Efficient Handling of Disequations in CLP over Linear Rational Arithmetic

Jean-Louis Imbert[1]
Pascal Van Hentenryck[2]

Technical Report No. CS-91-23
March 1991

[1]G.I.A. Parc Scientifique et Technologique de Luminy, 163, Avenue de Luminy, F-13288 Marseille cedex 9 France
[2]Department of Computer Science, Box 1910, Brown University, Providence, RI 02912

Efficient Handling of Disequations in CLP over linear Rational Arithmetic

Jean-Louis Imbert
G.I.A Parc Scientifique et Technologique de Luminy
163, Avenue de Luminy
F-13288 Marseille cedex 9 (France)

Pascal Van Hentenryck
Brown University, Box 1910,
Providence, RI 02912
Email: pvh@cs.brown.edu

Abstract

This paper considers the (efficient) handling of disequations for a CLP language over linear rational (resp. real) arithmetic (based on the simplex algorithm). It discusses both solved forms for the constraints and algorithms to transform systems of constraints into solved form.

The starting point of the paper is a syntactic solved form for equations and disequations which precludes the presence of fixed variables (or hidden constants). Several subclasses of constraints, that occurs frequently and can be handled more efficiently, are then identified. This leads to the definition of an improved solved form that reduces the computational burden imposed on the constraint-solving algorithm.

An incremental algorithm to obtain the improved solved form is then proposed. Its main feature is to exploit the possibility of lazy dereferencing offered by the improved solved form while still detecting all fixed variables. Two important refinments of the algorithm are then described, formally justified, and included in the algorithm. The first improvement avoids checking disequations altogether for an important class of constraints while the second improvement allows for a more active use of disequations, achieving an early detection of failures and precluding unnecessary calls to the simplex algorithm.

1 Introduction

In this paper, we consider a CLP language over linear rational arithmetic, say CLP($\mathcal{Q}$). CLP($\mathcal{Q}$) is a sublanguage of several CLP languages including Prolog III [2], CLP($\Re$)[1] [5] and CHIP [4]. The domain of computation are rational numbers and the constraints are of the form

$$t_1 \ \delta \ t_2$$

with $\delta \in \{>, \geq, =, \neq, \leq, <\}$ and t_1, t_2 are linear terms. The presence of $\{>, \neq, <\}$ as possible constraints entails that the problem is a generalization of phase I of linear programming.

[1] CLP($\Re$) works with linear constraints over real numbers but the results of the paper hold for real numbers as well.

Most CLP languages endowed with this constraint system are based on (generalizations of) the simplex algorithm [3]. The presence of disequations requires however a modification to the simplex algorithm. Informally, it is necessary, in order to decide disequations, to detect fixed variables (variables constrained to take only one value), a requirement not fulfilled by the simplex algorithm (see [7, 9, 10] for a discussion of fixed variables). Detecting fixed variables is also useful for other purposes such as implicit ask constraints (e.g. to detect that a (previously) non-linear constraint is now linear). However little attention has been devoted to the efficient handling of disequations and, to the best of our knowledge, no research has been concerned with their efficient handling.

The present paper is an attempt to fill this gap by presenting a rather complete account to the handling of disequations in CLP($\mathcal{Q}$). The starting point of the paper is a syntactic solved form for the constraints, precluding the presence of fixed variables. The limitations of the solved form from a computational standpoint are stressed and an improved solved form is proposed. The improved solved form reduces the computational burden imposed on the constraint-solving algorithm by identifying subclasses of constraints that arise frequently in practice and can be handled more efficiently. It also preserves the "no fixed variable" property. A constraint-solving algorithm for the improved solved form is then described in detail. Two significant improvements are identified, formally justified, and integrated inside the algorithm. This first improvement identifies a class of constraints which, when added, do not require checking the disequations and updating certain classes of equations. The second improvement shows how disequations can be checked (by "syntactic matching") before the call to the simplex algorithm, achieving an early detection of failures.

The remaining of the paper is organized as follows. Section 2 presents the syntactic solved form. Section 3 introduces the improved solved form. Section 4 describes the incremental constraint-solving algorithm while Section 5 describes its improvements. Section 6 describes the conclusion of this research and directions for further work.

