Solving Geometrical Constraint Systems Using CLP Based on Linear Constraint Solver

Denis Bouhineau

Solving Geometrical Constraint Systems Using CLP Based on Linear Constraint Solver

Denis Bouhineau

Laboratoire IMAG - LSR
BP 53 X, 38041 Grenoble Cedex 9
FRANCE

Abstract. Euclidean geometrical configurations obtained with ruler, square and compass may be described as arithmetic constraint systems over rational numbers and consequently belong to the domain of CLP(R). Unfortunately, CLP based on linear constraint solvers which are efficient and can deal with geometrical constraints such as parallelism, perpendicularity, belonging to a line i.e. pseudo-linear constraints, cannot handle quadratic constraints introduced when using circles.
Two problems arise with quadratic constraints : the first problem is how to solve mixed constraint systems i.e. linear constraints combined with quadratic constraints; the second problem is how to represent the real numbers involved in the resolution of mixed constraints, so that correctness and completeness of linear constraint solvers are preserved.
In this paper we present a naive algorithm for mixed constraints based on a cooperation with a linear constraint solver. We define a representation for the real numbers, i.e. constructible numbers, occuring in Euclidean geometry. This representation preserves correctness and completeness of above algorithms. A survey over 512 theorems of Euclidean geometry shows that from both theoretical and experimental points of view, this representation is appropriate. This work is intended to be used to verify geometrical properties in Intelligent Tutoring System for geometry.
Keywords: quadratic algebraic extension, representation of rational, real, algebraic, and constructible numbers, cooperation between solvers for mixed constraints.

1 Introduction

Constraint Logic Programming (CLP) is based on extensions of Prolog incorporating constraint solving algorithms tailored to specific domains such as trees, booleans, finite domains or real numbers. For example, in the domain of arithmetic constraints over rational numbers, Linear Programming algorithms have been incorporated to CLP languages such as Prolog III, CHIP, and CLP(R) see [Col90, Din88, Jaf92] respectively. The expressive power and flexibility gained by combining Linear Programming, Logic Programming, and Constraint Programming allows one to express systems of pseudo-linear[1] constraints that can be solved using linear constraint solvers. This is particularly true in the case of

[1] equations which become linear after solving some other equations

geometrical configuration defined by points and lines, often used in geometry courses. Moreover in practical issues in the domain of Intelligent Tutoring Systems in geometry, coordinates of objects defining geometrical configurations are directly obtained from the screen and correspond to rational numbers. When an exact representation of rational numbers is used by the solver, solutions given by the linear constraints solver are exact rational numbers not approximate. In this case, as a consequence that linear constraint systems solver over rational numbers algorithms are correct and complete, overconstraints resulting from geometrical overspecifications can be verified.

Unfortunately, Euclidean geometry is not based on lines and points, ruler and square only. Euclidean geometry is based on ruler, square **and** compass. An other point, rational numbers are not sufficient, algebraic numbers are needed to express some geometrical constructions. For example, a equilateral triangle necessarily has one coordinate in an algebraic extension of Q with $\sqrt{3}$. In consequence of the introduction of the compass, geometrical configurations deal quadratic constraints (second degree equations) which cannot be solved using linear programming algorithms over rational numbers. On the other hand, the numbers introduced to express coordinates of all constructible configurations occuring in Euclidean geometry do not belong to the numerical domain of rational numbers where the linear algorithms are complete and correct. Two main problems arise with quadratic constraints and real numbers of geometry :

1. The first problem, namely how to solve set of mixed constraints i.e. linear constraints combined with quadratic constraints, may be solved for certain geometrical configurations. For the particular geometrical configurations called "constructive" in [Chou92], we present an algorithm which is complete. See [Pes95] for complementary approach.
2. The second problem which will be essentially addressed in this paper is how to represent real numbers, called constructible numbers (a special class of algebraic numbers, cf. [Leb92]), involved in the resolution of mixed constraints, and in Euclidean geometry.

We propose in this paper one exact representation for constructible numbers compatible with the linear constraint solvers used in CLP. On the practical side, this representation aims at avoiding numerical errors due to the lack of precision with representations of real numbers with floating point arithmetic. On the theoretical side, this representation preserves the correctness and completeness of the linear constraint solvers over rational numbers. These are two essential perspectives in the application domain of Teaching Geometry in which we are concerned, cf. [All93, Lab95].

