Implementing Non-Linear Constraints with Cooperative Solvers

Eric Monfroy, Michaël Rusinowitch, René Schott

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Implementing non-linear constraints with cooperative solvers

Eric Monfroy , Michael Rusinowitch , René Schott

N° 2747

Décembre 1995

PROGRAMME 2

ISSN 0249-6399

Implementing non-linear constraints with cooperative solvers

Eric Monfroy* , Michael Rusinowitch** , René Schott***

Programme 2 — Calcul symbolique, programmation et génie logiciel
Projet Protheo

Rapport de recherche n2747 — Décembre 1995 — 22 pages

Abstract: We investigate the use of cooperation between solvers in the scheme of constraint logic programming languages over the domain of non-linear polynomial constraints. Instead of using a general and often inefficient decision procedure we propose a new approach for handling these constraints by cooperating specialised solvers. Our approach requires the design of a client/server architecture to enable communication between the various components. The main modules are a linear solver, a non-linear solver, a constraint manager, a communication protocol component and an answer processor module.

This work is motivated by the need for a declarative system for robot motion planning and geometric problem solving. We have implemented a prototype called **CoSAc** (**Co**nstraint **S**ystem **A**r**c**hitecture) to validate our approach using cooperating solvers for non-linear constraints over the real numbers. Our language is illustrated by an example that also shows the advantages of cooperation.

Key-words: constraint solving, non-linear constraint, cooperation, integration

(Résumé : tsvp)

*CRIN-CNRS & INRIA Lorraine, BP 239, 54506 Vandœuvre-lès-Nancy Cedex, France. E-mail: Eric.Monfroy@loria.fr.

**INRIA Lorraine & CRIN-CNRS, 615, rue du Jardin Botanique, BP 101, 54602 Villers-lès-Nancy Cedex, France. E-mail: Michael.Rusinowitch@loria.fr.

***CRIN-CNRS & INRIA Lorraine, BP 239, 54506 Vandœuvre-lès-Nancy Cedex, France. E-mail: Rene.Schott@loria.fr.

Unité de recherche INRIA Lorraine
Technôpole de Nancy-Brabois, Campus scientifique,
615 rue de Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY (France)
Téléphone : (33) 83 59 30 30 – Télécopie : (33) 83 27 83 19
Antenne de Metz, technopôle de Metz 2000, 4 rue Marconi, 55070 METZ
Téléphone : (33) 87 20 35 00 – Télécopie : (33) 87 76 39 77

Résolution de contraintes non linéaires par coopération de solveurs

Résumé : Nous étudions la coopération de solveurs dans le cadre de la programmation logique à contraintes polynomiales. Plutôt que d'utiliser une procédure globale de décision qui s'avère souvent inefficace, nous proposons une nouvelle approche permettant de manipuler ces contraintes à l'aide de solveurs spécialisés coopérant entre eux.

Notre façon d'aborder le problème nécessite la conception d'une architecture client/serveur pour établir les communications entre les différents composants intégrés dans le système (un solveur linéaire, un solveur non linéaire, un logiciel de calcul formel, un administrateur des communications, un administrateur des contraintes, un protocole de communication et un module de traitement des solutions).

Les besoins d'un système déclaratif pour la planification de trajectoires et la résolution de problèmes géométriques ont motivé ces travaux. Afin de valider notre approche utilisant des solveurs coopérant pour résoudre les contraintes non linéaires sur les réels, nous avons implanté un prototype nommé **CoSAc**. Le langage est illustré d'exemples montrant les avantages de la coopération.

Mots-clé : résolution de contraintes, contrainte non linéaire, coopération, intégration

1 Introduction

Many applications such as computational geometry [27], robot motion planning [17, 18], geometry theorem proving [5] involve non linear geometrical objects. These problems are specified using non-linear polynomial constraints describing the space, the objects, their trajectories and the relations between them.

Consider for instance the problem of moving a rectangle through an angled corridor. A rectangle R has to move from an initial position I to a position F at the other end of the corridor C subject to the following constraints: no point of R collides with the walls of C during the motion. Several interesting questions may be asked about this problem: is a trajectory valid w.r.t. the given constraints? Does there exist a valid trajectory? How to compute a valid trajectory? Is it possible to modify a wrong trajectory in order to derive a correct one?

By providing declarativeness and expressiveness, a constraint logic programming language (CLP) (see [9] for a survey) for the domain of non-linear constraints provides a natural way to reason about geometric objects. CLP languages combine the advantages of logic programming with the power of constraint solving. So the CLP paradigm provides us with a natural way for integrating algebraic tools inside a declarative language. Thus the user can state and solve problems without any knowledge of the underlying algebraic methods.

All the previously mentioned applications require a CLP system that handles equalities, disequalities and inequalities composed of linear and non-linear polynomials with rational coefficients (even if these problems contain transcendental functions they can generally be converted to polynomial relations by standard techniques). Several CLP languages already manipulate non-linear constraints: CLP($\mathcal{R}$) [14], CAL [1], RISC-CLP($\mathcal{R}$eal) [13], Newton [2], the cooperative architecture of [19] and the CLP from [22]. Indeed they do not fit our motivation due to one or several of these properties: they delay non-linear constraints till they eventually simplify to linear ones; they do not handle inequations; they apply general decision procedures that are often inefficient; they use interval methods that require to define "good" initial intervals for the variables and that are usually unable to detect whether a system of constraints is inconsistent or not.

A general decision procedure, as Collins one [6], is far too inefficient for practical problems. Most of the time a large part of these problems can be handled by "uncomplete" but efficient solvers. For instance when a linear subproblem is isolated, it can be solved by an efficient implementation of the simplex algorithm (though it is not a solver for non-linear CLP, it can solve important subproblems). Cooperation of solvers [3, 28, 15] has emerged as an alternative approach which competes with quantifier elimination. Marti and Rueher [19] propose such a cooperative architecture which allows using concurrently heterogeneous solvers when handling constraints over the reals. Their architecture is based upon agents that communicate via asynchronous message passing.