2 A Syntactic Solved Form

In CLP($\mathcal{Q}$) the constraint solver transforms all constraints into equations and disequations by introducing slack variables (s-variables for short). A slack variable is a variable constrained to take only nonnegative values. Hence an inequality $t_1 \geq t_2$ (resp. $t_1 \leq t_2$) is transformed into an equation $t_1 - x^+ = t_2$ (resp. $t_1 + x^+ = t_2$) where x^+ is a slack variable. Strict equalities are transformed into the conjunction of an equation and a disequation. Hence the constraint problem reduces to the satisfiability of a system of equations and disequations over arbitrary variables (a-variables for short) and s-variables. We present a solved form for these constraints in three steps by first considering only s-variables, then only a-variables, and finally by combining them together.

In the following, we use the letters x, y, z, possibly subscripted, for variables. s-variables, except when clear from the context, are superscripted with a $+$. Rational numbers are denoted by the letters a, b, possibly subscripted. Terms are denoted by the letters d, t, possibly subscripted. Constraints are usually denoted by the letters c and e.

2.1 Equations and Disequations over s-variables

The solved form for this problem was first introduced in [10] where it was called SF2.[2] The proofs of all results presented in this subsection can be found in the abovementioned paper. The syntactic solved form assumes a lexicographic ordering on the variables. Hence we will only consider in the following (lexicographically ordered) terms t of the form

$$a_0 + \sum_{i=1}^{n} a_i x_i^+$$

where $x_i^+ \ll x_{i+1}^+$ $(1 \leq i < n)$. We first consider systems of equations over s-variables.

Definition 1 An equation over s-variables is in solved form iff it is of the form

1. $y^+ = a_0$ $(a_0 \geq 0)$ or
2. $y^+ = a_0 + \sum_{i=1}^{n} a_i x_i^+$ $(a_0 \geq 0)$ where either the constant a_0 is positive or the first non-zero coefficient (say a_j) is positive and $x_j^+ \ll y^+$.

The left-hand side variable is called a *basic* variable while the variables appearing on the right-hand side are called *non-basic* variables.

Definition 2 A system of equations over s-variables $\mathcal{E}$ is in solved form iff

1. each equation in $\mathcal{E}$ is in solved form;
2. each basic variable occurs only once in $\mathcal{E}$;

There are two main results concerning the above solved form.

Lemma 3 A system $\mathcal{E}$ of equations over s-variables in solved form does not contain any fixed variables other than those made explicit by a constraint of the form $y^+ = a_0$.

Lemma 4 A system $\mathcal{E}$ of equations over s-variables is satisfiable iff it can be mapped into solved form.

The solved form can be extended in a straightforward way to disequations.

Definition 5 A disequation over s-variables is in solved form iff it is of the form

$0 \neq a_0 + \sum_{i=1}^{n} a_i x_i^+$ where at least one a_i $(0 \leq i \leq n)$ is non-zero.

Definition 6 A system $(\mathcal{E}, \mathcal{D})$ of equations and disequations over s-variables is in solved form iff

1. the system of equations is in solved form;
2. all disequations are in solved form and do not contain any of the basic variables of $\mathcal{E}$.

[2] Systems like Prolog III and CLP($\Re$) do not work with a purely syntactic solved form but rather assume semantics properties (i.e. the absence of fixed variables). Which of the two approaches is more efficient is still an open issue although the syntactic solved form, besides its simplicity, has some advantages to detect fixed variables and redundant constraints [11].

We now present the main result for the above solved form.

Theorem 7 A system of linear equations and disequations over s-variables is satisfiable iff it can be mapped into solved form.

Another important result is that the simplex algorithm, provided that it uses a lexicographic pivoting rule, is guaranteed to preserve the solved form through pivoting and hence can be used to provide the kernel of the decision procedure (see [10] for detail).

Theorem 8 The simplex algorithm using a lexicographic pivoting rule preserves the solved form through pivoting.

2.2 Equations and Disequations over a-variables

We now turn to the solved form for equations and disequations over a-variables.