The paper is organized as follows. Section 2 sketches the resolution of geometrical constraints with the help of the linear solver of Prolog III. Section 3 introduces a basic algorithm for solving mixed constraints. Sections 4 and 5 are concerned with the representation of constructible numbers. Section 6 provides examples of solution and includes details about the complexity of geometrical

configurations arising in practice : a survey of about 512 geometrical configurations from [Chou88] is exposed. A short conclusion ends the paper.

2 Geometric Configuration and Linear Constraint Solvers

Let's take one configuration from the 170 theorems without circles among the 512 theorems given by Chou[2] in [Chou88]. For example, let us consider theorem 82 page 143 :

Theorem 82 In triangle ABC, let F be the midpoint of the side BC, D and E the feet of the altitudes on AB and AC respectively, FG is perpendicular to DE at G. Show that G is the midpoint of DE.

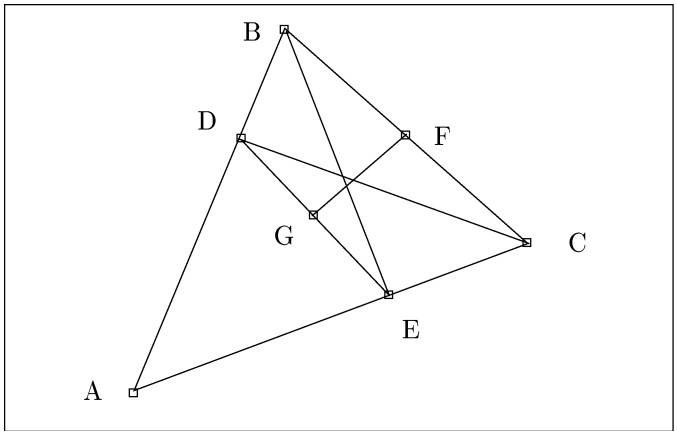

Fig. 1. Configuration 82

The configuration observed in theorem 82 can be specified in Prolog III in the following way :

```
Collinear(<A_x,A_y>,<B_x,B_y>,<C_x,C_y>) ->
    { (A_x-B_x)*(B_y-C_y)-(A_y-B_y)*(B_x-C_x)=0 };
Perpendicular(<A_x,A_y>,<B_x,B_y>,<C_x,C_y>,<D_x,D_y>) ->
    { (A_x-B_x)*(C_x-D_x)+(A_y-B_y)*(C_y-D_y)=0 };
Midpoint(<A_x,A_y>,<B_x,B_y>,<C_x,C_y>) ->
    { 2*B_x-A_x-C_x=0, 2*B_y-A_y-C_y=0 };

Configuration82(A,B,C,D,E,F,G) ->
```

[2] Numerous examples can be also found among theorems in Projective Geometry.

```
    Midpoint(B,F,C)
    Perpendicular(D,C,A,B)
    Collinear(D,A,B)
    Perpendicular(E,B,A,C)
    Collinear(E,A,C)
    Perpendicular(G,F,E,D)
    Collinear(G,E,D);

Theorem82(A,B,C) ->
    Configuration82(A,B,C,D,E,F,G)
    Midpoint(D,G,E);
```

The representation of the configuration using Prolog predicates is straightforward, and elegant. Within the same formalism, the theorem can be expressed as well. The solution of the linear constraint system gives complete configuration.

```
> Configuration82(<3,0>,<6,8>,<12,5>,D,E,F,G);
   {D = <420/73,536/73>, E = <921/106,335/106>, F = <9,13/2>,
    G = <111753/15476,81271/15476>}
```

Since the linear constraint solver of Prolog III computes exact solution of linear system over rational numbers, the overconstraint `Midpoint(D,G,E)` resulting from geometry in the theorem 82 is numerically verified :

```
> Theorem82(<3,0>,<6,8>,<12,5>);
    {}
```

The method informally introduced in this section provides sound results when the constraints introduced are pseudo-linear, see [Col93]. For the class of constructive configuration, the linear constraint solver approach is complete and correct.

3 Solution of Mixed Constraint Systems

This section is devoted to the solution of mixed constraint systems obtained when geometrical properties about circles are introduced. In [Chou88], 342 theorems out of 512 correspond to configurations with circles. For example let us consider theorem 110 on page 156 :

Theorem 110 Let D, E be two points on sides AC and BC of a triangle ABC such that AD=BE, F the intersection of DE and AB. Show that FD.AC=EF.BC.