In this paper we describe how we use cooperation techniques to design **CoSAc** (**Co**nstraint **S**ystem **A**r**c**hitecture), a CLP system handling non-linear polynomial constraints (equalities, disequalities and inequalities). Cooperating components allow one to tackle non-linear

constraints without the burden of a general decision procedure. This approach presents also other advantages such as the possibility to reuse existing tools, extensibility and easier maintenance [24].

To illustrate this paper, we show several queries and traces of execution for the following program:

```
zmul(R1, I1, R2, I2, R3, I3):-
        R3 - R1*R2 + I1*I2 =:= 0,
        I3 - R1*I2 - R2*I1 =:= 0.
```

It describes a circuit made of two parallel resistors [14]. Although this example is simple, it shows most of the advantages of cooperating solvers.

The rest of the paper is organized as follows. Section 2 discusses the need for cooperation in a CLP system and its advantages over a general solver. Then we present the components required to implement **CoSAc**: the platform, the solvers, the communication manager, the data exchange protocol module, the constraint manager, the answer processor module. To link these modules, **CoSAc** relies on a a client/server architecture (see section 4). Moreover exchanging information is one of the most important issue of cooperation. So we define the synchronized communication between processes (section 4.3) and the data exchange protocol (section 4.4) used in the implementation of **CoSAc**. Section 5 presents the progress of cooperation in **CoSAc**. First an example illustrates the effects and advantages of cooperation. Then we detail the routing of information through the different components. Some more examples are given in section 6 in order to illustrate the different aspects of **CoSAc**. Section 7 concludes with final remarks and present future works.

2 Motivation for a cooperative architecture

We define cooperation as follows: several solvers are applied to a **unique** computation domain (with possibly different languages) as opposed to combination [23, 16] where several domains are mixed. Each solver works on a part of the problems (generally these parts are not disjoint) and transmits its results to the other solvers. So they incorporate new information in order to complete their partial knowledge of the initial system. The computation terminates when no solver can deduce new information and when there is no more data exchange. A general decision procedure for the practical problems we have in mind would be far too inefficient and can be replaced advantageously by cooperating solvers.

2.1 Cooperation between solvers versus a general solver

Hence a linear solver (for equalities, disequalities and inequations) and a non-linear solver (for equalities) working together is a convenient method for increasing declarativity without jeopardizing efficiency:

- Declarativity nearly reaches the same level with cooperating solvers than with quantifier elimination method. Only non-linear inequations and disequations are not tackled. However in most applications, inequations and disequations are linear, for instance constraints on time or space parameters. Moreover it is always possible to transform them into equalities by adding slack variables.
- Inequations are used more dynamically during computation, not only at the final phase of a computation for checking the solutions (as with Gröbner bases [22]).
- The solvers working together are faster than quantifier elimination alone. Thus if no non-linear constraint is encountered during the resolution, there is no need of the non-linear solver: the linear solver is sufficient and is less expensive both in time and memory.
- New solvers can then be added to the first two solvers to increase declarativity and efficiency (solver specialised for certain classes of equations i.e. quadratics [26] for example). In order to integrate a new solver, two modules of **CoSAc** have to be extended: the functionalities of the new solver have to be declared in the communication manager (see section 3.2), and the constraint manager (see section 3.3) must know which constraints the new solver handles.

2.2 Cooperation with other tools

Symbolic computation facilities are required during several steps of the computation. In order to provide more declarativity, it is possible to define constraints with non expanded polynomials, or with derivatives of polynomials. Hence these constraints have to be modified in order to be processed by the solvers. Furthermore after Gröbner bases computation, some symbolic computation techniques are required for extracting solutions from the base.

That is the reason why a cooperation with a symbolic computation software is important. First it helps the user not to bother about transformation of polynomials when setting constraints. Furthermore it enables to use symbolic computation functionalities inside a program. Finally it offers a library of efficient algorithms (e.g. for polynomial roots computation) inside the language itself.

3 The components of CoSAc

CoSAc is a CLP system for non linear constraints based on cooperating solvers. So that the solvers can exchange data, **CoSAc** is required to manage the constraints and the solutions, and to synchronize the communications. We now describe the tools we have integrated in the modular architecture of **CoSAc**.

3.1 The platform: ECLiPSe

For integrating the cooperating tools, a development system is required. It has to be extensible, efficient and robust. The platform should support the development of new constraints solvers. That is why we have chosen ECLiPSe [20] which is the ECRC Common Logic Programming System. ECLiPSe presents all the required properties: it is a generic development system for CLP which already provides several libraries of constraint solvers.

3.2 The communication manager

This module controls the components of the system and synchronizes the communications. It is the only module knowing about the functionalities and the state of the several components. So when a component A sends a request (i.e. asks for a function to be applied to an input) the communication component finds the component B able to perform the function. But this is not the only task for the communication manager: it also launches the "requested" components (if not already running), it kills them when they are not needed anymore and it manages the list of running components (how to communicate with them).

3.3 The constraint manager

CHRs [8] define determinate conditional rewriting systems that specify how conjunctions of constraints propagate and simplify. CHRs have been integrated in ECLiPSe as an extension for writing user-defined constraints. Since the CHRs provide a framework for writing user-defined constraints, they are used in **CoSAc** to introduce the constraints and to plan their treatment (i.e. selection and distribution of the constraints to the solvers).

3.4 The data exchange protocol module

Since each tool admits its own representation (structure) for an object, this module performs all the conversion tasks when exchanging data (this component is detailed in section 4.4).

