A Rule-Constructive Geometric Constraint Solver

R. Juan-Arinyo and Antoni Soto
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Av. Diagonal 647, 8a, E–08028 Barcelona

e-mail: {robert, toni}@lsi.upc.es

May 1995

Abstract

This work reports on the implementation of a two-dimensional, variational geometric constraint solver based on a constructive approach. The solver computes a solution in two phases. First, using rewrite rules, the solver builds a sequence of construction steps. Then, the construction steps are carried out to generate an instance of the geometric object for the current dimension values. We discuss some issues concerning the data representation and the rules used. Then a simple example illustrates how the solver works. Finally we give a correctness proof of the solver.

1 Introduction

Constraints are an important enabling technology for interactive graphics applications. One of such applications where the use of constraints is growing more rapidly is in geometric modelling. The goal is to develop sketching systems based on geometric constraints solvers in which the user defines a rough sketch, annotated with dimensions and constraints, and the system builds an instance that satisfies all constraints.

Several approaches to the geometric constraint solving problem have been reported in the literature. Among them, the constructive technique is one of the most promising approaches. In this class of constraint solvers the constraints are satisfied constructively, by placing geometric elements in some order. They are based on the fact that most configurations in an engineering drawing are solvable by using a rather small set of tools like ruler, compass and protractor.

This paper reports on the work developed while building a prototype of a geometric constraint solving system (GCSS). We consider only well-constrained, two-dimensional sketches formed from points and lines.

The GCSS has two major components, the *solver* and the *valuator*. The solver deals with the problem of determining symbolically whether or not a geometric sketch is solvable. It is based on a constructive approach which exhibits properties of both rule- and graph-constructive approaches. The solver is fed with a dimensioning scheme; i.e. a topologically correct sketch properly annotated with a set of constraints. Then, if the set of constraints consistently defines the object, the solver generates a sequence of constructive steps that determine each geometric element such that the constraints are satisfied. The valuator responds to the problem of building an instance of the geometric object. The instantiation is carried out by applying the sequence of construction steps generated by the solver to the actual parameters values. Whenever no numerical incompatibilities arise in the computation, an instance of the geometric object is generated. By clearly separating in this way the symbolic computation from numerical computation, we can use tools specifically tailored for each commitment. As a result, the interactive performance of the whole GCSS is improved.

We demonstrate that given a dimensioning scheme which consistently defines a geometric object, the solver, considered as a rewrite system, is canonical, i.e. the system has a unique normal form that is obtained by finitely many reduction steps.

2 A Formalization of the 2D Problem

It is well know that the relative position of n given points $\{p_1, p_2, \dots, p_n\}$ in the bidimensional Euclidian space, are determined by $2n-3$ independent relations defined between the points. In what follows we will use the words *relation* and *constraint* indistinctly. We shall call a *dimensioning scheme* to a given set of points and the set of constraints defined between them.

We assume that a given set of n points on which a dimensioning scheme has been defined, can be split into two nonempty disjoint subsets. One subset, $\vec{p'} = \{p'_1, p'_2, \dots, p'_k\}$, contains all those given points with fixed position. The other subset, $\vec{p} = \{p_1, p_2, \dots, p_l\}$, contains all those points with unknown position. Note that this decomposition is always possible because $2n-3$ relations define a rigid body with three remaining degrees of freedom, two of them corresponding to a translation and the third one corresponding to a rotation. Hence, the absolute position for at least one given point should be specified.

The set of relations or constraints in the dimensioning scheme together with logical conjunction, disjunction and negation allow us to express the dimensioning scheme of a sketch by a formula $\varphi(\vec{p'}, p_1, \dots, p_l)$, [4]. Then, provided that the set of constraints is consistent, the formula

$$\exists p_1, \dots, \exists p_l \mid \varphi(\vec{p'}, p_1, \dots, p_l)$$

holds. By the *axiom of choice*, [4, 11], this is equivalent to say that

$$\exists p_1, \dots, \exists p_l \mid \varphi(\vec{p'}, p_1, \dots, p_l) \Rightarrow \exists f_1, \dots, \exists f_l \mid \varphi(\vec{p'}, f_1(\vec{p'}), \dots, f_l(\vec{p'}))$$

Hence, the goal in solving a geometric constraint problem is to prove the truth of the above formula, and to evaluate the functions $f_1, \dots, f_l$.

As we shall see in Section 4, our approach to geometric constraint solving is such that given the set of points $\{p_1, p_2, \dots, p_n\}$ and a dimensioning scheme $\varphi(\vec{p'}, p_1, \dots, p_l)$ defined on them, we search a *constructive* formula, $\Psi(\vec{p'}, p_1, \dots, p_l)$, such that if the constraints define the geometric object in space consistently we will be able to figure out the relative position of each point. If the predicate $pos(p_i, \{x_i, y_i\})$ assigns the position $\{x_i, y_i\}$ to point p_i, an example of the constructive formula would be [4],

$$\begin{aligned}\Psi(\vec{p'}, p_1, \dots, p_l) \quad = \quad & pos(p'_1, \{x_1, y_1\}) \wedge \dots \wedge pos(p'_k, \{x_k, y_k\}) \\ & \wedge pos(p_1, \{fx_1(\vec{p'}), fy_1(\vec{p'}\}) \\ & \bigwedge_{i=2}^{l} pos(p_i, \{fx_i(\vec{p'}, p_1, \dots, p_{i-1}), fy_i(\vec{p'}, p_1, \dots, p_{i-1})\})\end{aligned}$$

3 Approaches to Geometric Constraint Solving

In general *Numerical Constraint Solvers*, the constraints are translated into a system of algebraic equations and are solved using an iterative method like the Newton–Raphson method. *Constructive Constraints Solvers* are based on the fact that most configurations in an engineering drawing are solvable by using a rather small set of tools like ruler, compass and protractor. The constraints are satisfied constructively by placing geometric elements in some order. *Rule-constructive* and *graph-constructive* solvers are two different versions of the constructive approach. In *Propagation Methods* the constraints are first translated into a system of equations involving variables and constants. Then an undirected graph is created whose nodes are the equations, variables and constants, and whose edges represent the occurrence of a variable or a constant in an equation. The method then attempts to direct the graph edges so that every equation can be solved in turn, initially only from the constants. *Symbolic Constraint Solvers* transform the constraints into a system of algebraic equations; the system is solved with symbolic algebraic methods such as Gröbner bases. For a thorough review of approaches to geometric constraint solving see [2] and [7].

4 The Solver

The solver handles bidimensional geometric configurations composed from points and segments. The constraints that can be defined on those objects include distance between two points, perpendicular distance between a point and a segment, angle between two segments, incidence, perpendicularity, and parallelism.

The solver is *variational*, i.e., the solver processes the constraints without the need of arranging them in a predefined ordering sequence. Furthermore, the solver can deal with dimensioning schemes with circular constraints.

Our basic method for solving geometric constraints is a constructive approach and exhibits properties of both rule- and graph-constructive approaches, [2]. In a first phase, our solver uses rewrite rules to build a sequence of construction steps which, in some sense, represents the constructive formula given in Section 2 while each step is a function $f \in \{fx_i, fy_i, 1 \leq i \leq l\}$, in the formula. In a second phase, the construction steps are carried out to generate an instance of the geometric object for the current dimensions values. Let us see now how the data is represented and which rules are available.

