Verification of non-linear overconstraints in Euclidean geometry with linear programming

Denis Bouhineau

Verification of non-linear overconstraints in Euclidean geometry with linear programming

Denis Bouhineau
Denis.Bouhineau@imag.fr
LGI Bureau D 300, Bat. ENSIMAG
Institut IMAG BP 53x
38041 Grenoble Cedex
France

Abstract

This paper is about one possible exact, normal representation of "constructible" numbers, i.e. ones which can be defined in Euclidean geometry. Due to the special nature of these numbers, we are able to propose a representation which allows use of linear programming to solve linear systems over these algebraic numbers and a straightforward verification of overconstraints on these numbers. In the first part, constructible numbers are introduced as algebraic expressions containing rational numbers and the operations $+$, $-$, $\times$, $\div$ and $\sqrt{}$. Then we develop a representation and an arithmetic for these numbers. Explicit predicates for the manipulation of these number are given in Prolog III. In the third part, this representation is compared with standard floating point representation. The two points of comparison are : time of evaluation and precision of the calculations. These experiments show that computation time is obviously high compared to standard floating point calculation, but that precision become much higher with the same computation time ratio. In order to obtain better performances from computation time and a fixed length for the integer needed for the representation, we evoke as perspective a weak representation of constructible numbers based on finite fields.

Keywords : quadratic algebraic extension, constructible numbers, dynamic geometry, exact representation.

Introduction

In the last decades, graphical user interfaces [Borning, 1981; Helm *et al.*, 1993], drawing packages, geometric micro-world for the teaching of geometry [Baulac *et al.*, 1992; Bouhineau, 1995; Sketchpad, 1991], have been implemented with the same idea : adding constraints in order to provide intelligent behavior for direct manipulation of objects. Constraints specify semantic relationships between objects in user interfaces by means of geometric and spatial relation. In the most evident way, in the domain of education for the teaching of geometry, electronic sketch-book have been produced where geometric constraints are used to define figures. And direct manipulations of the figures are done according to the geometric definition given by the constraints. Apart, these last works have lead to a new frame work for the world of education introduced by a special sessions in the last MAA-AMS joint Meeting conference in January 1995 : Dynamic Geometry.

Softwares packages developed for Dynamic Geometry [Baulac *et al.*, 1992; Sketchpad, 1991] are essentially procedural. Geometric figures are obtained from a procedural construction given by the users with the tools of Euclidean geometry (ruler and compass). Our main work [Bouhineau, 1995] concerns the implementation of a declarative geometric drawing packages. We use Prolog III, both as a constraint logic programming languages with linear solver [Colemauer, 1990] and as an aid in interface development. Actually, the language used in Euclidean geometry contains constraints which are linear for the most part, e.g. parallelism, perpendicularity, belonging to a line, midpoint. But there is one important geometric constraint which is not linear, belonging to a circle. As a consequence, we have adopted the solver of linear systems of equations from Prolog III, and added predicates for the resolution of second degree equation of the form $aX^2+bX+c=0$ where a, b and c are known. These predicates, obviously, introduce algebraic numbers.

The implementation of a first prototype has shown that overconstraints appear very naturally in geometric specifications and that overconstraints are very important in the definition of geometric figures. First of all, overconstraints help the automatic construction of the user's geometric figure. Thus overconstraints are automatically added in the specification given by the user, e.g. when one point A is declared to be on two circles C_1 and C_2, the linear overconstraint $A \in \text{CommonChord}(C_1, C_2)$ is added. Secondly, inequalities are useful overconstraints for specifying geometric figures when multiple solutions are possible, e.g. the definition of the symmetric A' of point A with respect of point O : $A' \in (OA), A' \in Circle(O, A), A' \neq A$.

The introduction of overconstraints in the specification of geometric figures has lead us to consider the verification of these overconstraints. Some overconstraints could be verified by using automatic demonstration, see [Chou, 1988; Tarski, 1951; Wu, 1994]. Here we are only concerned with numerical verifica-

tion of their geometrical properties which can be expressed by algebraic expressions (we do not evoke in this article inequalities $<, >, \leq, \geq$). A problem connected to the verification of overconstraints concerns the representation of constructible numbers. We use one representation so that the linear solver of Prolog III can be used to solve linear systems over these numbers and overconstraints can be check straightforwardly. The solution that we propose is based on a binary tree of rational numbers whose height is the length of the sequence of quadratic extension (see section 1.1) needed to represents the numbers.