3.5 The solvers

The role of a solver is to solve/simplify the constraints it has received. Each solver works on a special domain, with specific constraints. We are using three solvers in **CoSAc**: the first one for linear constraints over the rationals, the second one for non-linear constraints and the third one for simplification of polynomials and the computation of roots of univariate polynomials.

Linear solver: the linear solver that is integrated into **CoSAc** is an equation solver over rational numbers (or real numbers) that has been implemented by T. Frühwirth with the CHRs. This solver is flexible and open for extension by more advanced tools.

Non linear solver: the non linear solver of **CoSAc**, is based on Gröbner bases computation. Though this technique provides us with a decision procedure over the complex numbers, it can also solve real constraints in many practical applications. Till now Gröbner bases computation was a slow procedure. But with the recent implementation of GB [7], it has become reasonable in time and space. GB is the fastest known system for Gröbner bases. It is divided into client/servers and each server is specialized for a particular computation. GB makes use of several algorithms such as the Sugar Cube Strategy [12]. Thus GB is the appropriate solver for non-linear constraints since it allows fast computation of large systems of polynomials.

Roots of univariate polynomials: the symbolic computation software Maple [11] is used to compute the roots of univariate polynomials. But Maple also simplifies and transforms polynomials before they are processed by GB or the equation solver embedded in the CHRs.

3.6 The answer processor module

During resolution, the solvers act in a cooperative way: they attempt to solve the constraints and exchange data. But at the end of a computation, their results have to be combined in order to build a complete solution and to provide the user with a "readable" answer. When working with non-linear constraints over the reals three different answers are possible: either the constraint store has no solution, or infinitely many solutions, or finitely many solutions. The computation proceeds according to these three cases:

- empty set of solution: then the Prolog engine backtracks as with standard resolution.
- finitely many solutions: the computed solutions are used to instantiate the variables. All the solutions are displayed using backtracking.
- infinitely many solutions: a canonical form of the solution is displayed. If the infinity is due to non-linear constraints, then it is not possible to assert that there exists some real solutions (as all the solutions may be complex numbers).

The next sections describe in details how the solutions are computed in each case.

3.6.1 Empty set of solutions

This case can either be detected by the linear solver or by the non-linear one. For instance this situation occurs when the GB module generates a Gröbner base that contains 1.

3.6.2 Infinitely many solutions

Both solvers can meet this situation. If this answer comes from the linear solver, then the non-linear solver is triggered (the addition of non linear constraints can lead to a system of constraints having finitely many solutions or no solution at all). If it is the non-linear

solver, then it is not possible to assert that there exists real solutions (Gröbner bases act as a decision procedure only on complex numbers). But even in this case Gröbner bases are worth computing as they produce a simplified and canonical form for the system of constraints. Furthermore relations between variables may be extracted from the bases. By setting a flag, the user can choose between two different (but equivalent) answers to a problem with infinitely many solutions: the result of the CHRs or the result of GB (a Gröbner base).

3.6.3 Finitely many solutions

This last case is the most complex one, although it can be detected from the syntactic structure of the Gröbner base.

Let GB_{lex} be the lexicographic Gröbner base of a set F of polynomials of $R[x_1, \ldots, x_n]$. Then F has finitely many solutions if for each variable x_i, there exists $c_i \in R$ and $k_i \in N$ such that $c_i * x_i^{k_i}$ is the head monomial of a polynomial of GB_{lex} [4].

This means that we obtain a triangular form of the system from which it is possible to produce the solutions i.e. GB_{lex} has the following form (with $x_n > x_{n-1} > \ldots > x_1$):

$GB_{lex} = \{p_1, \ldots, p_l, \ldots, p_k, \ldots, p_m, \ldots, p_p\}$
with

$p_1 \in K[x_1]$,
$p_2, \ldots, p_l \in K[x_1, x_2]$,
$p_{l+1}, \ldots, p_k \in K[x_1, x_2, x_3]$,
$\ldots$,
$p_{k+m}, \ldots, p_p \in K[x_1, x_2, \ldots, x_n]$

Hence it is possible to eliminate the variables one after the others. At this state of the computation, it is no more possible to apply further the solvers. Finding the roots of the successive univariate polynomials is obtained by numerical computation. Two different methods can lead to the extraction of the finitely many solutions. Moreover the user can select one of them when running a program by setting the flag `sol_extract` to 1 or 2.

Labelling with cooperation
Labelling is a usual technique of CLP languages (see [29] for more details), generally used in association with solvers over finite and discrete domains. It is a controlled way of making choice, usually called at the end of the computation for choosing values for the constrained variables.

Although we have an infinite and continuous domain, we can use labelling to process the solutions. Since at this state of the computation the only values a variable can take are roots of an univariate polynomials, we need only consider a discrete and finite domain.

As opposed to the next method, the constraint store is not modified with the Gröbner base. So the answer processor is working on the Gröbner base while CHRs are working on the constraint store. When a variable is eliminated we apply some labelling on the roots of the polynomial used for the elimination. As `A is 1` can be seen as the new constraint

`A =:= 1`, the instantiation is not only propagated in the Gröbner base (by ECLiPSe) but also to the related constraints (by the CHRs). Thus the CHRs can eliminate other variables since some constraints (from the constraint store) may become linear. The CHRs can even stop the computation if an inconsistency is detected by the linear equation solver. So when processing a polynomial of the Gröbner base, the variable may be already instantiated by the CHRs and we just need to check that this polynomial is equal to zero (see execution trace in section 3.6.3). This mechanism is iterated till no more variable remains.

Labelling without cooperation
This procedure is nearly the same as the previous one. But the constraint store is emptied so that the value of an eliminated variable is propagated to the Gröbner base only. Furthermore CHRs does not work, since there is no constraint to process in the store. Thus the answer processor module must eliminate the variables and perform the labelling itself (see execution trace in 3.6.3).