The configuration in theorem 110 can be specified in Prolog III as follows :

```
Collinear(<A_x,A_y>,<B_x,B_y>,<C_x,C_y>) ->
    { (A_x-B_x)*(B_y-C_y)-(A_y-B_y)*(B_x-C_x)=0 };

EquiDist(<A_x,A_y>,<B_x,B_y>,<C_x,C_y>,<D_x,D_y>) ->
     { (A_x-B_x)^2+(A_y-B_y)^2-(C_x-D_x)^2-(C_y-D_y)^2=0 };
```

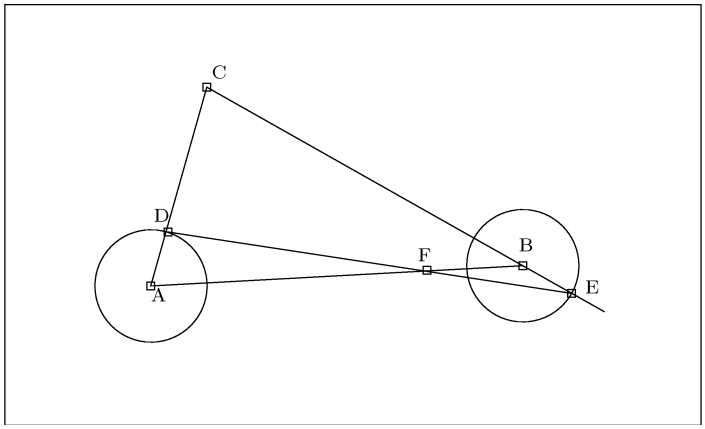

Fig. 2. Configuration 110

```
Configuration110(A,B,C,D,E,F) ->
    Collinear(A,C,D)
    EquiDist(E,B,A,D)
    Collinear(E,C,B)
    Collinear(F,E,D)
    Collinear(F,A,B);
```

The constraint system obtained here cannot be solved directly with linear constraint solvers. So we used the interaction between the linear constraint solver of Prolog III and a quadratic constraint solver. Each linear geometric constraint is given to the linear constraint solver. Pseudo-linear constraints are "frozen" until they become linear and then given to the linear constraint solver (as it is done automatically in Prolog III). Quadratic constraints are handled by a special program whose purpose is to transform them symbolically by interacting with the linear solver.

The quadratic constraint solver operates as follows : Let (E) $aX^2 + bX + cXY + dY^2 + eY + f = 0$ be a symbolic constraint

1. if it is found that $Y = mX + p$, then (E) is transformed into (E') $a'X^2 + b'X^+c' = 0$, and equation (E') is solved in a classical way, i.e. $X = (-b' \pm \sqrt{b'^2 - 4a'c'})/(2a')$.
2. else if (E) becomes linear this result is passed on to the linear constraints solver.
3. else the process is dis-activated until a new activation.

The naive algorithm presented here is complete for geometric configuration of constructive type. Assuming that the square root computed in step 1. is correct, the algorithm is correct.

4 Exact representation of constructible numbers

The algorithm given in the previous section introduces true real numbers during square root calculations. This does not involve an approximate representation with floating point arithmetic. An exact representation of square root numbers can be obtained through a symbolic representation. This section is devoted to the definition of one representation for constructible numbers and to the definition of the arithmetic used with this representation.

In subsection 4.1 a representation for constructible numbers is defined; in subsection 4.2 and 4.3 known algorithms are given for the four basic arithmetic operations. In subsection 4.4 a trivial square root operation is presented. In subsection 4.5 an original definition of nullity and positivity are given. In subsection 4.6 examples are given showing the appropriateness of this representation for exact calculations Some limitations are also point out.

4.1 Representation of constructible numbers

Suppose we have a sequence of square root calculation, then numbers belong to the following quadratic extension of Q :

- k_0=Q,
- $k_1 = k_0[\sqrt{\alpha_0}]$ where $\alpha_0 \in k_0, \alpha_0 \geq 0$, α_0 is the first square root introduced during the calculations.
- $k_n = k_{n-1}[\sqrt{\alpha_{n-1}}]$ where $\alpha_{n-1} \in k_{n-1}, \alpha_{n-1} \geq 0$, α_{n-1} is the last square root introduced during the calculations .

Let $A \in k_n$, we write $A : (a_1, a_2)$ where $a_1, a_2 \in k_{n-1}$ when $A = a_1 + a_2 * \sqrt{\alpha_{n-1}}$.