In the first section of this paper, we define the nature of numbers which occur in geometry, that is expressions with rational numbers and the operations $+$, $-$, $\times$, $\div$ and $\sqrt{}$ (a special case of algebraic number). In the second section we propose an exact representation and the associated arithmetic for these numbers. This representation is proposed to check correctly properties of geometric figures, i.e. to verify overconstraints, and more important this representation works with Gauss elimination of the linear solver. In the third section our representation and floating point representation are compared using examples taken from school exercises concerning figures construction. The precision of floating point arithmetic and the computing times of our exact arithmetic are shown for four examples.

1 Euclidean geometry and constructible numbers

We recall here rapidly the principal results concerning constructible numbers.

1.1 Preliminary

Below are the notations used in this paper. See [Carrega, 1989; Lang, 1971; Lebesgue, 1989] for complete definition of these notions.

Definition 1 *Let K be a field on Q, and x so that* $x^2 \in K$*, we denote K[x] the quadratic algebraic extension of K over R with x.*

Definition 2 *Let L be an algebraic extension of K, we denote L:K the dimension of L as vector space over K.*

1.2 Euclidean geometry

For simplicity, we consider a geometric figure defined by points whose coordinates are rational number [1] and points obtained as follows :

- intersection of two lines described by known points,
- intersection of one line, defined by two known points, and one circle, defined by its center points (one known point) and another known point belonging to the circle,
- intersection of two circles, each of these circles described by its center and another known point on it.

[1] Some people would have taken only two points with coordinates (0,0) and (0,1)

Assume that we aim to construct a point P from points whose coordinates belong to an algebraic extension K of Q. Then it can be shown that :

- If P is the point of intersection of two lines, then the coordinates of P belong to K.
- If P is the point of intersection of one line (d) and one circle whose center is O and which radius is R, then the coordinates of P belong to the quadratic extension K[δ], with $\delta^2 = R^2 - dist(O,d)^2$.
- If P is the point of intersection of two circles, then P is the intersection of one of the two circles and of the common chord to the two circles. As the equation defining the common chord also belongs to K, we come back to the the above case. The coordinates of P belong to a quadratic extension of K.

Consequently, the purpose of this paper is to study numbers belonging to sequences of quadratic extensions of Q, i.e. numerical expressions containing rational numbers and the operations $+$, $-$, $\times$, $\div$ et $\sqrt{}$ are concerned. Formally we define :

Definition 3 *Constructible numbers make the smallest real field containing rational numbers which is stable by square root (square root is considered as a partial function defined on positive numbers).*

1.3 Examples

The following expressions are constructible numbers :

$$e_1 = \sqrt{2} + \sqrt{3} - \sqrt{5 + 2\sqrt{6}}$$
$$e_2 = \sqrt{1 + \sqrt{3}} - \sqrt{10 + 6\sqrt{3}} + \sqrt{3 + 3\sqrt{3}}$$

The numbers have been of interest for Zippel [1985] and Borodin et al. [1985] because they can be simplified, see also [Landau, 1992]. Actually :

$$e_1 = 0$$
$$e_2 = 0$$

Essentially, constructible numbers like e_1 and e_2 are algebraic numbers, and consequently they can be worked out by general softwares for symbolic calculation as with Mathematica [Wolfram, 1991] or Maple. But the simplification of e_1 and e_2 is not direct, and these two computer algebra systems do not simplify e_1 and e_2 to 0. For example, numeric evaluation of e_1 with Mathematica gives the following uncertain numerical result :

$$N[Sqrt[2] + Sqrt[3] - Sqrt[5 + 2Sqrt[6]], 20] = 0.10^{-34}$$

Another example shows that these general symbolic calculators are not suitable for constructible numbers.
Consider the series :

$$\begin{cases} U_0 = 1,\ U_1 = 2,\ U_2 = \sqrt{2} \\ U_n = U_{n-1} * U_{n-2} + U_{n-3} \end{cases}$$

This series is defined in Q[$\sqrt{2}$] and we have $U_n = a_n + b_n * \sqrt{2}$, with

$$\begin{cases} a_0 = 1,\ a_1 = 2,\ a_2 = 0\sqrt{2} \\ b_0 = 0,\ b_1 = 0,\ b_2 = 1 \\ a_n = a_{n-1}a_{n-2} + a_{n-3} + 2b_{n-1}b_{n-2} \\ b_n = a_{n-1}b_{n-2} + b_{n-1}a_{n-2} + b_{n-3} \end{cases}$$

