrary points - points described by a single equation (”equation hyp double” denoted in the program) which represents a geometric locus : here can be a line, a circle (just unit circle for the current implementation), a line perpendicular on another line, a line parallel with another line through a fix point, a perpendicular bisector of a segment, or a tangent to a circle (unit circle). These points will be described by a equation of type $\overline{z} = f(z)$, where the function f is dependent of the other arbitrary and semiarbitary points.

• Deterministic points - points described as the interseciton of two geometric loci of type ”equation hyp double” or directly by a fixed position (”equation hyp single” denoted here : centroid of a triangle, symmetric of a point with respect to another, projection of a point on a line, or midpoint of a segment). These points will be described by one or 2 equations, but the goal of the program is to express their affixes and their conjugates completely using the affixes of arbitrary and semiarbitrary points.

3.1 Syntax and the Parser module

The problem will be expressed as a description, a hypothesis and a conclusion. The hypothesis contains the definition of the unit circle, arbitrary points, semiarbitary points and deterministic points. Future work will also include an addition constraint on all the points that should imply the conclusion.

The syntax is described as follows:

```
problem = description + hypotehsis + conclusion
 description = some text
 hypotesis = arbitrary + semiarbitrary +
           + deterministic + (constraint)?
  arbitrary = arbitrary_circle + (arbitrary_point)*
  semiarbitrary = (semiarbitrary_point)*
    semiarbitrary_point = point + eq_hyp_double
  deterministic = (det_point_single |
                  det_point_double)*
        det_point_single = point + eq_hyp_single
        det_point_double = point + eqn_hyp_double
                    + eq_hyp_double
  conclusion = eq_concl

eqn_hyp_double = line |
               = circle |
               = line + "perp on" + line |
               = line + "paralel with" + line |
               = "perp bisector of " + line |
               = "tangent" + circle + point
```

```
eq_hyp_single = "centroid of tr" + p1 + p2 + p3 |
       = "symmetric of" + point + "to" + point |
       = "projection of" + point + "on" + line |
       = "midpoint of" + point + point

constraint = " collinear" + point + point + point
           = " concurrent" + line + line + line
           = line + " perpendicular on" + line |
           = line + " parallel with" + line |

equ_concl = "collinear " + point + point + point
          = "concurrent " + line + line + line
          = line + " perpendicular on" + line |
          = line + " paralel with" + line |
          = "tr is equilateral " + p1 + p2 + p3|
          = "concyclic " + p1 + p2 + p3 + p4
```

One can see that there is a restriction on the types of loci that can contain points. This restricts our program to process just a class of geometric problems, but there are many difficult configurations that can rise from this.

3.2 Polynomial reasoner

All equations of geometric loci permited by the above syntax can be expressed as homogenous polynomials in complex variables equal to 0. A polynomial in complex variables is defined as a sum of factors in complex variables. A factor in complex variables is defined as a rational constant times a product of complex variables representing point affixes of conjugates.

Thus, we say that a polynomial is homogenous iff each factor of the polynomial has the same degree. The degree of a factor is the sum of degrees of each complex variable that appear in it. By definition, the degree of a point affix will always be 1 and the degree of a conjugate will be -1.

For example, the chord AB of the unit circle will be uniquely characterized (as proved above) by the equation :

$$x + \overline{x}ab - a - b = 0$$

Thus, we define as $x + \overline{x}ab - a - b$ to be its corresponding homogenous polynomial of degree 1.

To deal with polynomials, the program uses first the set of rational numbers together with some of their important operations such as simplify (a rational number is always kept in a irreducible form), plus, minus, etc.

Second, there is a class of type Factor that keeps a rational constant (defined above) and a hash of complex variables together with the power at which they appear in the factor. For example, factor $\frac{1}{2}a^3\overline{a}^2$ will be kept as constant $\frac{1}{2}$ and hash $(a \to 3, \overline{a} \to 2)$. There are methods for simplifying (collapse the same variable if it appears twice, reduce variables that appear at power 0), degree, conjugate entire factor and substitute one variable with another.

Third, the most important is the polynomial object defined as a list of factors. The difficulty here consists of maintaining this list simplified (such that opposite signed terms should not appear in the list, but cancel each other). This corresponds to the "simplify" function. In addition, functions for decomposing a polynomial as a first degree equation in one variable, for substitution of a variable with another or for decomposition of a polynomial with a common factor of form "a-b" or "a+b" using Bezout theorem were needed.