Comparison of the two methods
We can now compare the two methods with respect to our goal and the spirit of **CoSAc**. Since the first method uses cooperation between the linear and the non-linear solver, less variables have to be eliminated and more polynomials have to be checked. As checking a polynomial is faster than computing roots, this procedure is more efficient. In the first method, the inequations participate more actively as they are verified sooner (by the CHRs and ECLiPSe). So non valid instantiation of a variable are removed sooner. Furthermore the first method exploits the functionalities (labelling, propagation, ...) of ECLiPSe, of the CHRs and of the CLP scheme and really fits the spirit of **CoSAc** (cooperation, use of functionality of existing tools). In the second method, the solvers do not cooperate during answer processing and so, they do not exchange data. Hence less information is propagated (only substitution) because ECLiPSe runs, but not the CHRs.

The two methods can be retained for our purpose. But the first one is faster as some variable eliminations are performed by the linear solver and not by Maple. Instead of elimination we check that the polynomial is equal to zero (see the execution trace below). Root computation in Maple is triggered only when no other mechanism is possible.

labelling with cooperation

```
eclipse 27:  zmul(X,Y,X,Y,-3,4).

------Non Linear Solver Running------
Call Gb------------------Back from Gb
Finitely many solutions to the system

eliminate

labelling X is -1
check
check
check
check
check

X = -1
Y = -2_1 More?  (;)

labelling X is 1
check
check
check
check
check

X = 1
Y = 2_1 More?  (;)

no (more) solution.
```

labelling without cooperation

```
eclipse 29:  zmul(X,Y,X,Y,-3,4).

------Non Linear Solver Running------
Call Gb------------------Back from Gb
Finitely many solutions to the system

eliminate
labelling X is -1
eliminate
labelling _g408 is 1
eliminate
labelling Y is -2
eliminate
labelling _g624 is 4
eliminate
labelling _g1198 is 2
eliminate
labelling _g1390 is 2

X = -1
Y = -2 More?  (;)
labelling X is 1
eliminate
labelling _g408 is 1
eliminate
labelling Y is 2
eliminate
labelling _g624 is 4
eliminate
labelling _g1198 is 2
eliminate
labelling _g1390 is 2

X = 1
Y = 2 More?  (;)

no (more) solution.
```

4 Integration

This section describes the implementation of **CoSAc**, and the integration of its main components. We first expose the global client/servers architecture of the system before explaining the communication process and its protocol.

4.1 Architecture of the system

CoSAc is based on a distributed architecture (see Figure 1): each component is specialized for solving some subproblems of the input problem. ECLiPSe knows which parts of the problem can be manipulated by which server and plays the role of the client. In the

client/servers architecture, Maple and GB are servers that furnish new features to ECLiPSe which is a client. But Maple is also a server for GB: i.e. GB asks Maple to transform polynomials (expand, ...) before computing Gröbner bases.

Then ECLiPSe controls (via the constraint manager and the communication manager) all the data flows and calls to other components. This control is synchronous since ECLiPSe always waits for the answer after a request.

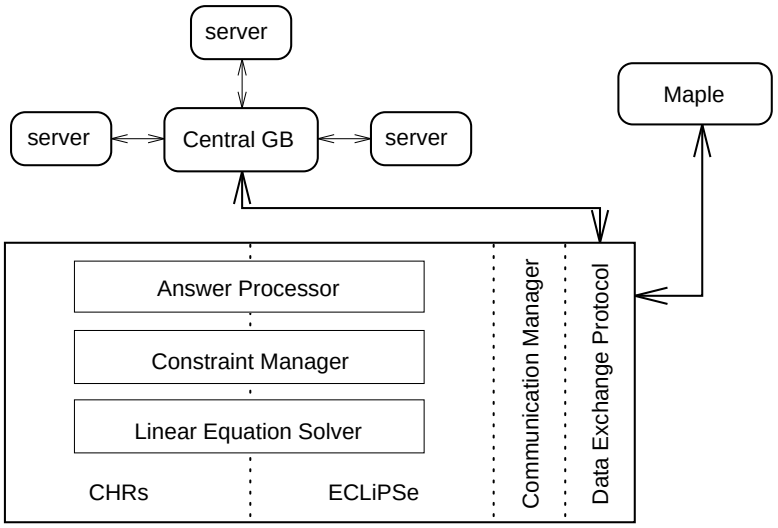

Figure 1: Architecture of CoSAc

4.2 Control of the calls to the solver

The main predicate of this part is `request(f,In,Out)` which asks for a function `f` to be applied on input `In` to obtain result `Out`. When such a predicate is encountered during a resolution, the communication manager searches for a server able to process `f`. Then the input In is converted (using the data exchange protocol) before transmitting the request to the appropriate server. The result from the solver is then converted into `In`.

The communication manager launches on demand the servers: if the server which is able to compute `f` is not running, then the communication manager launches it. For example, when there are only linear equations in the constraint store, the CHRs proceed by themselves, and the system is not overloaded with Maple and GB: they are loaded (if not yet) when some non-linear constraints remain in the store after their selection in the constraint manager.

Furthermore the user can call explicitely for the non-linear constraint solver. There is a special predicate `solver` which, when encountered during a resolution, triggers the following procedure via the constraint manager:

1. collects and converts the constraints
2. if there are some non linear constraints, calls for the servers Maple and GB
3. sends back their results converted in appropriate format

4.3 Communication between processes

ECLiPSe is the only client that synchronizes the communication. It has a central role and the servers are children processes linked by pipes to the platform. At this level the only datatype they can exchange is the string datatype. Since this is a low level communication **CoSAc** needs a data exchange protocol in order to transmit complex structures and to federate the Application Programming Interfaces (API's) of the components.