We would like to obtain :

$$\begin{aligned} U_3 &= 1+2\sqrt{2} \\ U_4 &= 6+\sqrt{2} \\ U_5 &= 10+14\sqrt{2} \\ U_6 &= 89+96\sqrt{2} \end{aligned}$$

Unfortunately, simplifying general algebraic expressions is difficult (as noted before) and Mathematica gives :

$$\begin{aligned} U_3 &= 2\sqrt{2}+1 \\ U_4 &= \sqrt{2}(2\sqrt{2}+1)+2 \\ U_5 &= (\sqrt{2}(2\sqrt{2}+1)+2)(2\sqrt{2}+1)+\sqrt{2} \\ U_6 &= ((\sqrt{2}(2\sqrt{2}+1)+2)(2\sqrt{2}+1)+\sqrt{2}) \\ &\quad *(\sqrt{2}(2\sqrt{2}+1)+2)+2\sqrt{2}+1 \end{aligned}$$

The calculation of U_{18} fails after a few minutes because of the limits of computer main memory. Symbolic calculation limitations are clear and are of two kinds : computing time and memory consumption.

2 Proposed representation for constructible numbers

2.1 General representation

The proposed representation is based on the decomposition of a constructible number in a sequence of quadratic extensions of Q. The crucial point of this representation, i.e. the construction of the sequence of extensions, is considered below in a paragraph on square root extraction.

Suppose we have a sequence of quadratic extensions of Q :

- k_0=Q,
- $k_1 = k_0[\alpha_1]$ where $\alpha_1^2 \in k_0$, $\alpha_1 \notin k_0$. The number α_1 have been chosen such that $k_1 : k_0 = 2$ and then $(1, \alpha_1)$ constitutes a basis of the vector space k_1 over k_0.
- Continuing like that, suppose we have $k_n = k_{n-1}[\alpha_n]$ where $\alpha_n^2 \in k_{n-1}$, $\alpha_n \notin k_{n-1}$. The number α_n have been chosen such that $k_n : k_{n-1} = 2$ and then $(1, \alpha_n)$ constitutes a basis of the vector space k_n over k_{n-1}.

Let $A \in k_n$. Since $(1, \alpha_n)$ is a basis of k_n as a vector space over k_{n-1}, then there is $a_1, a_2 \in k_{n-1}$ such that $A = a_1 + a_2 * \alpha_n$. The representation of A is defined with the binary tree whose leaves are a_1, a_2.

We shall write in the following : $A :: (a_1, a_2)$ in order to be close to the computational model. This notation may be ambiguous, then we shall note $A :: (a_1, a_2)_{k_n}$

Remark Given a sequence of quadratic extensions over Q, the representation of a constructible number is unique.

The unique representation of $A = 0$ in all sequences of extensions k_n is a binary tree whose leaves are all null :

$$A_{i_1 i_2 \ldots i_n} = 0, \quad \forall j = 1..n, i_j \in [1, 2]$$

For example, $0 :: ((0,0),(0,0))_{k[\sqrt{2}][\sqrt{3}]}$

The following predicate `ng_null(E,N)`, true if value of `N` is zero in the sequence of extension `E`, is defined by :

```
ng_null([],0) .
ng_null([E|L],[A,B]) :-
   ng_null(L,A),
   ng_null(L,B).
```

Example : representation of $A = \sqrt{2} + 2\sqrt{3} - 5\sqrt{6}$

In $k[\sqrt{2}][\sqrt{3}]$ we have : $A = a_1 + a_2\sqrt{3}$ with $a_1, a_2 \in k[\sqrt{2}]$, $a_1 = 0 + \sqrt{2}$ and $a_2 = 2 - 5\sqrt{2}$. Using the notation defined above : $A :: ((0,1),(2,-5))_{k[\sqrt{2}][\sqrt{3}]}$

We have also $A \in k[\sqrt{3}][\sqrt{2}]$, since $A = a_1 + a_2\sqrt{2}$ with $a_1, a_2 \in k[\sqrt{3}]$, $a_1 = 0 + 2\sqrt{3}$ and $a_2 = 1 - 5\sqrt{3}$. Using the same notation, $A :: ((0,2),(1,-5))_{k[\sqrt{3}][\sqrt{2}]}$

Similarly, we have $A \in k[\sqrt{6}][\sqrt{2}]$, thus : $A :: ((0,-5),(1,1))_{k[\sqrt{6}][\sqrt{2}]}$. And so on, depending on the choice of the extension where A is defined.