Definition 9 An equation over a-variables is in solved form iff it is of the form

$$y = a_0 + \sum_{i=1}^{n} a_i x_i.$$

Once again, the left-hand side variables is called a basic variable while the right-hand side variables are non-basic variables.

Note that no lexicographic ordering or nonnegativity constraint is imposed on the solved form resulting in a simpler solved form.

Definition 10 A system of equations $\mathcal{E}$ is in solved form iff

1. each equation in $\mathcal{E}$ is in solved form;
2. each basic variable appears only once in $\mathcal{E}$;

Definition 11 A disequation over a-variables is in solved form iff it is of the form

$0 \neq a_0 + \sum_{i=1}^{n} a_i x_i$ where at least one a_i $(0 \leq i \leq n)$ is non-zero.

Definition 12 A system $(\mathcal{E}, \mathcal{D})$ of equations and disequations over a-variables is in solved form iff

1. the system of equations is in solved form;
2. all disequations are in solved form and do not contain any of the basic variables of $\mathcal{E}$.

Gaussian elimination can be used to obtain a solved form for these constraints. All variables appearing in the solved form (except those which are assigned an explicit constant) are guaranteed not to be fixed variables. The proofs of these results are immediate.

2.3 Combining both Results

To complete the description of the solved form, it remains to describe how the above two solved forms can be combined. The basic idea is that the solved form for a-variables can be generalized to accommodate s-variables in the right-hand side.

Definition 13 An equation over (at least one) a-variables and s-variables is in solved form iff it is of the form

$$y = a_0 + \sum_{i=1}^{n} a_i x_i + \sum_{j=1}^{m} b_j x_j^+$$

where y is an a-variable.

Definition 14 A disequation over a-variables and s-variables is in solved form iff it is of the form

$$0 \neq a_0 + \sum_{i=1}^{n} a_i x_i + \sum_{j=1}^{m} b_j x_j^+$$

where at least one a_i $(1 \leq i \leq n)$ is non-zero.

The definitions for systems of equations and disequations are similar to those of the previous solved form. We can now define our basic solved form.

Definition 15 Let $\mathcal{E}_s$,$\mathcal{D}_s$ be systems of equations and disequations over s-variables. Let $\mathcal{E}_a$,$\mathcal{D}_a$ be systems of equations and disequations over a-variables and s-variables. A system is in solved form iff

1. $< \mathcal{E}_s, \mathcal{D}_s >$ is in solved form;
2. $< \mathcal{E}_a, \mathcal{D}_a >$ is in solved form;
3. None of the basic variables of $\mathcal{E}_s$ appear in $< \mathcal{E}_a, \mathcal{D}_a >$.

Theorem 16 A set of constraints over a-variables and s-variables is satisfiable iff it can be mapped into solved form. Moreover any system in solved form do not contain any implicit fixed variable.

Proof Obviously $< \mathcal{E}_s, \mathcal{D}_s, \mathcal{E}_a, \mathcal{D}_a >$ is solvable. Moreover it is possible to exhibit an algorithm to transform any system of constraints into the above solved form by combining the algorithms available for the above two solved forms.[3] Gaussian elimination can be used until only equations over s-variables are left. The simplex algorithm, using a lexicographic pivoting rule, can then be applied. Finally all equations and disequations should be processed to remove the basic variables. Similarly, it follows from the results above the solved form for s-variables and a-variables that $< \mathcal{E}_s, \mathcal{D}_s, \mathcal{E}_a, \mathcal{D}_a >$ has no fixed variable (except those made explicit). Clearly this is the case for $\mathcal{E}_s$. Moreover all non-constant right-hand side of an equation in $\mathcal{E}_a$ contain either an a-variable (which means that the basic variable is not fixed) or only s-variables (in which case the definition of the solved form for s-variables guarantees that the right-hand side can take infinitely many values). □

We conclude this section by presenting a number of definitions that are necessary for the rest of this paper. They are intended to capture the concept of dereferencing in this constraint system.

[3] A variation of such algorithm will be presented in Section 4.