4.4 Data exchange protocol

Each tool admits its own representation (structure) for an object. So when exchanging data, we need to convert a structure of the sending tool into the corresponding structure of the receiving tool [25]. The communication module of **CoSAc** performs all these conversion tasks. This module implemented in ECLiPSe knows the data structure used by each component.

The most interesting point is the variable conversion. If several variables are transfered together through the pipe (for instance when handling a multivariate polynomial), the variables are not recognized when received back. This is due to the term transformation of Prolog which creates new variables. Thus two cases occur:

- user variables (or query variables): the created variable has the same name as the user variable, but not the same internal name. So they are not substituted, but it is easy to force it using their user name.
- internal variables (created during resolution by ECLiPSe or the CHRs): they only have an internal name, and no user name. So the problem is to create a name that can be identified when the variable is back.

This mechanism has been implemented using the same technique for both cases. The variables to be sent are encoded using their name (or a given name for non query variables) and their internal name. The specificity of this coding is that the received variables give the same result that the sent variables after the encoding. So they can be recognized and the information they bring back is associated to the right variable. We illustrate this process with a multivariate polynomial:

-$P(x_1, \ldots, x_n)$ is a polynomial of ECLiPSe.
-The data exchange protocol converts $P(x_1, \ldots, x_n)$
into $P(x'_1, \ldots, x'_n)$ using the coding $\mathcal{C}$
-The communication manager sends $P(x'_1, \ldots, x'_n)$ to a
server S.

-$P(x'_1, \ldots, x'_n)$ is transformed by S into $P'(x'_1, \ldots, x'_n)$
-S sends back $P'(x'_1, \ldots, x'_n)$
-After term transformation, ECLiPSe receives
$P'(x''_1, \ldots, x''_n)$
-The data exchange protocol uses $\mathcal{C}$ to convert
$P'(x''_1, \ldots, x''_n)$ into $P'(x_1, \ldots, x_n)$.

Moreover the structure conversion preserves the variable ordering:

$$\text{if } \; x_n > x_{n-1} > \ldots > x_1 \; \text{ then } \; x'_n > x'_{n-1} > \ldots > x'_1 \; .$$

5 Cooperation in CoSAc

This section deals with the cooperation between the different components of **CoSAc** and demonstrates that **CoSAc** respects the specifications described in section 2.

5.1 Route of information related to a constraint

Cooperation between components can be summarized to "process and exchange data". This section exposes the data exchange. Figure 2 illustrates how the information hold in a constraint circulates in **CoSAc** and is used in the different solvers and modules.

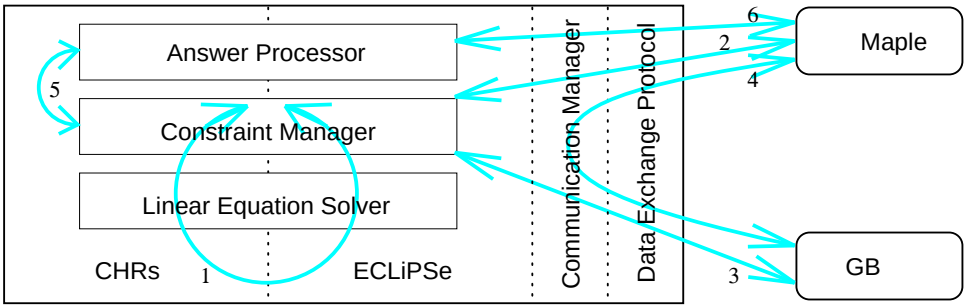

Figure 2: Route of the information

During the first phase (1), the constraint is manipulated by the constraint manager, and the linear equation solver. The transformation rules of the CHRs are applied, and the information related to the constraint is propagated to the other constraints. When the linear solver cannot progress further, the constraint manager sorts the constraints using Maple (2) to determine which constraints are non linear polynomial equations, and to simplify them (to expand them for example). Then the polynomial constraints are sent to GB (3). GB requires a special format for the polynomials. It requests Maple (4) to transform them. After Gröbner bases computation, GB transmits the result to the constraint manager. Then the

constraint store is given (5) to the answer processor module. This last one requests Maple (6) for the computation of roots of univariate polynomial. Then phase (1) propagates the roots and tries to solve the system of constraints (when using labelling without cooperation (section 3.6.3) the phase (1) is skipped). If the linear equation solver cannot do it by itself, the answer processor eliminates a new variable (5 and 6), and so on till all variables are instantiated.
Remark : communications with the servers are handled by the communication manager and the data exchange protocol module.

5.2 Advantages of Cooperation

We illustrate this section with the example of Section 1. It shows that cooperation between solvers permits to tackle some problems:

- that cannot be solved using only a linear solver.
- that cannot be solved using only a non-linear solver.
- that do not need a general solver to be solved, because occasionally only the linear part is mandatory.

Each query shows a particular point of **CoSAc**.

- **Query 1:** the following query instantiates some variables of the problem. Hence GB is not ran and the CHRs solve the program by themselves:

  ```
  zmul(1,2,3,4,R3,I3).
  ```

  The answer to the query is:

  ```
  R3 = -5_1
  I3 = 10_1
  yes.
  ```

- **Query 2:** with this query, some non-linear equations remain in the constraint store after the CHRs first processed. So GB transforms the polynomials into their Gröbner base, and the answer module extracts the solutions using propagation and the CHRs: the solvers cooperate in order to solve the problem.

  ```
  zmul(X,Y,X,Y,-3,4).
  ```

  Upon this query **CoSAc** answers:

```
------Non Linear Solver Running------
Call Gb------------------Back from Gb
Finitely many solutions to the system

       X = -1
       Y = -2_1     More? (;)

       X = 1
       Y = 2_1     More? (;)

       no (more) solution.
```

- **Query 3:** this query is the same as the previous one but more constrained with an inequation. Hence, during the labelling in the answer module, the non valid values are removed by the propagation rules and the CHRs.

  ```
  zmul(X,Y,X,Y,-3,4), X>0.
  ```

  Upon this query **CoSAc** produces the following answer:

  ```
  ------Non Linear Solver Running------
  Call Gb------------------Back from Gb
  Finitely many solutions to the system

         X = 1
         Y = 2_1     More? (;)

         no (more) solution.
  ```

  The following trace of execution shows the differences when using labelling with/without cooperation (see section 3.6.3) to solve problems containing inequations.