2.2 Elementary arithmetic

We define five operations in this arithmetic. The definition of the four first ones, elementary arithmetic operations, is straightforward with the chosen representation.

Addition

Given $A, B \in k_n$, with $A :: (a_1, a_2)_{k_n} = a_1 + a_2\alpha_n$ and $B :: (b_1, b_2)_{k_n} = b_1 + b_2\alpha_n$.

Then $A + B :: (a_1 + b_1, a_2 + b_2)$ since $A + B = a_1 + b_1 + (a_2 + b_2)\alpha_n$.

Let distinguish addition between elements of k_n and additions between elements of k_{n-1}, we have :

$$A +^n B :: (a_1 +^{n-1} b_1, a_2 +^{n-1} b_2)$$

The operation $+^0$ denote the usual addition in Q.

The predicate `ng_plus(E,A,B,C)`, true if `C` is equal to `A+B` in the extension `E`, can be described by :

```
ng_plus([],A,B,A+B).
ng_plus([E|L],[A1,A2],[B1,B2],[C1,C2]) :-
   ng_plus(L,A1,B1,C1),
   ng_plus(L,A2,B2,C2).
```

Subtraction

Let $A, B \in k_n$ with $A :: (a_1, a_2)_{k_n} = a_1 + a_2\alpha_n$ and $B :: (b_1, b_2)_{k_n} = b_1 + b_2\alpha_n$.

We could define $A - B :: (a_1 - b_1, a_2 - b_2)$. We propose the `ng_minus(E,A,B,C)` predicate below which uses the declarative predicate `ng_plus\4`. Since $C = A - B \Leftrightarrow A = C + B$, we define $C = A - B$ by C is solution of $A = C + B$.

```
ng_minus(L,A,B,C) :-
   ng_plus(L,B,C,A).
```

Multiplication

Given $A, B \in k_n$ where $A :: (a_1, a_2)_{k_n} = a_1 + a_2\alpha_n$ and $B :: (b_1, b_2)_{k_n} = b_1 + b_2\alpha_n$.

Then $A * B :: (a_1 * b_1 + a_2 * b_2 * \alpha_n^2, a_1 * b_2 + a_2 * b_1)$. Distinguishing operations in k_n and operations in k_{n-1}, we get :

$A *^n B :: (a_1 *^{n-1} b_1 +^{n-1} a_2 *^{n-1} b_2 *^{n-1} \alpha_n^2, a_1 *^{n-1} b_2 +^{n-1} a_2 *^{n-1} b_1)$,

The operation $*^0$ denotes the usual multiplication in Q.

The predicate `ng_plus(E,A,B,C)`, true if `C` is equal to `A*B` in the extension `E`, can be described by :

```
ng_mult([],A,B,A*B).
ng_mult([E|L],[A1,A2],[B1,B2],[C1,C2]) :-
   ng_mult(L,A1,B1,T1),
   ng_mult(L,A1,B2,T2),
   ng_mult(L,A2,B1,T3),
   ng_mult(L,A2,B2,T4),
   ng_mult(L,E,T4,T5),
   ng_plus(L,T1,T5,C1),
   ng_plus(L,T2,T3,C2).
```

Division

Given $A, B \in k_n$ with $A :: (a_1, a_2)_{k_n} = a_1 + a_2\alpha_n$ and $B :: (b_1, b_2)_{k_n} = b_1 + b_2\alpha_n$.

We may define $A/B = A * B^{-1}$ where $B^{-1} :: (b_1/(b_1^2 - b_2^2\alpha_n^2), -b_2/(b_1^2 - b_2^2\alpha_n^2))$, but since $C = A/B \Leftrightarrow A = C * B$, we define declaratively $C = A/B$ by C is solution of $A = C * B$.

```
ng_div(L,A,B,C) :-
   ng_mult(L,B,C,A).
```

This definition works with CLP languages with Gauss elimination procedure since the system of constraints, on the coordinates of A, B and C, equivalent to the equation $A = C * B$ is a linear system. The purity of this definition[2] is clearly indebted to logic programming with linear constraints.