For example, in $Q[\sqrt{2}][\sqrt{1+\sqrt{2}}]$, the real number $e_0 : ((5,2),(3,1))$ represents the value : $e_0 = 5 + 2\sqrt{2} + (3 + 1\sqrt{2})\sqrt{1+\sqrt{2}}$

In the following paragraphs, rational numbers will be represented as simple numerical values; constructive numbers, like $e_0 : ((5,2),(3,1))$, will be represented as a Prolog binary trees, like [[5, 2], [3, 1]]. The current field used for calculation, Q will be represented as [], and sequence of quadratic extension $Q[\sqrt{2}][\sqrt{1+1\sqrt{2}}]$ is represented by a Prolog list of constructible numbers, here [[1, 1]|[2]].

4.2 Addition and Subtraction

Given $A, B \in k_n$, with $A : (a_1, a_2)_{k_n} = a_1 + a_2\sqrt{\alpha_{n-1}}$ and $B : (b_1, b_2)_{k_n} = b_1 + b_2\sqrt{\alpha_{n-1}}$. Then $(A + B) : (a_1 + b_1, a_2 + b_2)$ since $A + B = a_1 + b_1 + (a_2 + b_2)\sqrt{\alpha_{n-1}}$. Let us distinguish additions 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.

We could define $(A-B) : (a_1 - b_1, a_2 - b_2)$, but this definition is superfluous : unification between $(A-B)$ and $(a_1 - b_1, a_2 - b_2)$ is equivalent to the unification of A with $(A-B)+B$. This definition stands with CLP, thanks to logic programming with linear constraints, as the constraints involved for the unification of A with $(A-B)+B$ are all linear constraints.

The predicate `ng_plus(E,A,B,C)`, true if `C` is equal to `A+B` in the extension `E` and predicate `ng_minus(E,A,B,C)` true if `C` is equal to `A-B` are 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).

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

4.3 Multiplication and Division

Given $A, B \in k_n$ where $A : (a_1, a_2)_{k_n} = a_1 + a_2\sqrt{\alpha_{n-1}}$ and $B : (b_1, b_2)_{k_n} = b_1 + b_2\sqrt{\alpha_{n-1}}$. Then $(A*B) : (a_1*b_1 + a_2*b_2*\alpha_{n-1}, a_1*b_2 + a_2*b_1)$. Distinguishing operations in k_n and operations in k_{n-1}, we get : $A *^n B : (a_1 *^{n-1} b_1 +^{n-1} a_2 *^{n-1} b_2 *^{n-1} \alpha_{n-1}, a_1 *^{n-1} b_2 +^{n-1} a_2 *^{n-1} b_1)$, where the operation $*^0$ denotes usual multiplication in $\mathbb{Q}$.

We may define $(A/B) = A * B^{-1}$ where $B^{-1} : (b_1/(b_1^2 - b_2^2\alpha_{n-1}), -b_2/(b_1^2 - b_2^2\alpha_{n-1}))$, but this definition is superfluous too. Unification between (A/B) and $A*B^{-1}$ is equivalent to the unification between A with $(A/B)*B$. This definition stands with CLP since the system of constraints, on the coordinates of A, B and (A/B), equivalent to the unification $A = (A/B) * B$ is a linear system.

The predicate `ng_mult(E,A,B,C)`, true if `C` is equal to `A*B` in the extension `E` and predicate `ng_div(E,A,B,C)` true if `C` is equal to `A/B` 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).

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

4.4 Square root

Given $A \in k_n$. We define $\sqrt{A}$ by $(0,1)_{k_{n+1}}$ where $k_{n+1} = k_n[\sqrt{A}]$.

The predicate `ng_sqrt(L,A,L',S)`, true if `S` in `L'` is equal to $\sqrt{A}$ when `A` is in the extension `L`, can be described by :

```
ng_sqrt(L,A,[A|L],[Z,0]) :-
   zero(L,Z)
   one(L,0).
```

4.5 Positivity and Nullity

Given $A \in k_n$ with $A : (a_1, a_2)_{k_n} = a_1 + a_2\sqrt{\alpha_{n-1}}$. Number A is positive in the following cases :

- if a_1 and a_2 are positive because $(a_1 \geq 0, a_2 \geq 0 \rightarrow a_1 + a_2\sqrt{\alpha_{n-1}} \geq 0)$
- if a_1 is positive, a_2 is negative and $a_1^2 - a_2^2\alpha_{n-1}$ is positive, because $(a_1 \geq 0, a_2 \leq 0) \rightarrow (a_1 + a_2\sqrt{\alpha_{n-1}} \geq 0 \iff a_1^2 \geq a_2^2\alpha_{n-1})$
- if a_2 is positive, a_1 is negative and $a_2^2\alpha_{n-1} - a_1^2$ is positive.