  We can see that with cooperation, the inequation `X>0` is checked as soon as the variable is instantiated. Moreover the CHRs eliminate the variables themselves, so the answer processor just checks the polynomials of the Gröbner base.

  Without cooperation, the answer processor eliminates the variables by itself. The inequation is checked only when all the variables are instantiated. Then the next value for `X` is tried and the process is iterated.

Labelling without cooperation

```
eclipse 25:  zmul(X,Y,X,Y,-3,4), X>0.

------Non Linear Solver Running------
Call Gb------------------Back from Gb
Finitely many solutions to the system

eliminate
labelling X is -1
eliminate
labelling _g420 is 1
eliminate
labelling Y is -2
eliminate
labelling _g636 is 4
eliminate
labelling _g1210 is 2
eliminate
labelling _g1402 is 2
eliminate
labelling _g2116 is -1
                <- X > 0 is checked
                <- so backtrack
labelling X is 1 <- try next value for X
eliminate
labelling _g420 is 1
eliminate
labelling Y is 2
eliminate
labelling _g636 is 4
eliminate
labelling _g1210 is 2
eliminate
labelling _g1402 is 2
eliminate
labelling _g2116 is 1

Y = 2
X = 1 More?  (;)

no (more) solution.
```

Labelling with cooperation

```
eclipse 26:  zmul(X,Y,X,Y,-3,4),X>0.

------Non Linear Solver Running------
Call Gb------------------Back from Gb
Finitely many solutions to the system

eliminate
labelling X is -1
                <- X>0 is already
                <- checked
labelling X is 1 <- try next value
for X
check
check
check
check
check
check

Y = 2_1
X = 1 More?  (;)

no (more) solution.
```

6 Examples

In this section we present several examples which illustrate the applications that can be treated with **CoSAc** (see section 1).

6.1 A ballistic problem

This problem from [19] consists in finding the falling point of an object.

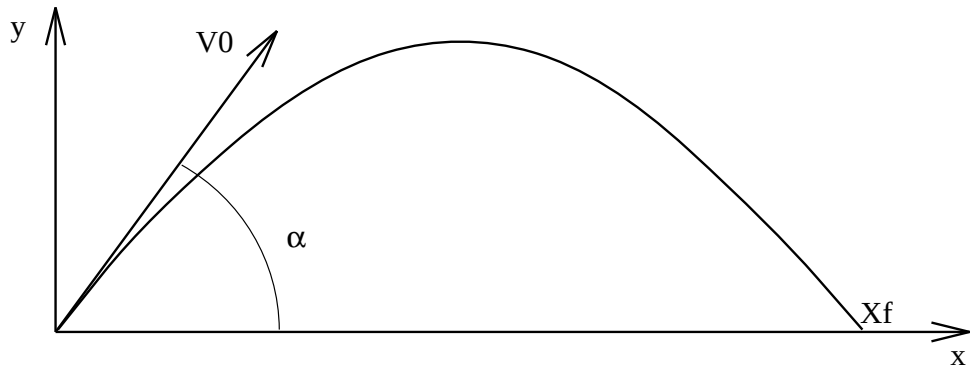

Figure 3: Ballistic problem

The object is launched with an initial speed (V_x, V_y) and an incidence α with the ground (see Figure 3). Thus the parametric trajectory of the object (submitted to the gravitation and having an initial speed equal to 0) is:
$x = V_x t$ and $y = 1/29.81t^2 + V_y t$.

Two more constraints are added to state the relation between the initial speed and the incidence of the object: $V_y = V_x tan(\alpha)$ and $V_x^2 + V_y^2 = V0^2$

We denote X_f, Y_f and T_f the falling point and the falling time. As noticed in section 1 we "abstract" $tan(\alpha)$ with a new variable `Tan`. We obtain the following predicate:

```
ballistic(Vy, Vx, Tan, V0, Xf, Tf, Yf):-
        Vy>0,
        Vx>0,
        Xf>0,
        V0>0,
        Vy =:= Vx*Tan,
        Vx^2 + Vy^2 =:= V0,
        Xf =:= Vx*Tf,
        Yf =:= - 981/200*Tf^2 + Vy*Tf.
```

To determine the falling point on the earth (i.e. $X_f = 0$) of an object knowing its initial speed ($V0 = 500$) and the initial incidence ($tan(\alpha) = 1$), we ask the following query:

```
ballistic(Vy, Vx, 1, 500, Xf, Tf, 0).
```

and the answer is

```
Vy = 6240166921_394662809
Vx = 6240166921_394662809
Xf = 50000_981
Tf = 3606307121_1118746570
```

The whole computation requires about 1 second. The comparison with [19] is not easy. By only giving an interval for the initial speed and the incidence they compute a spot covering the possible falling points: the computation takes about 5 seconds.

6.2 Rectangle in a right angled corridor

This problem is a restriction of the well known **"piano mover's problem"** [18]. The problem [21] is to move a rectangle `R` of length `L` and width `LL` through a right angled corridor (`Cor_Hor` is the width of the horizontal corridor, `Cor_Ver` is the width of the vertical corridor). R can rotates around each of its points and translates in the direction of its length. R has to move from an initial position I in the vertical corridor to a position F in the vertical corridor.