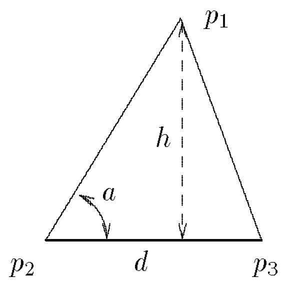


Figure 1: Triangle specified by a side length d, an angle a and an altitud h.

4.1 Data Representation

The data representation used is based on the representation reported in [14] and [15], but differs from them in some rather important details. In [15] it is assumed that the distance constraint perpendicular to a segment defined by the user has been translated into distance and angle constraints before calling the solver. Let us show that this assumption has a major drawback. For a complete discussion of this issue see [10].

Assume that the user has sketched and constrained a triangle as shown in Figure 1. The altitud constraint can be translated into distance and angle constraints in several ways by adding an extra point and two more relations that constraint it. Figure 2 shows some possible translations.

Among all the possible translations of the altitud constraint to distance and angle constraints, there is no a unic translation which allows to solve all the triangles specified by dimensioning schemes where one or more altituds are included, applying just the set of rules defined in [15]. As an example, Figure 2 shows several constraint translations for the dimensioning scheme in Figure 1; only the translation in Figure 2c is solvable with the rules given in [15].

This means that, in the framework of these constructive rules, the constraint translation process is context-dependent, i.e. the dimensioning scheme defined by the user should be recognized in order to translate it to a proper representation. Note that this is equivalent to having available a specific rule for each situation where the distance constraint perpendicular to a segment is used. Since this constraint is so often used, we have included a specific representation for it. Therefore, besades the **CD** sets and the **CA** sets given in [14] and [15], we include the **CH** sets. These sets are defined as follows.

A **CA** set is a pair of oriented segments which are mutually constrained in angle.

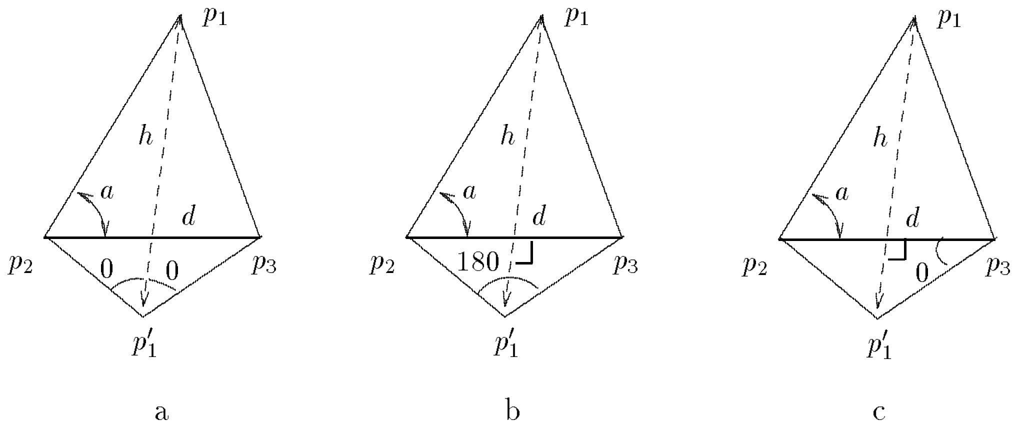


Figure 2: Some possible translations of distance constraint perpendicular to a segment.

A **CD** set is a set of points with mutually constrained distances. A frame of reference is attached to each **CD** set to which the points in the set are refered to. It is worth to note that a sketch is solved when all the points in it belong to the same **CD** set. This **CD** set is conceptually the result of applying the constructive formula in Section 2.

A **CH** set is a point and a segment constrained by the perpendicular distance from the point to the segment.

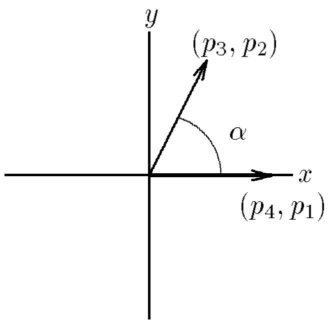
In what follows, we will refer generically to the **CD**, **CA** and **CH** sets as *constraints sets.*

4.2 Rules

In order to deal with the **CH** sets, we have consistently extended the rules given in [15]. Depending on the functionality of the rules, we classify them as belonging to one of the following types: creation rules, merging rules or construction rules.

4.2.1 Creation rules

Creation rules create elementary **CD**, **CA** and **CH** sets as an interpretation of the dimensioning scheme defined by the user. The sign of the distances and angles are defined based on what the user has sketched. When an angle



$$angle((p_4, p_1), (p_3, p_2)) = \alpha$$

$$\Downarrow$$

$$ca_0 = \{((p_4, p_1), 0), ((p_3, p_2), \alpha)\}$$

Figure 3: Creation of an elementary CA set.

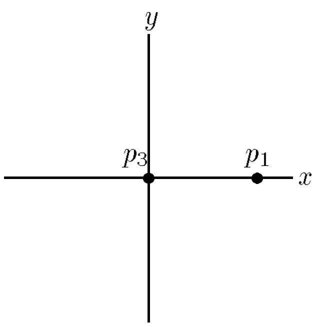
constraint between two directed segments is placed, a **CA** set is created. See Figure 3.

When a distance constraint between two points is given, a **CD** set is created. The position of the points in the associated frame of reference are $(0, 0)$ and $(d, 0)$. See Figure 4.

Whenever a point, a segment and the perpendicular distance from the point to the segment are given, a **CH** set is created. See Figure 5.

4.2.2 Merging rules

Only one rule belongs to this type. See Figure 6. When a segment belongs to two different **CA** sets, ca_1 and ca_2, a new **CA** set, ca_3, is created which constrains



$$distance(p_3, p_1) = d_1$$

$$\Downarrow$$

$$cd_0 = \{(p_3, (0, 0)), (p_1, (d_1, 0))\}$$

Figure 4: Creation of an elementary CD set.

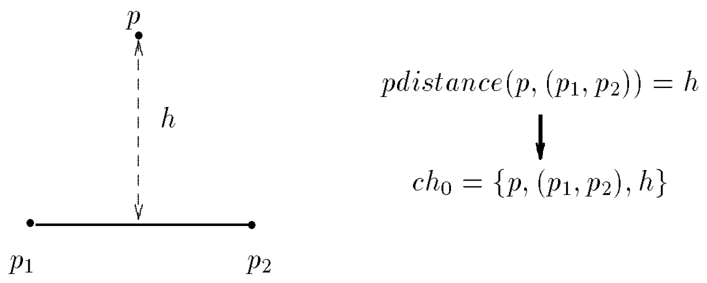


Figure 5: Creation of an elementary CH set.

in angle two segments, one from ca_1 and one from ca_2, both segments being different from the segment shared by ca_1 and ca_2.

This rule allows to compute the transitive closure of the angle constraints set.