Definition 17 Let $\mathcal{E}$ be a set of equations. An equation $y = t$ (resp. a disequation $0 \neq t$) is dereferenced wrt $\mathcal{E}$, denoted $IsDeref(y = t, \mathcal{E})$ (resp. $IsDeref(y \neq t, \mathcal{E})$) iff t does not contain any of the basic variables of $\mathcal{E}$. The definition can be extended to sets of constraints. If $\mathcal{C}$ is a set of constraints, $IsDeref(\mathcal{C}, \mathcal{E})$ holds if $IsDeref(c, \mathcal{E})$ holds for each $c \in \mathcal{C}$.

We also need a function that returns the dereferenced version of a constraint.

Definition 18 Let $\mathcal{E}$ be a set of constraints in solved form. The function $Deref(y = t, \mathcal{E})$ (resp. $Deref(0 \neq t, \mathcal{E})$) returns the constraint $y = t'$ (resp. $0 \neq t'$) where each basic variable of $\mathcal{E}$ has been replaced by its right-hand side in the solved form. The definition can be extended to sets of constraints. If $\mathcal{C} = \{c_1, \ldots, c_n\}$ is a set of constraints, $Deref(\mathcal{C}, \mathcal{E})$ is the set $\mathcal{C}' = \{c'_1, \ldots, c'_n\}$ such that $c'_i = Deref(c_i, \mathcal{E})$.

3 A (Computationally) Improved Solved Form

The above solved form is fully adequate to decide the satisfiability of a system of constraints. However it imposes a strong requirement on the constraint-solving algorithm. In this section, we show how the solved form can be specialized further in order to relax the requirements on the algorithm. The basic idea is to divide the sets $\mathcal{E}_a$ and $\mathcal{D}_a$ depending on the presence of a-variables in the right-hand side of a constraint and/or the number of s-variables.

Consider first equations. Remember that one of the main purposes of the solved form is to recognize fixed variables[4]. The first subclass of equations that is worth exploiting is the one where an equation in solved form contains at least one a-variable in its right-hand side. In that case, it is guaranteed that the left-hand side variable is not fixed. Hence the s-variables possibly appearing in the right-hand side do not need to be dereferenced wrt $\mathcal{E}_s$, sparing computation. This subclass is important since s-variables are always introduced by inequalities containing a-variables producing a constraint of the above type.

A second syntactic form of constraints that is worth exploiting is the case where the right-hand side of a constraint contains only one s-variable (and no a-variable). By definition of the solved form, this variable is not fixed (implying that the a-variable is not fixed) and hence the s-variable does not need to be dereferenced. It is only necessary to reconsider this type of constraints when the s-variable becomes fixed. Once again the improvement corresponds to a frequent case in the algorithm. Constraints of the form $x \geq a$ and $x \leq a$ are frequent and leads directly to the special case.

Finally the last syntactic form worth exploiting is the case where the right-hand side contains only one a-variable (and no s-variable). Provided that this variable is not fixed, there is no need to dereference it.

Exactly symmetrical classes exist for disequations (except that here the left member is always zero). The first case correspond to a disequation

$$0 \neq a_0 + a_1 x + t$$

where $a_1 \neq 0$ and x is an a-variable. This constraint is trivially satisfied as x can take infinitely many values (whatever the values assigned to the variables in t). Hence if t contains s-variables, they do not need to be dereferenced wrt $\mathcal{E}_s$.

[4]This is suitable to decide the satisfiability of disequations and necessary for other purposes such as *ask* constraints.

Class Name	Syntactic Form	Comments
$\mathcal{E}_F$	$y = a_0$	Only a constant
$\mathcal{E}_{A1}$	$y = a_0 + a_1 x$	One a-variable
$\mathcal{E}_{A2}$	$y = a_0 + a_1 x_k + a_2 x_l + t_1$	At least one a-variable
	or $y = a_0 + a_1 x + a_2 x^+ + t_2$	and at least two variables
$\mathcal{E}_{C1}$	$y = a_0 + a_1 x^+$	Exactly one s-variable
$\mathcal{E}_{C2}$	$y = a_0 + a_1 x_k^+ + a_2 x_l^+ + t^+$	At least two s-variables