We have chosen the following movement (Figure 4: corridor): R slices in the angle with one point of contact with the x-axis (B) and a second point of contact with the y-axis (A). This path is always a solution, assuming that the problem can be solved.

In this way we obtain the trajectory of the rectangle R, and we have to determine under which conditions this path is valid. For this purpose, we use the upper envelop E of the moving rectangle. E is a parametric equation:

$$
\begin{aligned}
x(t) &= lt^3 + L\sqrt{1-t^2} \\
y(t) &= Lt + l(1-t^2)^{(3/2)} \\
t_1 &\leq t \leq t_2
\end{aligned}
$$

with

$$
\begin{aligned}
t_1 &= (1/2 - (9l^2 - 4L^2)^{1/2}/(6l))^{1/2} \\
t_2 &= (1/2 + (9l^2 - 4L^2)^{1/2}/(6l))^{1/2}
\end{aligned}
$$

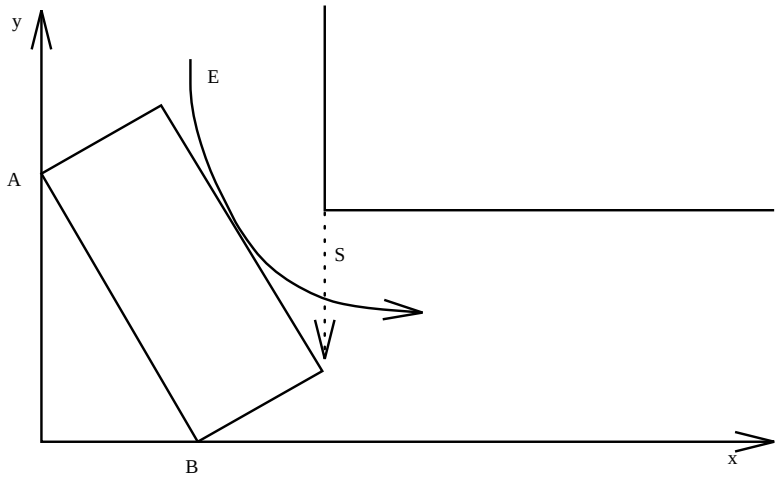

Figure 4: Corridor

We want to determine the minimal width `Cor_Hor` of the horizontal corridor when knowing the dimensions `L` and `LL` of the rectangle `R` and the width `Cor_Ver` of the vertical corridor.

The corresponding program is equivalent to the computation of the intersection of two moving points: one moving on `E`, the other one moving vertically on `S`. The two parametric curves (`E` and `S`) are defined with the same variables `X` and `Cor_Hor`. So these two variables represent the coordinates of the intersection of the two curves. `W` is added to represent the square root of $1 - t^2$.

```
corridor(L,LL,Cor_Vert,Cor_Hor):-
    square_root(W, (1-T^2)),
    param(L*T^3,LL*W,LL*T ,L*W^3,X,Cor_Hor),
    param(0, Cor_Vert, R*T, 0, X, Cor_Hor),
    T>(1/2-(9*L^2-4*LL^2)^(1/2)/6*L)^(1/2),
    T<(1/2+(9*L^2-4*LL^2)^(1/2)/6*L)^(1/2).
```

Remark 1: the parametric curves are defined using the predicate `param` of our geometric library. This predicate is a meta constraint which sets two constraints (i.e. the projection of the motion on the x-axis and the y-axis knowing the speed and the initial point of the object). We could also use `param` to treat the previous example. The definition of `param` is:

```
param(Fx,Dx,Fy,Dy,X,Y):-
        Y - Fy - Dy =:= 0,
        X - Fx - Dx =:= 0.
```

Remark 2: `square_root` is also a predicate of the geometric library and is define as follows:

```
square_root(X,Y):-
        X^2 - Y =:= 0,
        X > 0.
```

Minimal width: the minimal width (see Figure 4) of the horizontal corridor knowing that the width of the vertical corridor is 3/2, the length of the rectangle is 2 and its width is 1 can be computed with the query:

```
corridor(2,1,3_2, Min_Cor).
```

and the solution is

```
Min_Cor = 3180632219_2385445214
```

The flexibility of the system allows processing the same treatment with different parameters, or determining several relations between these parameters. Furthermore some other problems (as determining if the object can go through when knowing all the parameters) can be solved easily by the addition of new predicates using the same primitives (defined in the geometric library).

Remark 3: we are developing a library for constraint transformation. So square roots and trigonometric relations can be transformed automatically into polynomial constraints. Hence the user will not have to add extra variables.

7 Conclusion and future works

In this paper we have presented our work on introducing cooperation between solvers in a CLP system over non-linear constraints. This work was motivated by applications and the need for cooperation to increase the efficiency of such a system. We have shown the great benefit of cooperation: it does not only increase the capabilities of **CoSAc**, it also enables to re-use powerful tools as ECLiPSe, CHRs, GB and Maple.

Moreover the client/server architecture allows one just as well to extend **CoSAc** with new features and new solvers or to replace them by some more powerful future ones. Thus we plan to add some specialized solvers and some new components for the following reasons. First, solving systems of polynomial constraints is a hard work. So specialized solvers (e.g. for quadratic equations [26]) can cooperate with the other solvers and lighten their tasks. Furthermore we want to extend the expressiveness of **CoSAc** with more general constraints (e.g. transcendental functions [10]). Finally, to make life easier for the user, we will introduce a library of polynomial transformations. The aim of this component is to automatically convert into polynomial equations constraints that are not presented in this form.

Sequential synchronous communication works well as long as there are only one "fast" solver (e.g. Simplex) and one "slow" solver (e.g. GB). However, asynchronous communication protocols can offer more flexibility if we want to integrate another "slow" solver (e.g. another polynomial solver). Hence we will extend **CoSAc** with an asynchronous communication protocol. So it would be possible for several "slow" solvers to work at the same time, and possibly on the same set of constraints, but with different algorithms.