4.2.3 Construction rules

Construction rules merge **CD** sets, **CH** sets and **CA** sets into larger **CD** sets. Merging is performed by building triangles and a few quadrilaterals. Consistently with the constraints that the solver can handle, triangles should be specified by a set of three independent relations each of them being either a side-length, an angle between two sides or an altitud. A complete description of each

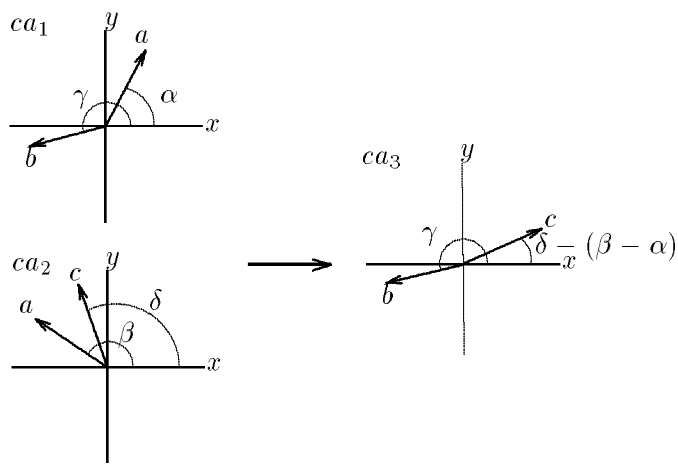


Figure 6: Merging two CA sets.

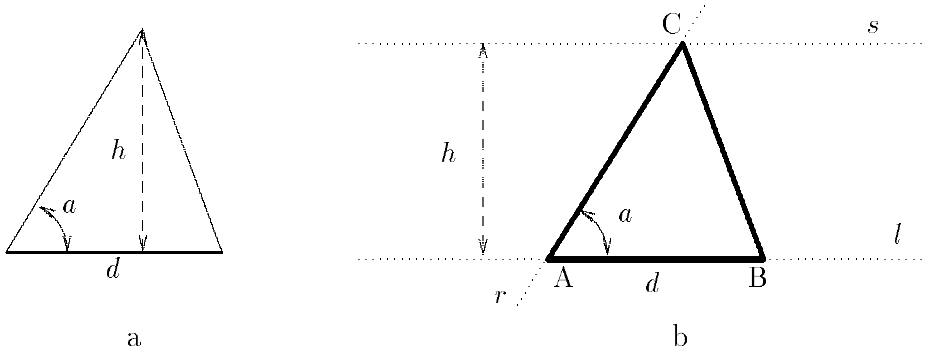


Figure 7: Construction of a triangle. a) Given constraints. b) Construction rule.

rule can be found in [9, 10]. As an example, Figure 7a shows a triangle specified by a sidelength, an angle and an altitud, and Figure 7b shows the associated construction rule. The four quadrilateral rules are those reported in [15] and can also be found explained with detail in [10].

4.3 Implementation

Figure 8 shows the architecture of our Geometric Constraints Solving System (GCSS). The geometric constraint solving problem is split into two main components: the *solver* and the *valuator*.

The solver determines whether the dimensioning scheme is or is not symbolically solvable. If it is solvable, the solver generates a sequence of construction steps equivalent to the constructive formula we were looking for in Section 2. Then, the valuator instantiates the geometric object by applying the sequence of construction steps generated by the solver to the actual parameters values. The main reasons that lead us to this architecture are:

- The nature of the computations in each step are quite different. The solver requires symbolic computation while the valuator only performs numerical computations.
- Determining whether the problem can be symbolically solved or not can be done in the solver step and it does not depend neither on the actual parameters values nor on the geometric computations.
- With the proposed uncoupling, when computing instances for different parameters values, only the second step needs to be carried out. This allows

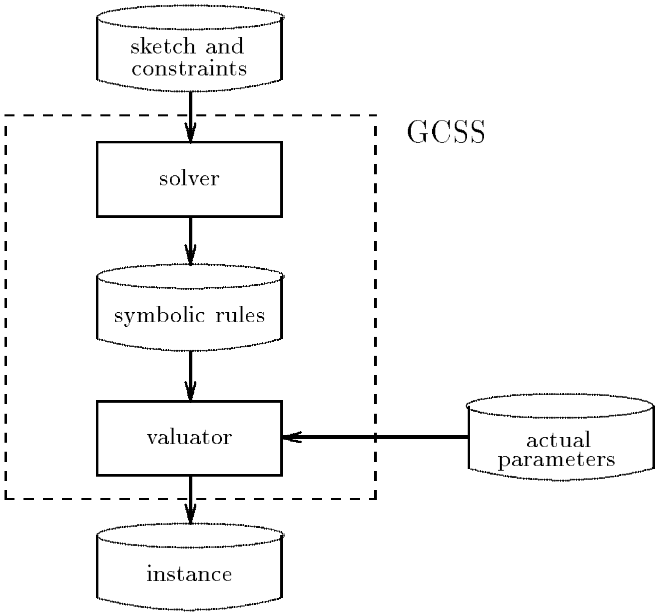


Figure 8: System Architecture.

to avoid the solver step which is computationally much more expensive.

- Given a symbolically solvable dimensioning scheme and a set of actual values for the parameters, the object can be instantiated if there are not numerical impossibilities. These impossibilities are detected while carrying out the geometric computations.

The solver is an expert system programmed in Prolog [3, 5]. We have chosen this programming language because of the symbolic computation needed and because it is a rapid prototyping language. The sequence of construction steps generated by the solver is automatically encoded as a TSC toolkit program, [13]. The TSC toolkit provides the tools for building easily a graphic user interface. The geometric object is generated by executing the TSC program.

5 Case Study

In order to illustrate how the system works, let us present here a case study. Assume that the user has sketched and constrained a pentagon as shown in Figure 9a. The initial set of constraints is given in Figure 10 and has been derived from the dimensioning scheme defined by the user.

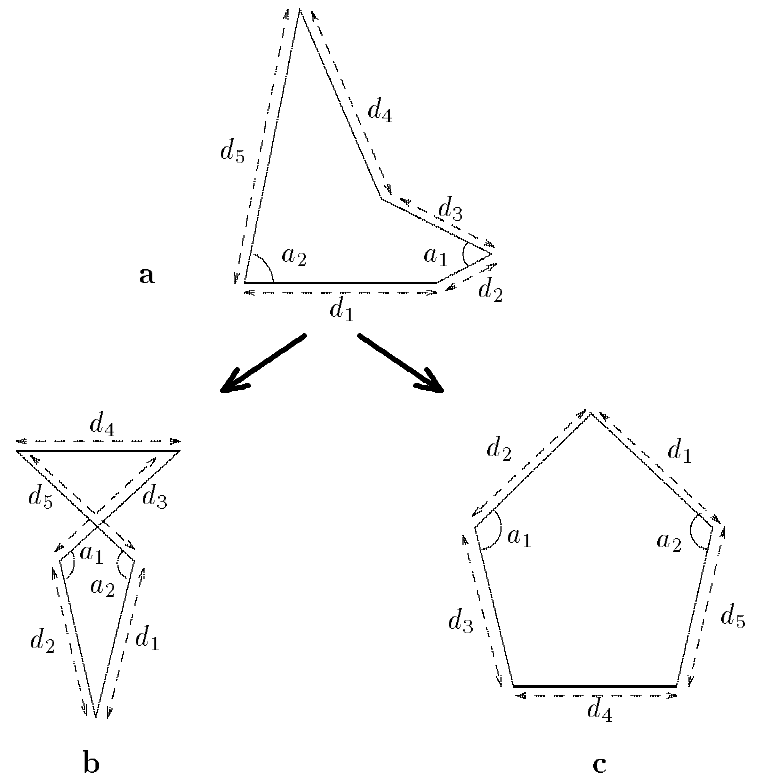


Figure 9: Case Study. a) Sketch and dimensionig scheme. b) and c) Instances generated with $d_1 = d_2 = d_3 = d_4 = d_5 = 100$ and $a_1 = a_2 = 120°$.