I also maintained fractions of polynomials represented as pairs of polynomials. For this I needed functions for substitution in a polynomial of a variable by a fraction of polynomials (and obtain a new fraction), or for solving a first order degree system or second order degree system. But, the simplify function is the most important here. It tries to simplify both polynomials by common factors of form "a-b" or "a+b" at maximum power or just by common variables. This is needed in order to avoid explosion of the states (fractions of polynomials are kept irreducible) and to find additional restriction that can apply to the problem. For example, when a fraction simplifies with the common factor "a-b" then we know that points A and B must be different in order for the fraction denominator to be non zero and the problem to be correct. These kind of situations can arise when, for example, we consider line AB and need A and B to be different, or A and B must not be diametral opposite in the unit circle (a + b = 0).

The current version of the program is not able to find other types of restrictions because this would imply to try to simplify a fraction of polynomials by more complicated factors than just first order ones. This would imply a very time consumming simplify function which is rarely efficient for high-order common factors.

3.3 Problem solver

The problem solver module is the engine of the program. It uses the following algorithm:

- It maintains three hashsets: one of arbitrary points and their conjugates, one of semiarbitrary points in which every point Z has a fraction of polynomials corresponding to the function f from the formula $\overline{z} = f(z)$ and one of deterministic points that keeps for a point Z the irreducible fractions of polynomials that correspond to its afix z and its conjugate $\overline{z}$
- The solver starts by taking each point definition from the program and updating its corresponding hashset entry by computing the fractions of polynomials from the polynomials corresponding to the point's equations. If it founds an equation of degree higher than 1 for a point (for example the intersection of a line with the circle can have 2 solutions), the program exits stating that it does not know how to handle such a non-deterministic situation.
- At each step, if a restriction (e.g 2 points are different) is discovered, it is reported
- Finally, the program transforms the conclusion in a polynomial and substitutes all the values of deterministic points and conjugates of semiarbitrary points in accordance with the hashsets.
- If the resulting polynomial is 0 after all simplifications , then the problem is correct. Else, it is incorrect (and a counter example can be found).

4 Results

The program was run over 20 geometry theorems (including Pascal thm, Euler thm, Gauss thm, Mathot thm, Simson line) and difficult geometry problems (problem 2 from IMO 1982, problem 2 from 3th IMO selection selection test from Romania 2004, Balkan Mathematical Olympiad for Juniors 2001, etc) and it run successfully.

The program was also tested on some wrong problems and reported the errors correctly.

However, I noticed a consistent difference in the speed of termination in case of using more than 3 arbitrary points. For example, proving the intersection of altitudes of a triangles takes under 1 second if the unit circle is the circumscribed circle of the triangle and 143 seconds if the vertices of the triangle are arbitrary. In the same fashion, Gauss theorem (the midpoints of the diagonals of a complete quadrilateral are collinear) is proved in 4 seconds if A, B and C are put on the unit circle, and in 184 seconds if all the 4 vertices are arbitrary.

This behavior is because simplifications with factors of form "a - b" of fractions of polynomials are almost impossible if A and B are arbitrary variables and very probable if A and B are on the unit circle (because in this case their conjugates where replaced by the inverse of their affixes).

In conclusion, these results encourage the user to use over the unit circle as much as possible (for example, by setting it the circumscribed circle of given triangle ABC).

5 Future work

There is place to make this program better. First of all it must deal with additional constraints over whole configuration. Thus, instead of proving that the polynomial from the conclusion reduces to 0, we should prove that the constraints equations imply that polynomial from the conclusion reduces to 0.

Also, there is room for dealing with quantifiers, length of segments, similar triangles, more circles and reasoning with angles and with areas. However, the author is circumspect in dealing with inequalities in both conclusion and constraints.

Last, the author thinks that this tool can be used together with other automated geometric provers (based on other theories) to complete prove more difficult Euclidian problems and theorems. In fact, some parts of a geometry problem can be proved using complex numbers, others using other tools.

6 Conclusions

The above algorithm was successfully used by myself during many mathematical contests in high-school. Its corresponding program is a useful tool that proved difficult geometry problems and can be used in future, together with other tools, to test the satisfiabilty of different geometric assertions.

References

[1] **Complex Numbers in Geometry**, Marko Radovanovic, Olympiad Training Materials, www.imomath.com

[2] **Automated Geometry Theorem Proving by Vector Calculation**, Shang-Ching Chou et all, ACM 1993

[3] S.C. Chou, Mechanical Geometry Theorem Proving, D.Reidel Publishing Company, Netherlands, 1988

[4] **www.mathlinks.ro**

[5] `http://www.imo-official.org/problems.aspx`

[6] **http://www.cut-the-knot.org/arithmetic/algebra/ComplexNumbersGeometry.shtml**