Properties of the proposed representation

The predicates proposed above, using the representation defined earlier, have the following characteristics :

- Exact, explicit and normal representation of constructible numbers. The representation is exact and explicit (obvious). The representation is normal, i.e. zero has only one representation, since $\alpha_1, ..., \alpha_n$ are 1-linear independent in $k_0, ..., k_{n-1}$.
- Compatibility with Gauss elimination procedure. One linear system on constructible numbers is transposed, with the predicates $+, -, *, /$ defined above, into a set of linear systems on rational numbers, into exactly 2^n linear system, where n is the length of sequence of quadratic extensions. Each subsystem corresponds with a projection of the complete system on one element of the basis, defined with $(\alpha_i)_{i=1..n}$, of k_n as vector space over k_0. The compatibility with simplex algorithm can not be easily obtained with this representation since $A > B$ is not equivalent with $a_1 > b_1$ and $a_2 > b_2$.
- Exponential complexity of the representation, with the length n of sequence of quadratic extensions. The complexity of the representation is in $O(2^n)$. The complexity of addition and subtraction are exponential, like $O(2^n)$. Complexity of multiplication and division are in $O(5^n)$. But this exponential complexity is not so catastrophic in a teaching environment since geometric figures at school are simple and rarely need more than three or four extensions (It does mean four intersections with circles or more than four).

[2]Usually, definition of division with algebraic numbers needs Bezout relation and long program

2.3 Square root

This is the crucial point.

Let $A \in k_n$, we want to calculate the square root of A, i.e. $\sqrt{A}$. The main idea is to check whether A is a square in k_n or not.

If A is a square k_n, i.e. $A = a^2$ with $a \in k_n, a \geq 0$, then $\sqrt{A} = a$ (we show below how a square root of A can be find in k_n).

If A is not a square in k_n, the calculation of $\sqrt{A}$ introduces a new quadratic extension : $k_{n+1} = k_n[\alpha_{n+1}]$ where $\alpha_{n+1} = \sqrt{A}$ and we get $A :: (0, 1)_{k_{n+1}}$. In that case, we verify that $k_{n+1} : k_n = 2$ since A is not a square in k_n.

We have seen that a quadratic extensions is introduced if A is not a square in k_n. So we have now to show how to compute a square root in k_n. This is the particularity of constructible number : explicit square root can be obtained, if one exists.

Let $A \in k_n$, is there $a \in k_n$ such that $a^2 = A$? Let us consider the following cases :

$n = 0, k_n = k_0 = Q$.

Then A is rational, i.e. $A = N/D$ with N, D integer and $gcd(N, D) = 1$, $N > 0$, $D > 0$.

Then a square root of A exists if and only if N and D have a square root in N. Thus, if $N = n^2, D = d^2$ then $A = a^2$ with $a = n/d$, (we always choose the positive root).

Otherwise A is not a square in Q.

$n > 0$.

Then $A = a_1 + a_2 * \alpha_n$ with $a_1, a_2 \in k_{n-1}$.

A is a square in k_n if and only if there are $x, y \in k_{n-1}$ such that :

$$a_1 + a_2 * \alpha_n = A = (x + y * \alpha_n)^2 \quad (0)$$

Rewriting this equation we get :

$$\begin{cases} a_1 = x^2 + y^2(\alpha_n)^2 \\ a_2 = 2xy \end{cases} \quad (1)$$

We have to consider different cases in order to solve this system of non-linear equations.

$a_2 = 0$.

Then A is a square in k_n if and only if $x = 0$ or $y = 0$.

Assume $x = 0$ then A is a square in k_n if and only if A/α_n^2 is a square in k_{n-1}.

Assume $y = 0$ then A is a square in k_n if and only if A is square in k_{n-1}.

In both cases, we have to look for a square using recurrence in k_{n-1}.

$a_2 \neq 0$.

Then we have to solve the following equation, obtained from (1) by substitution of x :

$$y^4(\alpha_n)^2 - a_1 y^2 + (a_2)^2/4 = 0 \quad (2)$$

We note $Y = y^2$. This leads to the second degree equation :

$$Y^2(\alpha_n)^2 - a_1 Y + (a_2)^2/4 = 0 \quad (3)$$

whose discriminant is : $\Delta = 16(a_1)^2 - 4(a_2)^2(\alpha_n)^2$.