The solver processes the set of initial constraints until either all the points belong to the same **CD** set or no rule applies. The data representation evolves as follows.

1. The **CD** set creation rule is applied to constraints 1, 2, 3, 5, and 6 giving the **CD** constraints sets

$$\begin{aligned} cd_0 &= \{v_1, v_2\} \\ cd_1 &= \{v_2, v_3\} \\ cd_2 &= \{v_3, v_4\} \\ cd_3 &= \{v_4, v_5\} \\ cd_4 &= \{v_5, v_1\} \end{aligned}$$

2. The **CA** set creation rule is applied to constraints 4 and 7 giving the

$$
\begin{aligned}
distance(v_1, v_2) &= d_1 && (1)\\
distance(v_2, v_3) &= d_2 && (2)\\
distance(v_3, v_4) &= d_3 && (3)\\
angle((v_3, v_4), (v_3, v_2)) &= a_1 && (4)\\
distance(v_4, v_5) &= d_4 && (5)\\
distance(v_5, v_1) &= d_5 && (6)\\
angle((v_1, v_2), (v_1, v_5)) &= a_2 && (7)
\end{aligned}
$$

Figure 10: Case Study. Initial constraints.

constraints sets

$$
\begin{aligned}
ca_0 &= \{(v_3, v_4), (v_3, v_2)\}\\
ca_1 &= \{(v_1, v_2), (v_1, v_5)\}
\end{aligned}
$$

Note that, in this case, the set of angle constraints and its transitive closure coincide.

3. Now the rule that merges two **CD** sets can be triggered on cd_1, cd_2 and ca_0. Constraints sets cd_1, cd_2 and ca_0 are replaced with the **CD** set

$$cd_5 = \{v_2, v_3, v_4\}$$

4. Again the rule that merges two **CD** sets can be triggered on cd_0, cd_4 and ca_1. Constraints sets cd_0, cd_4 and ca_1 are replaced with the **CD** set

$$cd_6 = \{v_1, v_2, v_5\}$$

5. Finally, a constructive rule which builds a triangle given three **CD** sets with three pairwise common points can be triggered with the constraints sets cd_3, cd_5 and cd_6. These **CD** sets are replaced with

$$cd_7 = \{v_1, v_2, v_3, v_4, v_5\}$$

The last step yields a **CD** set to which all the points in the original sketch belong to. This fact tells the solver that the solving process is over. Then, the TSC program in Figure 11 is generated from the set of construction steps.

The TSC program is written using TSC commands. It has a procedure called `instantiate` and a command called `reader`. The main difference between

```
PROCEDURE instantiate(
        REAL d1, REAL d2, REAL d3, REAL d4, REAL d5, REAL a1, REAL a2)
WITHIN DEF IS
BEGIN
        relDist("cd0", "v1", "v2", d1);
        relDist("cd1", "v2", "v3", d2);
        relDist("cd2", "v3", "v4", d3);
        relDist("cd3", "v4", "v5", d4);
        relDist("cd4", "v5", "v1", d5);
        relAngle("ca0", "v3v4", "v3v2", a1);
        ruleS2("cd5", "v3", "v3v2", "v3", "v2",
               "v3v4", "v3", "v4",
               "cd1", "cd2", "ca0");
        relAngle("ca1", "v1v2", "v1v5", a2);
        ruleS2("cd6", "v1", "v1v2", "v1", "v2",
               "v1v5", "v1", "v5",
               "cd0", "cd4", "ca1");
        ruleT1("cd7", "v5", "v4", "v2", "cd3", "cd5", "cd6");
ENDPROCEDURE

COMMAND reader WITHIN DEF IS
        REAL d1;
        REAL d2;
        REAL d3;
        REAL d4;
        REAL d5;
        REAL a1;
        REAL a2;
BEGIN
        d1 := ReadReal("d1");
        d2 := ReadReal("d2");
        d3 := ReadReal("d3");
        d4 := ReadReal("d4");
        d5 := ReadReal("d5");
        a1 := SexaToRadians(ReadReal("a1"));
        a2 := SexaToRadians(ReadReal("a2"));
        instantiate(d1, d2, d3, d4, d5, a1, a2);
ENDCOMMAND
```

Figure 11: Case Study. TSC program.

commands and procedures is that TSC commands can be called from the process using the TSC toolkit while TSC procedures can not. The command `reader` reads the actual values for the parameters and calls the procedure `instantiate` who, in turn, calls the procedures in the underlaying geometric engine to perform the geometric computations.

6 Solver Correctness

The idea behind solvers based on geometric rewrite rules is to replace some facts in the database by *simpler* ones, [4, 6]. In our solver, initially, the **CD** and **CH** constraints sets, represent the sets of point-point and point-segment distance constraints derived from the dimensioning scheme while the **CA** sets are the transitive closure of the angle constraints defined by the user. The solver starts by applying the rules to these initial sets. Then the rules are repeatedly applied to the resulting constraints sets until either there is only one **CD** set which contains all the points in the sketch or no rule applies. In the first case the resulting **CD** set is a solution whereas in the second case the dimensioning scheme either does not define the geometric object consistently or is not solvable with the available set of rules. Every time a rule is triggered we will say that a reduction step has been performed.

Assuming that the dimensioning scheme does define the geometric object consistently, two things should be proved in order to prove the solver correctnes: 1) the algorithm applying the rules stops in any case, and 2) the sequence in which the rules are applied does not matter for the result. In next section we recall some of the rules available in our solver as rewrite rules. Then we shall prove the solver correctness.

6.1 Rewriting Rules

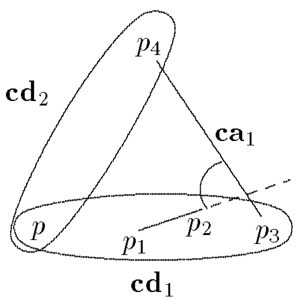
In rewriting theory, a rule over a set of terms $\mathcal{T}$ is an ordered pair $< l, r >$ of terms, which are usually written as $l \to r$. It is said that l rewrites or reduces to r, [1, 6, 12]. All the construction rules in our system can be expressed as rewrite rules. The whole set of rewrite rules can be found in [10]. Let us recall some of them just as an example.