4.6 Examples

The representation and arithmetic given in section 4.1, through,4.5 may be used to set mixed constraint system over constructible numbers. The linear constraint solver and the quadratic solver are obviously correct over constructible number in this representation, that is a first important step compared to the linear constraint solvers over real represented as floating point number which is not correct nor complete. But the algorithms are not complete as multiple representations of one number are possible. For example the following system fails : $3 = sqrt(9)$

We consider in this section two examples on symbolic manipulation involving constructible numbers. The first example shows the advantages of the symbolic representation, the second example shows some limitations which can be eliminated.

First Example 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 $\mathrm{Q}[\sqrt{2}]$

The predicate `series(U,N)`, true if `U` is the element `N` of the series is given by :

```
series(U,N) :-
  ng_series(E,N,U,V,W).

ng_series([[2]],3,[0,1],[2,0],[1,0]).
```

```
ng_series(E,N,U,V,W) :-
  ng_series(E,N-1,V,W,X),
  ng_mult(E,V,W,Z),
  ng_plus(E,X,Z,U).
```

Calculation of the U_n series, with our representation gives :

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} : $(769533792308902217329427 2, 544142578273187201946014 51)$
U_{12} : $(1172005312243417792577922713561480518962771$
$, 828732903874311492000197638627684580540604)$

Significant growth of the coefficients is observed. This growth is common in symbolic calculation. The calculations have been executed in less than 0.1 second with a Mac II SI and Prolog III.

Let us compare the basic manipulations of algebraic numbers with those of a symbolic processor like Mathematica or Maple; the calculation of the U_n series using Mathematica yields :

$$U_3 = 2\sqrt{2} + 1$$
$$U_4 = \sqrt{2}(2\sqrt{2} + 1) + 2$$
$$U_5 = (\sqrt{2}(2\sqrt{2} + 1) + 2)(2\sqrt{2} + 1) + \sqrt{2}$$
$$U_6 = ((\sqrt{2}(2\sqrt{2} + 1) + 2)(2\sqrt{2} + 1) + \sqrt{2})(\sqrt{2}(2\sqrt{2} + 1) + 2) + 2\sqrt{2} + 1$$

Some limitations become apparent. They depend on the strategy used for calculation. When simplification are privileged, computing time becomes important, when efficiency is desirable memory consumption is neglected. In both cases, the calculation of the U_n series fails to compute U_{20} on a Mac II SI in less than 30 minutes.

Second example How is represented $A = \sqrt{2} + \sqrt{3}$ and $B = \sqrt{5 + 2\sqrt{6}}$ taken from [Bor85] ? We calculate successively :

$$\begin{aligned}
R_0 &: (0,1) \text{ in } Q[\sqrt{2}] \\
&= \sqrt{2} \\
R_1 &: ((0,0),(1,0)) \text{ in } Q[\sqrt{2}][\sqrt{3}] \\
&= \sqrt{3} \\
R_2 &: (((0,0),(0,0)),((1,0),(0,0))) \text{ in } Q[\sqrt{2}][\sqrt{3}][\sqrt{6}] \\
&= \sqrt{6} \\
R_3 &: (((5,0),(0,0)),((2,0),(0,0))) \text{ in } Q[\sqrt{2}][\sqrt{3}][\sqrt{6}] \\
&= 5 + 2\sqrt{6} \\
\text{And then, } R_5 &: ((((0,0),(0,0)),((0,0),(0,0))),(((1,0),(0,0)),((0,0),(0,0)))) \\
&\quad \text{in } Q[\sqrt{2}][\sqrt{3}][\sqrt{6}][\sqrt{5 + 2\sqrt{6}}] \\
&= B \quad = \sqrt{5 + 2\sqrt{6}} \\
\text{and, } R_6 &: ((((0,1),(1,0)),((0,0),(0,0))),(((0,0),(0,0)),((0,0),(0,0)))) \\
&\quad \text{in } Q[\sqrt{2}][\sqrt{3}][\sqrt{6}][\sqrt{5 + 2\sqrt{6}}] \\
&= A \quad = \sqrt{2} + \sqrt{3}
\end{aligned}$$

One can observe the following limitations :

- an exponential growth of the representation size with the number of square root extractions.
- a significant difficulty to prove equality between expressions ($A = B$ indeed !). As a consequence an additional algorithm would be necessary to verify overconstraints.