Table 1: Syntactic Classification of Constraints

Class Name	Invariant
$\mathcal{E}_F$	True
$\mathcal{E}_{A1}$	$IsDeref(\mathcal{E}_{A1}, \mathcal{E}_F \cup \mathcal{E}_{A1})$
$\mathcal{E}_{A2}$	$IsDeref(\mathcal{E}_{A2}, \mathcal{E})$
$\mathcal{E}_{C1}$	No fixed s-variable wrt $\mathcal{E}_s$
$\mathcal{E}_{C2}$	$IsDeref(\mathcal{E}_{C2}, \mathcal{E}_s)$
Global	All basic variables appear only once

Table 2: Invariant for the Equation Classes

The second class contains constraints of the form

$$0 \neq a_0 + a_1 x^+$$

where $a_1 \neq 0$ and corresponds to a frequent case. Recall that a strict inequality $t_1 > t_2$ (resp. $t_1 < t_2$) is transformed into a conjunction $t_1 - x^+ = t_2 \ \& \ x^+ \neq 0$ (resp. $t_1 + x^+ = t_2 \ \& \ x^+ \neq 0$), showing that each strict inequality leads to the special case we just isolated[5]. The variable x^+ does not need to be dereferenced wrt $\mathcal{E}_s$ and needs only to be checked when x^+ becomes fixed.

The last case corresponds to the case where only one a-variable is present. We only need to reconsider it once the variable gets fixed.

The new syntactic classification of equations is depicted in Table 1. In the tables, we denote by $\mathcal{E}$ the union of all equation classes $\mathcal{E}_F, \mathcal{E}_{A1}\mathcal{E}_{A2}, \mathcal{E}_{C1}, \mathcal{E}_{C2}$, we assume that a_1 and a_2 are different from zero, t_1 (resp. t_2) represents a term without constant and without variables x_k, x_l (resp. x, x^+) and t^+ is is a term containing only s-variables and not containing any constant and s-variables x_k^+, x_l^+.

It is also necessary to impose an invariant with each class. This defines precisely the improved solved form and will help in defining the algorithm (it would be in fact rather difficult to design a correct algorithm without them). The invariants are depicted in Table 2.

Theorem 19 A system of equations $< \mathcal{E}_s, < \mathcal{E}_F, \mathcal{E}_{A1}, \mathcal{E}_{A2}, \mathcal{E}_{C1}, \mathcal{E}_{C2} >>$ in improved solved form is satisfiable and all fixed variables are made explicit.

Proof First recall that a system of equations contains a loop if, in the process of dereferencing one of the equations (say e), it is necessary to use e itself. It follows from [1] that a system of equations in solved form containing no loop is satisfiable. Hence the unsatisfiability of a system in improved

[5] A strict inequality $t_1 > t_2$ (resp. $t_1 < t_2$) could be transformed into a conjunction $t_1 - x^+ = t_2 \ \& \ t_1 \neq t_2$ (resp. $t_1 + x^+ = t_2 \ \& \ t_1 \neq t_2$). This is however much less efficient.

Class Name	Definition	Comments
$\mathcal{D}_{A1}$	$0 \neq a_0 + a_1 x$	One a-variable
$\mathcal{D}_{A2}$	$0 \neq a_0 + a_1 x_k + a_2 x_l + t_1$	At least one a-variable
	or $0 \neq a_0 + a_1 x + a_2 x^+ + t_2$	and at least two variables
$\mathcal{D}_{C1}$	$0 \neq a_0 + a_1 x^+$	Exactly one s-variable
$\mathcal{D}_{C2}$	$y = a_0 + a_1 x_k^+ + a_2 x_l^+ + t^+$	At least two s-variables

Table 3: Syntactic Classification of Disequations

Class Name	Invariant
$\mathcal{D}_{A1}$	No fixed a-variable wrt $\mathcal{E}$
$\mathcal{D}_{A2}$	$IsDeref(\mathcal{D}_{A2}, \mathcal{E})$
$\mathcal{D}_{C1}$	No fixed s-variable wrt $\mathcal{E}_s$
$\mathcal{D}_{C2}$	$IsDeref(\mathcal{D}_{C2}, \mathcal{E}_s)$