The rule that merges two **CD** sets that share a point and are constrained in angle can be expressed as

$$\{\mathrm{cd}_1, \mathrm{cd}_2, \mathrm{ca}_1\} \longrightarrow \{\mathrm{cd}'\}$$

where

$$\begin{aligned} \mathrm{cd}_1 &= \{p, p_1, p_2, p_3, \cdots\} \\ \mathrm{cd}_2 &= \{p, p_4, \cdots\} \\ \mathrm{ca}_1 &= \{(p_1, p_2), (p_3, p_4)\} \end{aligned}$$



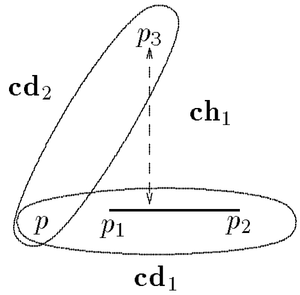
and

$$\mathrm{cd}' = \mathrm{cd}_1 \cup \mathrm{cd}_2$$

The following rule merges two **CD** sets such that share a point and are constrained by a **CH**

$$\{\mathrm{cd}_1, \mathrm{cd}_2, \mathrm{ch}_1\} \longrightarrow \{\mathrm{cd}'\}$$

where



$$\begin{aligned}
\mathrm{cd}_1 &= \{p, p_1, p_2, \cdots\} \\
\mathrm{cd}_2 &= \{p, p_3, \cdots\} \\
\mathrm{ch}_1 &= \{p_3, (p_1, p_2)\}
\end{aligned}$$

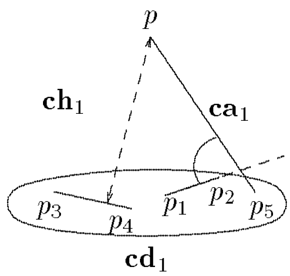
and

$$\mathrm{cd}' = \mathrm{cd}_1 \cup \mathrm{cd}_2$$

Finally, the rule that locates a point constrained by a **CD** set, a **CA** set and a **CH** set is

$$\{\mathrm{cd}_1, \mathrm{ca}_1, \mathrm{ch}_1\} \longrightarrow \{\mathrm{cd}'\}$$

where



$$\begin{aligned}
\mathrm{cd}_1 &= \{p_1, p_2, p_3, p_4, p_5, \cdots\} \\
\mathrm{ca}_1 &= \{(p, p_5), (p_1, p_2)\} \\
\mathrm{ch}_1 &= \{p, (p_3, p_4)\}
\end{aligned}$$

and

$$\mathrm{cd}' = \mathrm{cd}_1 \cup \{p\}$$

The set of all the rewriting rules in our system will be denoted by $\rightarrow_R$.

6.2 Termination Properties

Let us define a *term* as a set whose members are constraints sets. Let $\mathbf{C}_0$ be the initial term with

$$\mathbf{C}_0 = \cup_i \mathbf{CD}_i \cup_j \mathbf{CH}_j \cup_k \mathbf{CA}_k$$

$\mathbf{CD}_i$ and $\mathbf{CH}_j$ are, respectively, the sets of point-point and point-segment distance constraints derived from the dimensioning scheme defined by the user. $\cup_k \mathbf{CA}_k$ is the transitive closure of the angle constraints sets defined by the user. Let us call *a class of angles* to each maximal set of angle constraints derived from the dimensioning scheme on which the transitive property holds.

Lemma 6.1 Let n be the number of given constraints in a dimensioning scheme that consistently defines a geometric object. Then an upper bound of the number of constraints sets in $\mathbf{C}_0$ is $n + \frac{1}{2}n^3$.

Proof Each constraint in the dimensioning scheme generates either one **CD** set or one **CH** set or one **CA** set. This gives n constraints sets. Let nc be the number of classes of angles in the dimensioning scheme and let n_i be the number of angle constraints in class i. Then the number of extra **CA** sets generated by the transitive closure is given by

$$\sum_{i=1}^{nc} \frac{n_i(n_i-1)}{2} = \frac{1}{2}\sum_{i=1}^{nc} n_i(n_i-1) \leq \frac{1}{2}\sum_{i=1}^{nc} n_i^2 \leq \frac{1}{2}\sum_{i=1}^{nc} n^2 \leq \frac{1}{2}n^3$$

□

Let us denote by $\mathbf{C}_i$ the term whose members are the constraints sets after having applied the i-th reduction step. Then, the pair $(\mathbf{C}_i, \rightarrow_R)$ is a *reduction system*, [12]. We shall make use of the following definitions.

Definition 6.1 A *derivation* in $(\mathbf{C}_i, \rightarrow_R)$ is any sequence of applications of rewrite rules in $(\mathbf{C}_i, \rightarrow_R)$. We will denote a derivation by $\stackrel{*}{\rightarrow}$.

Definition 6.2 A term $\mathbf{C}'$ is *derived* from $\mathbf{C}$ if and only if there exists a derivation such that $\mathbf{C}\stackrel{*}{\rightarrow}\mathbf{C}'$. A term $\mathbf{C}$ to which no rule in $(\mathbf{C}_i, \rightarrow_R)$ applies is called *irreducible* or *normal form.*

Next, we will show that the algorithm applying the rules stops in any case; i.e. the reduction system $(\mathbf{C}_i, \rightarrow_R)$ has a *uniform termination.* Algorithms which exhibit this property are also known as *Noetherian* or *well-founded* algorithms, [1, 6].

Lemma 6.2 The reduction system $(\mathbf{C}_i, \rightarrow_R)$ is Noetherian.

Proof Let $\mathbf{CR}_l$ be a set of constraints sets such that $\mathbf{CR}_l \subset \mathbf{C}_i$ and let $\mathbf{CR}_l \rightarrow \mathbf{CR}_r$ be a reduction step. Then $\mathbf{C}_i \rightarrow \mathbf{C}_{i+1}$ where $\mathbf{C}_{i+1} = (\mathbf{C}_i - \mathbf{CR}_l) \cup \mathbf{CR}_r$. Note that, as it is shown in [10], for all the rules in $\rightarrow_R$ the term $\mathbf{CR}_l$ has three or more components while $\mathbf{CR}_r$ is a singleton.

Let n be the number of given constraints that consistently define the geometric object. By Lemma 6.1, the number of constraints sets in $\mathbf{C}_0$ is less or equal than $n + \frac{1}{2}n^3$. Each reduction step reduces the number of sets in term $\mathbf{C}_i$ in at least by two. Therefore $\mathbf{C}_0$ is reduced in at most $(n + \frac{1}{2}n^3 + 1)/2$ steps to an irreducible term. □

Let us state a lemma that relates the rules that build quadrilaterals with the other rules. Figure 12a shows an example of a quadrilateral rule and Figure 12b the equivalent pentagon.

Lemma 6.3 Each problem solved by a quadrilateral rule in $\rightarrow_R$ is equivalent to a problem derived from the original one by adding an auxiliar point and two more constraints taken from the original dimensioning scheme. Furthermore, each quadrilateral rule in $\rightarrow_R$ can be defined as a sequence of already defined rules where the quadrilateral rules themselves and the **CA** merging sets rule are not involved.