In fact, this section defined the constraints system (D,O,R), where D is the domain of fixed size binary trees representing constructible numbers, O is the set of operators $(+, -, *, /)$ on these trees, and R is the set of relational predicates $(=, \geq)$. The definition of this constraint systems is only partially correct because equality is not equivalent to unification. The next section considers the constraints system (D',O',R') where D' is the domain of fixed minimum size binary trees representing constructible numbers, O=O', and R'=($\geq$). In (D',O',R') trees representing real numbers are compacted, and as a consequence, unification is equivalent to $=$.

Note that the problem of equality of expression is difficult to solve. Mathematica and Maple have identical behavior with expressions A and B : both of them numerically evaluate $(A - B)$ to 0 with floating point arithmetic, but evaluate the boolean $(A - B == 0)$ to false.

5 Normal representation for constructible numbers

The representation proposed in section 4 may be definitely improved if quadratic extensions are only introduced when necessary. This is the principal and original part of this work.

Suppose we have a sequence of quadratic extensions of Q :

- k_0=Q,
- $k_1 = k_0[\sqrt{\alpha_0}]$ where $\alpha_0 \in k_0$, $\sqrt{\alpha_0} \notin k_0$. We suppose that the introduction of $\sqrt{\alpha_0}$ has been necessary to express the square root of an element $\mathcal{A}_0$ of k_0 which is not a square in k_0 . Note that in this condition $(1, \sqrt{\alpha_0})$ constitutes a basis of the vector space k_1 over k_0.
- we set $k_n = k_{n-1}[\sqrt{\alpha_{n-1}}]$ where $\alpha_{n-1} \in k_{n-1}$, $\sqrt{\alpha_{n-1}} \notin k_{n-1}$. We suppose that the introduction of $\sqrt{\alpha_{n-1}}$ has been necessary to express the square root of an element $\mathcal{A}_{n-1}$ of k_{n-1} which is not a square in k_{n-1}. Note that in this condition $(1, \sqrt{\alpha_{n-1}})$ constitutes a basis of the vector space k_n over k_{n-1}.

Let $A \in k_n$. Since k_n is a vector space over k_{n-1} and that $(1, \sqrt{\alpha_{n-1}})$ is a basis of that vector space, then there are unique elements $a_1, a_2 \in k_{n-1}$ such that $A = a_1 + a_2 * \sqrt{\alpha_{n-1}}$. The representation of A is unique. It is a normal representation.

The first improvement in this new definition concerns the uniqueness of the representation. This establishes a correspondence between equality defined for constructible numbers and unification defined for Prolog binary trees and rational numbers representing constructible numbers. So, overconstraint systems on constructible numbers are verified without supplementary work as overconstraint systems on rational numbers and binary trees are. As a consequence, completeness and correctness of the linear constraint solver over constructible numbers are preserved in this normal representation. An other direct consequence of this normal representation is provided by the following predicate `ng_null(E,N)`, true if value of `N` is zero in the sequence of extension `E`, defined by :

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

5.1 Square root

The normal definition of the quadratic extension of k_n needs to change the algorithm given in subsection 4.4 with the following :

Let $A \in k_n$, we want to calculate the square root of A, i.e. $\sqrt{A}$.

If A is a square in k_n, i.e. $A = a^2$ with $a \in k_n$, 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 : $k_{n+1} = k_n[\sqrt{\alpha_n}]$ where $\alpha_n = A$ and we get $\sqrt{A} : (0, 1)_{k_{n+1}}$.

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 $a^2 = A$? Consider the following cases :

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

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

Then a square root of A exists if and only if X and Y have a square root in N. Thus, if $X = x^2, Y = y^2$ then $A = a^2$ with $a = x/y$, (we always choose the positive root). Calculation of integer square root can be found in [Rol87].

$n > 0$.

Then $a = a_1 + a_2 * \sqrt{\alpha_{n-1}}$ 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 :

$$A = (x + y * \sqrt{\alpha_{n-1}})^2 \quad (0)$$

Rewriting this equation as an equation in vector space k_n over k_{n-1} we get :

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

We have to consider different cases in order to solve this non-linear system of 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-1} 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}.

$a_2 \neq 0$.

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

$$y^4\alpha_{n-1} - a_1y^2 + (a_2)^2/4 = 0 \quad (2)$$

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

$$Y^2\alpha_{n-1} - a_1Y + (a_2)^2/4 = 0 \quad (3)$$

whose discriminant is : $\Delta = (a_1)^2 - (a_2)^2\alpha_{n-1}$.