Usually, in real field, solution of (3) exists if the sign of Δ is positive. But here the sign of Δ is not sufficient : Δ must be a square in k_{n-1}. (apart, if Δ is a square, then the question of the sign is settled) [proof : Actually, (1) has solution for the variable y in k_{n-1}, is equivalent to (3) has solutions for the variable Y in k_{n-1}. As $\Delta = (\pm(\alpha_n)^2 Y - 4a_1)^2$, with $\pm(\alpha_n)^2 Y - 4a_1 \in k_{n-1}$, (1) has solution implies that Δ has roots in k_{n-1}. Conversely, if Δ has no root in k_{n-1} then (3) has no solution for Y in k_{n-1}; therefore (1) has no solution in k_n]
If, by recurrence, a root δ of Δ is found in k_{n-1}, i.e. $\Delta = \delta^2$ then two solutions for Y are obtained :

$$Y_1 = (4a_1 + \delta)/2(\alpha_n)^2, \quad and$$
$$Y_2 = (4a_1 - \delta)/2(\alpha_n)^2$$

We look, by recurrence in k_{n-1} for square root y_1 et y_2, of Y_1 et Y_2. If square roots exist for Y_1 or Y_2 then y_1 and y_2 are solutions of (1) in k_{n-1} .
If Y_1 and Y_2 are not squares in k_{n-1} then (1) has no solution and A is not a square in k_n.

Remarks

- As $(\alpha_n)^2 > 0, (a_2)^2 > 0$ solutions of equations (3) have the same sign that a_1. Therefore, if $a_1 < 0$, equation (2) has solution. Otherwise (2) has no solution.
- As $Y_1 * Y_2 = (a_2)^2/(\alpha_n)^2$, if Y_1 is a square in k_{n-1} then Y_2 can not be a square in k_{n-1}. That is because if Y_1 and Y_2 are square in k_{n-1}, i.e. $Y_1 = y_1^2$ and $Y_2 = y_2^2$ with $y_1, y_2 \in k_{n-1}$, then $\alpha_n = a_2/(y_1 y_2)$. Thus we would have $\alpha_n \in k_{n-1}$, and this is impossible by construction (because of the choice of α_n such that $\alpha_n : k_{n-1} = 2$).
  Consequently, Y_1 and Y_2 can not be square simultaneously, and the search for square leads to an unique solution (the only convention is to choose for the positive root)

2.4 Examples of calculation with constructible numbers

Uniquity of the representation

How is represented $A = \sqrt{2} + \sqrt{3} - \sqrt{5 + 2\sqrt{6}}$
We calculate successively :

$$\begin{array}{lll}
A_0 :: & (2)_Q & = 2 \\
A_1 :: & (0,1)_{Q[\sqrt{2}]} & = \sqrt{2} \\
A_2 :: & (3,0)_{Q[\sqrt{2}]} & = 3 \\
A_3 :: & ((0,0),(1,0))_{Q[\sqrt{2}][\sqrt{3}]} & = \sqrt{3} \\
A_4 :: & ((0,1),(1,0))_{Q[\sqrt{2}][\sqrt{3}]} & = \sqrt{2} + \sqrt{3} \\
A_5 :: & ((6,0),(0,0))_{Q[\sqrt{2}][\sqrt{3}]} & = 6 \\
A_6 :: & ((0,0),(0,1))_{Q[\sqrt{2}][\sqrt{3}]} & = \sqrt{6} \\
A_7 :: & ((0,0),(0,2))_{Q[\sqrt{2}][\sqrt{3}]} & = 2\sqrt{6} \\
A_8 :: & ((5,0),(0,0))_{Q[\sqrt{2}][\sqrt{3}]} & = 5 \\
A_9 :: & ((5,0),(0,2))_{Q[\sqrt{2}][\sqrt{3}]} & = 5 + 2\sqrt{6} \\
A_{10} :: & ((0,1),(1,0))_{Q[\sqrt{2}][\sqrt{3}]} & = \sqrt{5 + 2\sqrt{6}} \\
A_{11} :: & ((0,0),(0,0))_{Q[\sqrt{2}][\sqrt{3}]} & = \sqrt{2} + \sqrt{3} \\
 & & \quad - \sqrt{5 + 2\sqrt{6}} \\
 & & = A
\end{array}$$

Most of the calculation in this sequence are "square root", they are performed accordingly to section 2.3. They are not straightforward. On one hand, checking that 3 is not a square in $Q[\sqrt{2}]$ needs computation. On the other hand, in $Q[\sqrt{2}][\sqrt{3}]$ the same computation are performed to find a square root for 6 and for $5 + 2\sqrt{6}$ (the square root of 6 is not evident). The other calculations, "addition", can be verified by hand.

Remark We notice here that the computation in $Q[\sqrt{2}][\sqrt{3}]$ of A gives the unique representation of zero. The computation of A in an other sequence of extension leads to the same result ((0,0),(0,0)) since it is the only representation of zero.