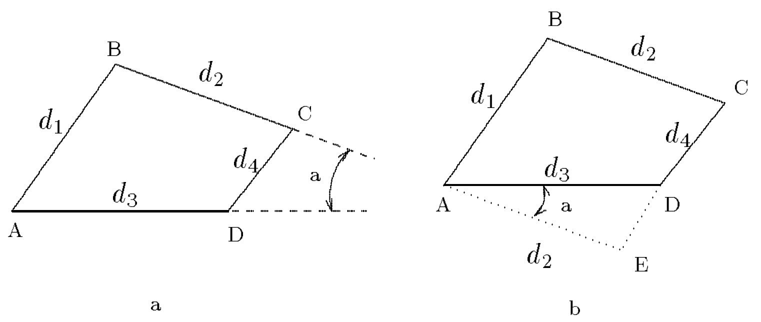


Figure 12: A quadrilater and the equivalent pentagon.

The lemma is proved in [10]. Now all the construction rules $\mathbf{CR}_l \rightarrow \mathbf{CR}_r$, in $\rightarrow_R$, are such that $\mathbf{CR}_l = \{\mathbf{CX}_1, \mathbf{CX}_2, \mathbf{CX}_3\}$ and $\mathbf{CR}_r$ is a **CD** set.

Two questions are now in order. First we should point out that constructing quadrilaterals as a sequence of already defined rules, does not make use of the **CA** sets merging rule, [10]. This and the fact that the **CA** sets merging rule is used only initially to compute the transitive closure of the **CA** constraints sets, allow us to exclude this rule from any rational about derivations without loss of generality. The second question is that the extra point used in each quadrilateral rule does not entail any further treatment; this extra point will be just ignored because it neither belongs to the object the user inteded to design nor is needed in further constructions.

The definitions and proofs that follow are developed using the techniques given in [8]. Given a dimensioning scheme, we define an induced subscheme as a subset of points of the scheme and the subset of constraints defined just on this subset of points. In what follows assume that a set with n points and a dimensioning scheme, $\varphi(\vec{p'}, p_1, \ldots, p_l)$, defined on them are given.

Definition 6.3 The dimensioning scheme $\varphi(\vec{p'}, p_1, \ldots, p_l)$ is overconstrained if there is a subscheme induced by $m \leq n$ points such that the number of constraints is greater than $2m - 3$.

Definition 6.4 A dimensioning scheme $\varphi(\vec{p'}, p_1, \ldots, p_l)$ is underconstrained if it is not overconstrained and the number of constraints is less than $2n - 3$.

Definition 6.5 A dimensioning scheme $\varphi(\vec{p'}, p_1, \ldots, p_l)$ is well-constrained if it is not overconstrained and the number of constraints is equal to $2n - 3$.

Lemma 6.4 Assume that the dimensioning scheme from which $\mathbf{C}_0$ has been derived is well-constrained. If $\mathbf{C}_0 \overset{*}{\rightarrow} \mathbf{C}$ then the subscheme induced by each

CD set in **C** is well-constrained.

Proof First note that rules $\{\mathbf{CX}_1, \mathbf{CX}_2, \mathbf{CX}_3\} \rightarrow \mathbf{CD}$ where $\mathbf{CX}_1, \mathbf{CX}_2$ and $\mathbf{CX}_3$ are either **CA** or **CH** sets always generate a new **CD** set made up of three points, [10]. Since the number of constraints defined on them is exactly three, the induced subscheme is well-constrained.

Let us now consider the rules with at least one **CD** set in their left side. We shall prove the lemma by induction on the length of the derivation $\xrightarrow{*}$. For the induction basis, $\mathbf{C} = \mathbf{C}_0$ and by hypothesis all the initial **CD** sets are well-constrained.

For the induction step, assume that the **CD** sets in term $\mathbf{C}_i$ after having applied the i-th reduction step are well-constrained, and let n_j be the number of points in $\mathbf{CD}_j$. Then each induced subscheme has $2n_j - 3$ relations.

Finally, for the next reduction step we need to study three different situations. First we consider the rule $\{\mathbf{CD}_1, \mathbf{CD}_2, \mathbf{CD}_3\} \rightarrow \mathbf{CD}$. Since

$$| \mathbf{CX}_1 \cap \mathbf{CX}_2 | \quad = \quad | \mathbf{CX}_2 \cap \mathbf{CX}_3 | \quad = \quad | \mathbf{CX}_3 \cap \mathbf{CX}_1 | \quad = \quad 1$$

the number of elements in the rewritten **CD** set is $n = n_1 + n_2 + n_3 - 3$ and the number of relations defined on it is given by $(2n_1 - 3) + (2n_2 - 3) + (2n_3 - 3) = 2(n_1 + n_2 + n_3 - 3) - 3 = 2n - 3$.

Next we consider the family of rules with exactly two **CD** sets in the left side, $\{\mathbf{CD}_1, \mathbf{CD}_2, \mathbf{CX}_3\} \rightarrow \mathbf{CD}$. If $\mathbf{CX}_3 = \mathbf{CA}_3$, then, [10], either

$$| \mathbf{CD}_1 \cap \mathbf{CD}_2 |= 1, \quad | \mathbf{CD}_1 \cap \mathbf{CA}_3 |= 1, \quad | \mathbf{CD}_2 \cap \mathbf{CA}_3 |= 3$$

or

$$| \mathbf{CD}_1 \cap \mathbf{CD}_2 |= 1, \quad | \mathbf{CD}_1 \cap \mathbf{CA}_3 |= 2, \quad | \mathbf{CD}_2 \cap \mathbf{CA}_3 |= 2$$

In both cases the number of points in the rewritten **CD** set is $n_1 + n_2 + n_3 - 5$. Since $n_3 = 4$, the number of points is $n_1 + n_2 - 1$. The number of constraints is $(2n_1 - 3) + (2n_2 - 3) + 1 = 2(n_1 + n_2 - 1) - 3$. If $\mathbf{CX}_3 = \mathbf{CH}_3$, then, [10], either

$$| \mathbf{CD}_1 \cap \mathbf{CD}_2 |= 1, \quad | \mathbf{CD}_1 \cap \mathbf{CH}_3 |= 1, \quad | \mathbf{CD}_2 \cap \mathbf{CH}_3 |= 2$$

or

$$| \mathbf{CD}_1 \cap \mathbf{CD}_2 |= 1, \quad | \mathbf{CD}_1 \cap \mathbf{CH}_3 |= 2, \quad | \mathbf{CD}_2 \cap \mathbf{CH}_3 |= 1$$

In both cases the number of points in the rewritten **CD** set is $n_1 + n_2 + n_3 - 4$. Since $n_3 = 3$, the number of points is $n_1 + n_2 - 1$. The number of constraints is $(2n_1 - 3) + (2n_2 - 3) + 1 = 2(n_1 + n_2 - 1) - 3$.