Table 4: Invariants for the Disequation Classes

solved form requires the presence of a loop. Given the invariants, this can only occurs in $\mathcal{E}_{A1}$ or $\mathcal{E}_{C1}$. This cannot occur neither $\mathcal{E}_{A1}$ (as the right-hand side of a constraint can only contain constraints from the sets $\mathcal{E}_{A2}$,$\mathcal{E}_{C1}$, or $\mathcal{E}_{C2}$) nor in $\mathcal{E}_{C1}$ (as the right-hand side contains only a s-variable).

Now if there exists a fixed (non-explicit) variable, this should be in $\mathcal{E}_s$, $\mathcal{E}_{A1}$, $\mathcal{E}_{A2}$ or $\mathcal{E}_{C2}$. It cannot be in $\mathcal{E}_{C2}$ since $\mathcal{E}_{C2} \cup \mathcal{E}_s$ is in solved form. Moreover the right-hand sides of $\mathcal{E}_{A2}$ does not contain any a-variable from $\mathcal{E}_{C2}$ and include at least one a-variable. Hence dereferencing $\mathcal{E}_{A2}$ wrt $\mathcal{E}_s$ cannot fix any variable. Since $\mathcal{E}_{A2}$ and $\mathcal{E}_{C2}$ have not fixed variable, $\mathcal{E}_{A1}$ cannot have fixed variables either. □

A direct consequence of the above theorem is the fact that the solved form can be obtained from the improved solved form by dereferencing. It now remains to define the improved solved form for disequations. It is fully symmetric to that of equations and depicted in Tables 3 and 4. Note however that there is no class $\mathcal{D}_F$ as these constraints can be trivially checked (no variables).

Once again we can recover the solved form from the improved solved form by dereferencing the disequations. We obtain the following result.

Theorem 20 A system of equations and disequations in improved solved form is satisfiable and has no fixed variables other than those made explicit.

4 The Basic Algorithm

We now describe the basic algorithm to update a system in improved solved form with a new equation. The key here is to make sure that we preserve the invariants for the constraint classes. The main function adds an equation.

```
AddEquation(in e:  Equation):  Boolean;
begin
e := Deref(e, E_F ∪ E_A1 ∪ E_A2 ∪ E_C1 ∪ E_C2);
if e does not contain variables then
   return SatisfiableTrivial(e)
```

```
else if e has at least one a-variable then
   return AddArbitrary(e)
else
   return AddSlack(e)
end;
```

Function SatisfiableTrivial returns true iff a constraint without variables is solvable. Note that the equation is not dereferenced wrt $\mathcal{E}_s$ at this point. This is not necessary and may entail redundant work.

```
AddArbitrary(in e: Equation): Boolean;
begin
let e be x = t with x being an a-variable not appearing in t in
   return ReviseEA2(x = t)
end;
```

The first step in adding an equation with a-variables amounts to isolating one a-variable. The set $\mathcal{E}_{A2}$ then needs to be revised. The sets $\mathcal{E}_{C1}$ and $\mathcal{E}_{C2}$ are not concerned with the new equation as made clear by the invariants.

```
ReviseEA2(x = t): Boolean;
begin
if not SatisfiableDA2(x = t) then
   return FALSE
else begin
   E_A2 := {x_1 = t_1 ∈ E_A2 | t_1 does not contain x};
   revised := {x = t} ∪ {x_1 = t_1 ∈ E_A2 | t_1 contains x};
   for each c in revised do
      begin
      c := Deref(c,{x = t});
      case c of
      type E_F:
         if not RevisedEF(c) then
            return FALSE;
         type E_A1:
            E_A1 := E_A1 ∪ {c};
         type E_A2:
            E_A2 := E_A2 ∪ {c};
         type E_C1, E_C2:
            RevisedEC2({c},E_S);
         end
      end
   end;
return TRUE
end;
```