Conciseness of the representation

Calculation of the U_n series, described at the beginning of the paper.

$$\begin{array}{ll}
U_0 :: & (1,0) \\
U_1 :: & (2,0) \\
U_2 :: & (0,1) \\
U_3 :: & (1,2) \\
U_4 :: & (6,1) \\
U_5 :: & (10,14) \\
U_6 :: & (89,96) \\
U_7 :: & (3584,2207) \\
U_8 :: & (742730,540501) \\
U_9 :: & (5047715823,3576360790) \\
U_{10} :: & (7615143139931954,5384565899606230) \\
U_{11} :: & (769533792308900221732942 \\
 & 72,5441425782731872019460145 1) \\
U_{12} :: & (1172005312243417792577922713561480518962771 \\
 & ,828732903874311492000197638627684580540604)
\end{array}$$

An important growth of coefficients's size is observed. This growth is natural in symbolic calculation. This calculation have been executed in less than 0.1 second with a Mac II SI and Prolog III.

3 Tests

Four geometric figures have been chosen. Two tests are performed, the evaluation of the relative error with floating point arithmetic, and the evaluation of the time needed for the computation with floating point representation and with exact representation of constructible numbers.

Note Within the computation with constructible numbers, a computation with floating point arithmetic is done for a fast evaluation of the sign of the expressions.

3.1 Triangle and altitudes

The first geometric figure is the construction of the three perpendicular heights of a triangle. Numerical calculation permits testing whether the three altitudes are concurrent or not. All the calculations are performed in Q, where no extension is introduced. The complexity of this figure can be compared with the resolution of a system of sixteen linear equations with one overconstraint, i.e. the intersection of two perpendicular height belongs to the third one.

The tests give the following results :

	Test 1	Test 2	Test 3
Error (Float P.)	0e-16	1e-10	0e-16
Time (Float P.)	45 ms	45 ms	45 ms
Time (Exact R.)	80 ms	80 ms	328 ms

Test 1 is performed with a general position of the points. Test 2 is performed with two altitudes almost parallel. What is to be noticed is the fact the relative error can be found as big as wanted. Test 3 is performed with big rational numbers (more than 20 digits). This introduces the limitation of exact rational calculations.

3.2 Right triangle inscribed in a circle

The second geometric figure is composed of one circle given by one diameter [A,B] and a line intersecting the circle on point M. Numerical calculation permits testing whether lines (A,M) and (B,M) are perpendicular. One quadratic extension is introduced during the calculation. The complexity of this figure can be compared with the resolution of a system of twelve equations (including non-linear equations) and one overconstraint, i.e. line (A,M) and (B,M) are perpendicular.

The tests give the following results :

	Test 1	Test 2
Error (Float P.)	0e-16	1e-5
Time (Float P.)	82 ms	320 ms
Time (Exact R.)	82 ms	320 ms

Test 1 is performed with a general position of the points. Test 2 is performed with M very close to A. The time needed to perform exact calculation was expected to be at least three times longer than the time for floating point calculation (one extension that double the size of the data, and, moreover, floating point calculations are performed to evaluate the sign of expression).

3.3 Regular triangle

The third geometric figure corresponds to the construction of a regular triangle from one edge A, and the center of the circumscribe circle G. Numerical calculation permits testing whether it is really a regular triangle. One quadratic extension is introduced during the calculation. The complexity of this figure can be compared with the resolution of a system of fourteen equations (including non-linear equations), and verification of two overconstraints, i.e. tests whether vertices are equal.

The tests give the following results :

	Test 1	Test 2
Error (Float P.)	0e-16	1e-9
Time (Float P.)	326 ms	1s
Time (Exact R.)	326 ms	1s

Test 1 is performed with a general position of the points. Test 2 is performed with point G almost on the same vertical line as point A.

3.4 Regular pentagon

This geometric figure corresponds to the construction of a regular pentagon from one edge A, and the center of the circumscribe circle G. Numerical calculation permits testing whether it is really a regular pentagon. Two quadratic extensions are introduced during the calculation. The complexity of this figure can be compared with the resolution of a system of 38 equations (including non-linear equations).

The tests give the following results :

	Test 1	Test 2
Error (Float P.)	0e-16	1e-5
Time (Float P.)	1s	39 s
Time (Exact R.)	1s	39 s

Test 1 is performed with a general position of the points. Test 2 is performed with point G almost on the same vertical line as point A (a line is represented by the equation $y = mx + p$).