Δ must be a square in k_{n-1}. [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 $\Delta = (\pm 2\alpha_{n-1}Y - a_1)^2$, with $\pm 2\alpha_{n-1}Y - a_1 \in k_{n-1}$, (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}, then solutions for Y are :

$$Y_1 = (a_1 + \delta)/2\alpha_{n-1}, \quad \text{and} Y_2 = (a_1 - \delta)/2\alpha_{n-1}$$

Then square root y_1 et y_2, of Y_1 et Y_2 in k_{n-1} are calculated by recurrence. 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.

5.2 Examples with normal representation

Uniqueness of the representation Let us consider A and B defined in 4.6 with the new representation model. We compute successively :

$$\begin{array}{lrcc}
R_0 : & (0,1)_{Q[\sqrt{2}]} & = & \sqrt{2} \\
R_1 : & ((0,0),(1,0))_{Q[\sqrt{2}][\sqrt{3}]} & = & \sqrt{3} \\
R_2 : & ((0,1),(1,0))_{Q[\sqrt{2}][\sqrt{3}]} & = & \sqrt{2}+\sqrt{3} \\
R_3 : & ((0,0),(0,1))_{Q[\sqrt{2}][\sqrt{3}]} & = & \sqrt{6} \\
R_4 : & ((5,0),(0,2))_{Q[\sqrt{2}][\sqrt{3}]} & = & 5+2\sqrt{6} \\
R_5 : & ((0,1),(1,0))_{Q[\sqrt{2}][\sqrt{3}]} & = B = & \sqrt{5+2\sqrt{6}} \\
R_6 : & ((0,1),(1,0))_{Q[\sqrt{2}][\sqrt{3}]} & = A = & \sqrt{2}+\sqrt{3}
\end{array}$$

First we observe a diminution in the size of the representation. Secondly, $A = B$ is now quite trivial to check. The overconstraint $A = B$ is reduced without any extra algorithm.

Survey of 512 theorems in Euclidean geometry Two evaluations conclude this article. First we have tested whether Euclidean configurations are really hard or not. If they need too many quadratic extensions the exponential growth of the representation of the constructible numbers will be a difficult task to cope with. The second evaluation will test time needed for calculation with constructible numbers in normal representation.

A survey over the 512 theorems of Euclidean geometry proved in [Chou88], shows that :

- at least 479 theorems concern geometrical configurations where all coordinates may be rational numbers,
- at most 29 theorems concern geometrical configurations where coordinates need one quadratic extension,
- only 4 theorems need two quadratic extensions to represent coordinates of all the elements.

Therefore, most of the geometrical configuration do not require quadratic extension, just rational numbers. Almost all square root calculations done in geometrical configuration, do not introduce quadratic extension, but remain rational numbers. But in few cases, an exact representation, with quadratic extension is required, for example with equilateral triangle (1 quadratic extensions), or regular pentagon (2 quadratic extensions).

From a performance point of view, we have expressed and solved 30 configurations found in [Chou88] grouped in 3 sets. The first group contains the 10 configurations where Chou's demonstrator spends the least time to prove the corresponding theorem. The second group corresponds to the 10 times between the 250 and the 260 time for Chou's demonstrator. And last group corresponds to the hardest theorems.

We obtain the following tables, where "Ex." stands for the number given in [Chou88] for the problem, "Tms" is the time needed for the calculation, and "Ext." is the number of quadratic extensions involved.

Group 1.			Group 2.			Group 3.		
Ex.	Tms	Ext.	Ex.	Tms	Ext.	Ex.	Tms	Ext.
173	3 ms	0	260	5 ms	0	440	24 ms	0
86	3 ms	0	90	6 ms	0	80	28 ms	0
141	6 ms	0	110	350 ms	1	393	330 ms	1
140	5 ms	0	339	11 ms	0	17	23 ms	0
193	7 ms	0	116	12 ms	0	144	30 ms	0
92	6 ms	0	336	10 ms	0	347	31 ms	0
121	4 ms	0	434	13 ms	0	401	28 ms	0
181	7 ms	0	82	6 ms	0	396	35 ms	0
31	6 ms	0	330	13 ms	0	453	18 ms	1
37	6 ms	0	506	15 ms	0	316	40 ms	0

In group 1. the times to compute the configuration and to check numerically the theorem are 100 times faster than Chou's prover times. In the second group times are 1000 times faster except for 110 which is only 10 times faster. In the last group, speed-up are even more important. For 316, Chou's prover needs 20058s.