Finally, when the family of rules has just one **CD** set in the left side, the rewritten **CD** set is $\mathbf{CD}_1 \cup \{p\}$; that is, a new point is added to the initial **CD** set, [10]. If the subscheme induced by $\mathbf{CD}_1$ has n_1 points and $2n_1-3$ constraints defined on them, the number of points in the rewritten $\mathbf{CD}$ set is n_1+1 and the number of constraints in the induced subscheme is $2n_1-3+2=2(n_1+1)-3$. $\square$

Lemma 6.5 Let $\mathbf{C}_0 \xrightarrow{*} \mathbf{C}$. If there are constraints sets in $\mathbf{C}$ such that $\mid \mathbf{CD}_i \cap \mathbf{CD}_j \mid \geq 2$ with $i \neq j$, or $\mathbf{CA}_i \subset \mathbf{CD}_j$ or $\mathbf{CH}_i \subset \mathbf{CD}_j$, for some i, j, then the dimensioning scheme defined on $\mathbf{C}_0$ is overconstrained.

Proof Assume that $\mid \mathbf{CD}_i \cap \mathbf{CD}_j \mid = m \geq 2$. By Lemma 6.4, both $\mathbf{CD}_i$ and $\mathbf{CD}_j$ are well-constrained and the number of constraints in the induced subschemes are $2n_i-3$ and $2n_j-3$, respectively. Consider the subscheme induced by $\mathbf{CD}_i \cup \mathbf{CD}_j$. This subscheme is defined on $n=n_i+n_j-m$ points and has $(2n_i-3)+(2n_j-3)=2(n_i+n_j)-6=2n+2m-6$ constraints. But $m \geq 2$, then $2n+2m-6 \geq 2n-2 > 2n-3$.

As before, by Lemma 6.4, the number of points in the subscheme induced by $\mathbf{CD}_j$ has n_j points and $2n_j-3$ constraints. But $\mathbf{CA}_i \subset \mathbf{CD}_j$ or $\mathbf{CH}_i \subset \mathbf{CD}_j$ means that the number of constraints defined on the set of points in $\mathbf{CD}_j$ is actually $(2n_j-3)+1=2n_j-2>2n_j-3$. $\square$

Now we will show that, in the reduction system $(\mathbf{C}_i, \rightarrow_R)$, the sequence in which the rules are applied does not matter for the result; that is, different reduction processes starting from the same dimensioning scheme always terminate at the same irreducible term. These reduction systems are called systems with *unique termination* or *Church-Rosser*, [1, 6, 12].

Lemma 6.6 If the dimensioning scheme defines consistently the geometric object, then the reduction system $(\mathbf{C}_i, \rightarrow_R)$ is Church-Rosser.

Proof To prove the lemma sufficies to show that every pair of rules commute, [6]. Let $\mathbf{C}_0$ be the initial set of constraints sets such that consistently define the geometric object and $\mathbf{C}_i$ be a term such that $\mathbf{C}_0 \xrightarrow{*} \mathbf{C}_i$. Furthermore, assume that there are two different rules that can be applied, $\mathbf{C}_i \rightarrow_1 \mathbf{C}_1$, involving the constraints sets $\mathbf{CX}_{11}, \mathbf{CX}_{12}, \mathbf{CX}_{13} \in \mathbf{C}_i$ and $\mathbf{C}_i \rightarrow_2 \mathbf{C}_2$ involving the constraints sets $\mathbf{CX}_{21}, \mathbf{CX}_{22}, \mathbf{CX}_{23} \in \mathbf{C}_i$. Now we should show that $\mathbf{C}_1 \rightarrow_2 \mathbf{C}_3$, and $\mathbf{C}_2 \rightarrow_1 \mathbf{C}_3$.

Four different cases can be distinguished depending on how many distinct constraints sets there are among the sets $\mathbf{CX}_{ij}$.

1. Case three different sets. The only possibility where two different rules can be applied is when two of the constraints sets are $\mathbf{CD}$ sets with more than one point in common, [10]. This means that the dimensioning scheme must be overconstrained.

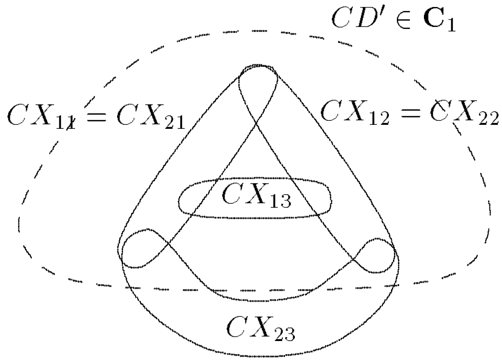


Figure 13: Four different constraints sets.

2. Case four different sets. Consider rule $\rightarrow_1$ and let $\mathbf{CX}_{23}$ be the set not used by the rule. Since each rule requires three distinct constraints sets, $\mathbf{CX}_{23}$ must have points in common with two different sets out of $\mathbf{CX}_{11}, \mathbf{CX}_{12}$ and $\mathbf{CX}_{13}$. See Figure 13. First assume that $\mathbf{CX}_{23}$ is a $\mathbf{CD}$ set. Then the term $\mathbf{C}_1$ has two $\mathbf{CD}$ sets, $\mathbf{CX}_{23}$ and the rewritten $\mathbf{CD}$ set, $\mathbf{CD}'$, that share more than one point and the dimensioning scheme is overconstrained. Assume now that $\mathbf{CX}_{23}$ is either a $\mathbf{CA}$ set ot a $\mathbf{CH}$ set. Every point in $\mathbf{CX}_{23}$ belongs to either $\mathbf{CX}_{11}$ or $\mathbf{CX}_{12}$ and $\mathbf{CX}_{23}$ is a subset of $\mathbf{CD}'$. Hence the dimensioning scheme is overconstrained. The same rational applies to rule $\rightarrow_2$.

3. Case five different sets. Here we consider two different subcases. First let us consider the subcase where the set shared by the two rules is a $\mathbf{CD}$ set; that is $\mathbf{CD}_{11} = \mathbf{CD}_{21}$. See Figure 14. Assume that both $\mathbf{CX}_{22}$ and $\mathbf{CX}_{23}$ are $\mathbf{CD}$ sets. Since it is assumed that the object is consistently defined and rule $\rightarrow_2$ applies, $\mathbf{CD}_{22} \cap \mathbf{CD}_{11} = \{p_1\}$ and $\mathbf{CD}_{23} \cap \mathbf{CD}_{11} = \{p_2\}$. Two different situations can arise now. If $(\mathbf{CD}_{22} \cup \mathbf{CD}_{23}) - \{p_1, p_2\}) \cap (\mathbf{CX}_{12} \cup \mathbf{CX}_{13}) = \emptyset$, then $\rightarrow_1$ and $\rightarrow_2$ commute. But if $(\mathbf{CD}_{22} \cup \mathbf{CD}_{23}) - \{p_1, p_2\}) \cap (\mathbf{CX}_{12} \cup \mathbf{CX}_{13}) \neq \emptyset$, then $\mathbf{C}_1$ and $\mathbf{C}_2$ must contain two $\mathbf{CD}$ sets that share more than one common point, contradicting that the dimensioning scheme is well-constrained. When only either $\mathbf{CX}_{22}$ or $\mathbf{CX}_{23}$ is a $\mathbf{CD}$ set, the same rational applies just considering $\mathbf{CX}_{22}$ or $\mathbf{CX}_{23}$ respectively. When neither $\mathbf{CX}_{22}$ nor $\mathbf{CX}_{23}$ are a $\mathbf{CD}$ set, rule $\rightarrow_2$ adjoins just a new point to the $\mathbf{CD}$ set rewritten by rule $\rightarrow_1$, [10]. If this point belongs to the $\mathbf{CD}$ set rewritten by rule $\rightarrow_1$ both $\mathbf{CX}_{22}$ and $\mathbf{CX}_{23}$ are subsets of the rewritten set and the scheme is overconstrained; otherwise the rules clearly commute.