The time needed for exact calculation is important because of the number of extensions and the length of the calculation. The two extensions introduced mean that a constructible number is represented with four rational numbers (so exact calculation takes at least four times as long as a floating point calculation). The length of the calculation has lead to important growth of the coefficients of constructible numbers. At the end of calculations, integers used in the representation of constructible numbers are very big (more than 50 digits).

Perspective and further works

The proposed arithmetic has been shown effective for exact computation in Euclidean geometry using Gauss elimination and immediate verification of overconstraints. But two limitations have been observed : the growth of the coefficient (normal in symbolic calculation), and the exponential growth of the representation's size with the number of extensions introduced due to square roots.

In order to avoid the growth of coefficients, a solution can be considered : take a finite field k_0 instead of Q. For example a weak representation can be defined with Q replaced by $k_0 = Z/pZ$, where p is a prime number. Two advantages can be observed by considering finite field $k_0 = Z/pZ$. First, the coefficients are bounded. Second, in a finite field whose characteristic is odd, about one element in two is a square (or a residue square). Consequently, the length of the sequence of quadratic extensions is divided by two. Moreover, the prime number p can be easily chosen to avoid three or four first extensions. Last but not least, when p is chosen such that $p \equiv 3 \pmod 4$, the calculation of the square root is immediate.

The weak representation can be used to evaluate equality between expressions. When expressions are not equal in the weak representation, they are not equal at all. When expressions are equal in the weak representation, they are equal in R with probability $P_{eq} = 1 - P_{er} = 1 - (1/p)^{2^n}$, where n represents the length of the sequence of extensions. For example, with $p = 32749$ and $n = 3$, we have a test with a

probability of error $P_{er} = 7,55818E - 37$ which is very small (equal to 0 for a human being). What is most important is that this probability of error is independent of the length of the calculation.

Acknowledgments

I would like to thank Evelyne Chevigny, Richard Allen and Laurent Trilling for their assistances with the redaction of this paper.

References

[Baulac *et al.*, 1992] Y. Baulac, F. Bellemain, J.M. Laborde, , *Cabri, the interactive geometry notebook*, Brooks/Cole publishing company, Pacific grove, CA, 1992.

[Borning, 1981] Alan Borning, *The programming language aspects of ThingLab -a constraint-oriented simulation laboratory*, in ACM Transaction on Programming Languages and Systems, vol. 3, no.4, page 343–387, October 1981.

[Borodin *et al.*, 1985] Allan Borodin, Ronald Fagin, John E. Hopcroft, Martin Tompa, *Decreasing the nested depth of expression involving square roots*, Journal of Symbolic Computation, 1 page 169-188, 1985.

[Bouhineau, 1995] Denis Bouhineau, *Vers une approche déclarative pour les logiciels de dessins géométriques,* in Actes des quatrièmes journées EIAO de Cachan, ed Eyrolles, 1995.

[Carrega, 1989] Carrega, J.C., *Théorie des corps, la règle et le compas,* réedition, Herman, Paris, 1989.

[Chou, 1988] Chou, S.C., *Mechanical Geometry Theorem Proving,* Reidel Publishing, Norwell, 1988.

[Colemauer, 1990] Colmerauer, A. *An introduction to Prolog III*, Communication of the ACM vol. 33, no. 7 page 69–90, 1990.

[Helm *et al.*, 1993] Helm, R., Huynh, T., Marriot, K., Vlissides, J., *An Object-Oriented Architecture for Constraint-Based Graphical Editing*, Advances in Object-Oriented Graphics II, Springer Verlag, 1993.

[Lang, 1971] Serge Lang, *Algebra,* Addison-Wesley, Reading, MA,1971.

[Landau, 1992] Landau, S., *Simplification of nested radicals,* SIAM Journal of Computing Vol 21, no.1 page 85-110, February 1992.

[Lebesgue, 1989] Henry Lebesgue, *Leçons sur les constructions géométriques,* réédition, Gauthier - Villard, Paris, 1989.

[Sketchpad, 1991] Geometer's Sketchpad, Key Curriculum Press, Berkeley CA

[Tarski, 1951] Tarski, A., *A decision method for elementary algebra and geometry*, University of California Press, second edition, Los Angeles, 1951.

[Wolfram, 1991] Wolfram, S., *Mathematica: a system for doing mathematics by computer,* Addison-Wesley, 1991.

[Wu, 1994] Wu, W., *Mechanical Theorem Proving in Geometries,* Springer-Verlag, Wien, 1994.

[Zippel, 1985] Zippel, R., *Simplification of expression involving radicals,* Journal of Symbolic Computation, 1 page 189-210, 1985.