References

[1] A. Aiba, K. Sakai, Y. Sato, D. J. Hawley, and R. Hasegawa. Constraint Logic Programming Language CAL. In *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS'88), Tokyo (Japan)*, pages 263–276. ICOT Research Center, December 1988.

[2] F. Benhamou, D. McAllester, and P. Van Hentenryck. CLP(Intervals) Revisited. In *Proc. of ILPS'94: International Logic Programming Symposium*, pages 124–138, 1994.

[3] H. Beringer and B. De Backer. Combinatorial problem solving in constraint logic programming with cooperative solvers. In *Logic programming: formal methods and practical applications.* Elsevier Science Publisher B.V., 1995.

[4] B. Buchberger. Gröbner Bases: an Algorithmic Method in Polynomial Ideal Theory. In N. K. Bose Ed., editor, *Multidimensional Systems theory*, pages 184–232. D. Reidel Publishing Company, Dordrecht - Boston - Lancaster, 1985.

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

[6] G. E. Collins. Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. In *Proceedings of the Second GI Conference on Automata Theory and Formal Languages*, pages 515–532. Springer Lecture Notes in Computer Science 33, 1975.

[7] J-C. Faugere. *Résolution des systèmes d'équations algébriques*. PhD thesis, Université Paris 6, 1994.

[8] T. Frühwirth. Constraint handling rules. In A. Podelski, editor, *Constraint Programming: Basics and Trends*, volume 910 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[9] T. Frühwirth, A. Herold, V. Kuechenhoff, T. Le Provost, P. Lim, E. Monfroy, and M. Wallace. Constraint logic programming - an informal introduction. In G. Comyn, M. Ratcliffe, and N. Fuchs, editors, *Logic programming in action : proceedings LPSS'92, Second international logic programming Summer school, Zurich, Switzerland*, volume 636 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1993.

[10] Xiaoshan Gao. Transcendental functions and mechanical theorem proving in elementary geometries. *Journal of Automated Reasoning*, 6:403–417, 1990.

[11] Keith O. Geddes, Gaston H. Gonnet, and Benton L. Leong. *Maple V : Language reference manual*. Springer Verlag, New York, Berlin, Paris, 1991.

[12] A. Giovani, T. Mora, G. Niesi, L. Robbiano, and C. Traverso. One sugar cube, please, or selection strategies in the buchberger algorithm. In S. M. Watt, editor, *Proceedings of ISSAC'91: the 1991 International Symposium on Symbolic and Algebraic Computation*. ACM Press, 1991.

[13] Hoon Hong. Non-linear Real Constraints in Constraint Logic Programming. In Hélène Kirchner and G. Levi, editors, *Proceedings 3rd International Conference on Algebraic and Logic Programming, Volterra (Italy)*, volume 632 of *Lecture Notes in Computer Science*, pages 201–212. Springer-Verlag, September 1992.

[14] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages, Munich, Germany*, pages 111–119. ACM Press, January 1987.

[15] T. Khedro and M.R. Genesereth. Modeling multiagent cooperation as distributed constraint satisfaction problem solving. In A. Cohn, editor, *Proc. of ECAI'94: 11th European Conference on Artificial Intelligence*, pages 249–253, 1994.

[16] H. Kirchner and C. Ringeissen. Combining symbolic constraint solvers on algebraic domains. *Journal of Symbolic Computation*, 18(2):113–155, 1994.

[17] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston/ Dordrecht/ London, 1991.

[18] S. R. Maddila and C. K. Yap. Moving a Polygon arround the Corner in a Corridor. In *Proceedings of the ACM Symposium on Computational Geometry*, pages 187–182, Yorktown Heights, NY, 1986.

[19] P. Marti and M. Rueher. A cooperative scheme for solving constraints over the reals. In Hoon Hong, editor, *Proc. of PASCO 94: First Parallel Symbolic Computation Symposium*, volume 5, pages 284–293. World Scientific, 1994.

[20] M. Meier and J. & al Schimpf. ECLiPSe User Manual. Technical report ECRC-93-6, ECRC, Munich (Germany), 1993.

[21] E. Monfroy. Gröbner Bases : Strategies and Applications. In J. A. Campbell J. Calmet, editor, *Proceedings International Conference Artificial Intelligence and Symbolic Mathematical Computing*, Karlsruhe (Germany), 1992. Springer Verlag. Lecture Notes in Computer Science, 737.

[22] E. Monfroy. Non Linear Constraints: a Language and a Solver. Report 93, ECRC, Munich (Germany), FEV 1993.

[23] C. G. Nelson and D. C. Oppen. Simplifications by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2), 1979.

[24] OMG. The Common Object Request Broker : Architecture and Specification. Technical Report OMG Document Number 91.12.1, Revision 1.1, OMG, December 1991.

[25] O. Perrin and N. Boudjlida. A Formal Framework and a Procedural Approach for Data Interchange. In IEEE Computer Society Press, editor, *Proceedings of the 3rd International Conference on System Integration*, São Paulo, Brazil, August 1994.

[26] G. Pesant and M. Boyer. QUAD-CLP(R): Adding the power of quadratic constraints. In Alan Borning, editor, *Proc. of PPCP'94: Second Workshop on Principles and Practice of Constraint Programming*, Seattle, May 1994.

[27] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[28] M. Rueher. An architecture for cooperating constraint solvers on reals. In A. Podelski, editor, *Constraint Programming: Basics and Trends*, volume 910 of *Lecture Notes in Computer Science*, pages 231–250. Springer-Verlag, 1995.

[29] P. Van Hentenryck. *Constraint satisfaction in logic programming*. MIT Press, Cambridge, London, 1989.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399