Comparisons between numerical evaluations and proving times may seem a bit tricky. But it is worth to recall that "Given a geometric proposition, we can easily present a concrete numerical example such that in order to determine whether the proposition is generally true, one need only to try this example" from [Hon86]. Within the perspective of an Euclidean prover based on numerical evaluations, the proposed comparison is worth considering.

From both theoretical and experimental point of view, the representation seems appropriate.

6 Conclusion and further works

We have proposed a normal representation for constructible number occuring in Euclidean geometry. This normal representation is compatible with linear constraint solver and preserves correctness and completeness over constructible numbers. The normal representation is also compatible with the quadratic constraint solver presented briefly in section 3. We have shown on 512 examples that the complexity of the representation is not too high in general cases. We have shown on 30 runtime examples that times consuming are closer to floating point representation times, than to symbolic representation prover times. In consequence, this approach seems appropriate. It is already used in one Intelligent Tutoring System for geometry, see [All93].

Our next goal is to introduce this exact arithmetic over constructible number in a Euclidean Prover based on numerical evaluations, cf. [Hon86, Dav77,

Chou92]. In future work we also plan to define a notion of constructible numbers and a normal representation over finite fields, cf. [Nau85]

Open problem The main result of this paper relies on the possibility to find explicit square root in algebraic extension of Q with square roots. Can this be extended to root of arbitrary degree ?

References

[All93] Allen, R., Idt, J., Trilling, L., *Constrained based automatic construction and manipulation of geometric figures*, Proceedings of the 13th IJCAI Conference, Chambery, Morgan Kaufmann Publishers, Los Altos, 1993.

[Bor85] Borodin, A., Fagin, R., Hopcroft, J.E., Tompa, M., *Decreasing the nested depth of expression involving square roots*, Journal of Symbolic Computation, no. 1 page 169-188, 1985.

[Chou92] S.C. Chou and X.S. Gao, *Proving Geometry Statement of Constructive Type*, CADE, LNCS 607, D. Kapur Eds, 1992.

[Chou88] S.C. Chou, *Mechanical Geometry Theorem Proving*, D. Reidel Publishing Company, 1988.

[Col90] A. Colmerauer, *An Introduction to Prolog III*, Commun. ACM, 28(4):412-418,1990.

[Col93] A. Colmerauer, *Naive Solving of Non-linear Constraints*, Constraint Logic Programming : Selected Research, F. Benhamou and A. Colmerauer eds, The MIT Press, page 89-112, 1993.

[Dav77] Davis, P.J., *Proof, Completeness, Transcendental, and Sampling.* Journal of the ACM, Vol. 24, no. 2, pages 298-310, 1977.

[Din88] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. *The Constraint Logic Programming Language CHIP*, In Proceedings of the Internationnal Conference on 5 Fifth Generation Computer Systems, Tokyo, Japan, December 1988.

[Jaf92] J. Jaffar, S. Michaylov, P-J. Stuckey, , and R. Yap, *The CLP(R) Language and System. ACM Trans. on Programming Languages and Systems*, 14(3):339-395,1992.

[Hon86] J. Hong, *Proving by Example and Gap Theorems* 27th An. Symp. on Foundations of Computer Science, Toronto, Ontario, Canada, IEEE press, p107-116, Oct 1986.

[Lab95] J.M. Laborde, *Des connaissances abstraites aux réalités artificielles, le concept de micromonde Cabri*, IV^{èmes} journées EIAO de Cachan, ed Eyrolles, 1995.

[Lan92] Landau, S., *Simplification of nested radicals,* SIAM Journal of Computing Vol 21, no.1 page 85-110, February 1992.

[Leb92] Lebesgue, H., *Leçons sur les constructions géométriques,* réédition, Gauthier-Villard, Paris, 1989.

[Nau85] Naudin, P., Quitté, C. *Algorithmique algébrique* Collection Logique mathématiques informatique, édition Masson, pages 312–324, 1985.

[Pes95] Pesant, G., *Une approche géométrique aux contraintes arithmétiques quadratiques en programmation logique avec contraintes* Thèse de l'Universite de Montréal, 1995.

[Rol87] Rolfe, T., *On a fast integer square root algorithm.* ACM SIGNUM Newsletter, Vol. 22, no. 4, pages 6–11, 1987.

This article was processed using the LaTeX macro package with LLNCS style