   Second consider the subcase where the set common to both rules is either a $\mathbf{CH}$ set or a $\mathbf{CA}$ set and let us assume that it is $\mathbf{CX}_{11} = \mathbf{CX}_{21}$. See

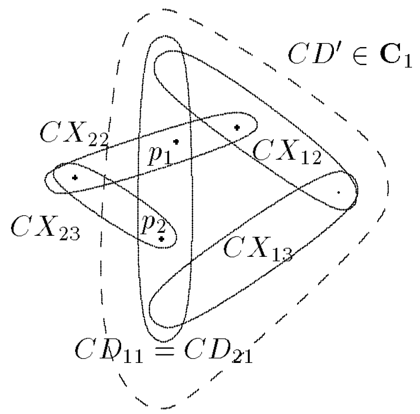


Figure 14: Five different constraints sets. The set shared by the rules is a **CD** set.

Figure 15. If the two constraints sets $\mathbf{CX}_{22}$ and $\mathbf{CX}_{23}$ are **CD** sets, at least one of them shares two points with the set common to both rules. Then term $\mathbf{C_1}$ has two **CD** sets that share more than one point and the scheme is overconstrained. If only either $\mathbf{CX}_{22}$ or $\mathbf{CX}_{23}$ is a **CD** set, then rule $\rightarrow_2$ is such that this **CD** set and $\mathbf{CX}_{11}$ always intersect in two or more points, [10]. Consequently, the dimensioning scheme should be overconstrained. When neither $\mathbf{CX}_{22}$ nor $\mathbf{CX}_{23}$ are **CD** sets, rule $\rightarrow_2$ is such that the number of points involved is exactly three and all of them belong to every

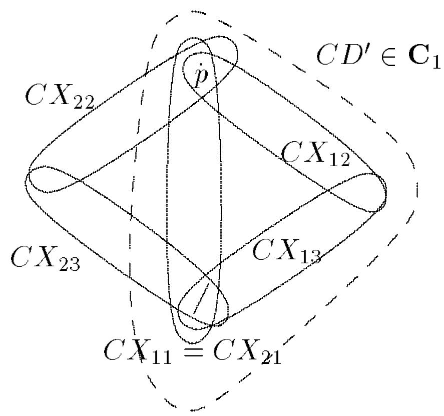


Figure 15: Five different constraints sets. The set shared by the rules is either a **CH** set or a **CA** set.

constraints set, [10]. Then $\mathbf{CX}_{22}$ and $\mathbf{CX}_{23}$ are subsets of the $\mathbf{CD}$ set rewritten by rule $\rightarrow_1$ and the dimensioning scheme is overconstrained.

4. Case six different sets. Since the constraints sets are all distinct, the rules $\rightarrow_1$ and $\rightarrow_2$ commute.

$\square$

Thus, we have proved the following theorem:

Theorem 6.7 If the dimensioning scheme defines the geometric object consistently, the reduction system $(\mathbf{C}_i, \rightarrow_R)$ is a terminating Church-Rosser system, that is, the reduction system $(\mathbf{C}_i, \rightarrow_R)$ is *canonical.*

7 Conclusion and Future Work

We have presented the work developed while building a prototye of a variational geometric constraint solver based on the constructive approach which exhibits properties of both rule- and graph-constructive approaches. We have proved that given a dimensioning scheme which consistently defines a geometric object, the solver, considered as a rewrite system, is canonical, i.e. the system has a unique normal form that is obtained by finitely many reduction steps.

Currently, our group is working on a number of topics to improve the solver. An immediate goal is to extend the solver in order to deal with a wider set of geometric elements and related constraints like arcs, circles, tangency and concentricity. We are also working on two open issues that are of capital importance in order to build useful Computer Aided Design systems based on geometric constraint solving. One issue concerns detecting dimensioning schemes which do not consistenly define a geometric object because the scheme is either underconstrained or overconstrained. In both cases the system should provide appropiate hints to help the user. The other concern refers to extend the logic capabilities of the solver following two concurrent directions 1) Given a subset of parameters with fixed values, figure out ranges of acceptable values for the remaining parameters, and 2) to extend the way of expressing constraints by including relations between variables.

References

[1] L. Bachmair. Proof methods for equational theories. Technical report, Department of Computer Science, SUNY at Stony Brook, Stony Brook, New York 11794, 1991.

[2] W. Bouma, O. Fudos, C. Hoffmann, J. Cai, and R. Paige. A geometric constraint solver. Technical report, CSD-TR-93-054, Purdue University, 1993.

[3] I. Bratko. *PROLOG. Programming for Artificial Intelligence. Second Edition.* Addison Wesley, 1990.

[4] B.D. Brüderlin. *Rule-Based Geometric Modelling.* PhD thesis, Institut für Informatik der ETH Zürich, 1988.

[5] Y. Cosmadopoulos and D. Chu. *IC Prolog II.* Deprt. of Computing, Imperial College, London, November 1992. Version 0.94, for Sun Workstations.

[6] N. Dershowitz and J.-P. Jouannaud. *Handbook of Theoretical Computer Science*, chapter Rewrite Systems, pages 244–320. Elsevier Science Publishers B.V., 1990.

[7] I. Fudos. Editable representations for 2D geometric design. Master's thesis, Purdue University, Department of Computer Sciences, 1993.

[8] I. Fudos and C.M. Hoffmann. Correctness proof of a geometric constraint solver. Technical Report CSD 93-076, Computer Sciences Department, Purdue University, December 1993.

[9] R. Juan-Arinyo. Triangles, ruler and compass. Technical Report LSI-95-6-R, Universitat Politècnica de Catalunya, LiSI, 1995.

[10] R. Juan-Arinyo and A. Soto. A set of rules for a constructive geometric constraint solver. Technical Report LSI-95-19-R, Universitat Politècnica de Catalunya, LiSI, 1995.

[11] S.C. Kleene. *Mathematical Logic.* John Wiley and Sons, New York, 1967.

[12] B.K. Rosen. Tree-manipulating systems and Church-Rosser theorems. *Journal of the ACM*, 20(1):160–187, January 1973.

[13] A. Soto and S. Vila. *TSC V1.2 User Manual.* Universitat Politècnica de Catalunya, Dept Llenguatges i Sistemes Informàtics, January 1994.

[14] G. Sunde. A CAD system with declarative specification of shape. *Eurographics Workshop on Intelligent CAD Systems*, pages 90–105, April 1987.

[15] A. Verroust, F. Schonek, and D. Roller. Rule-oriented method for parameterized computer-aided design. *Computer Aided Design*, 24(10):531–540, October 1992.