The addition of a new equation in $\mathcal{E}_{A2}$ requires the checking of the disequations in $\mathcal{D}_{A2}$. Note that initially there is no need to consider $\mathcal{D}_{A1}$, $\mathcal{D}_{C1}$, and $\mathcal{D}_{C2}$. Disequations in $\mathcal{D}_{C1}$ and $\mathcal{D}_{C2}$ are not affected since their right-hand sides do not contain a-variables. Disequations in $\mathcal{D}_{A1}$ need only to be check in presence of a fixed variable. Each equation in $\mathcal{E}_{A2}$ containing the new basic variable needs to be reconsidered, possibly leading to new additions in the other sets.

```
SatisfiableDA2(x = t): Boolean;
begin
revised := {0 ≠ t1 ∈ D_A2 | t1 contains x};
D_A2 := {0 ≠ t1 ∈ D_A2 | t1 does not contain x};
for each c in revised do
   begin
   c := Deref(c,{x = t});
   case c of
      type trivial:
         if not SatisfiableTrivial(c) then
            return FALSE;
      type D_A1:
         D_A1 := D_A1 ∪ {c};
      type D_A2:
         D_A2 := D_A2 ∪ {c};
      type D_C1, D_C2:
         if not SatisfiableDC2({c},E_S) then
            return FALSE;
      end
   end;
return TRUE
end;
```

To check disequations in $\mathcal{D}_{A2}$ wrt a new equation, only the disequations containing the new basic a-variable need to be reconsidered. Note the need for further processing when only s-variables remain. This comes from the fact that the s-variables are not necessarily dereferenced.

```
RevisedEF(x = t): Boolean;
begin
Fixed := Deref({x1 = t1 ∈ E_A1 | t1 contains x},{x = t});
E_A1 := {x1 = t1 ∈ E_A1 | t1 does not contain x};
Fixed := Fixed ∪ {x = t};
E_F := E_F ∪ Fixed;
for each c ∈ Fixed do
   if not SatisfiableOneVar(c,D_A1)
      return false;
return true
end;
```

Function RevisedEF has to make sure that the invariant of $\mathcal{D}_{A1}$ is preserved since we have discovered a fixed variable. It also has to update $\mathcal{E}_{A1}$ which now contain fixed variables.

```
SatisfiableOneVar(x = t, D): Boolean;
begin
check := {0 ≠ t1 ∈ D | t1 contains x};
D := {0 ≠ t1 ∈ D | t1 does not contain x};
for each c ∈ check do
   if not SatisfiableTrivial(Deref(c,{x = t})) then
      return FALSE;
return TRUE
end;

RevisedEC2(D,R): Boolean;
begin
EC2 := EC2 \ D;
for each c in D do
   begin
   c := Deref(c,R);
   case c of
      type EF:
         if not SatisfiableOneVar(c,DA1) then
            return FALSE;
         EF := EF ∪ {c};
      type EC1:
         EC1 := EC1 ∪ {c};
      type EC2:
         EC2 := EC2 ∪ {c};
      end
   end;
return TRUE
end;
```

Function RevisedEC2 can also discover fixed a-variables in which case it is necessary to check $\mathcal{D}_{A1}$ to preserve the invariants. The extra generality provided by the set D in the function will be justified by its need when adding equations over s-variables.

```
SatisfiableDC2(D,R): Boolean;
begin
DC2 := DC2 \ D;
for each c in D do
   begin
   c := Deref(c,R);
   case c of
      type trivial:
         if not SatisfiableTrivial(c) then
            return FALSE;
      type DC1:
         DC1 := DC1 ∪ {c};
```

```
      type D_C2:
         D_C2 := D_C2 ∪ {c};
      end
   end;
return TRUE
end;
```

The second main function is AddSlack. It makes use of a function SIMPLEX(S,e) which, given a set S of equations over s-variables in solved form and a new equation, returns a triple $< S', FixedSet, satisfiable >$. *satisfiable* is true iff $S \cup \{e\}$ is satisfiable. If *satisfiable* is true, S' is a solved form for $S \cup \{e\}$ and *FixedSet* are the new fixed variables and their values in S' (i.e. equations of the form $y = a_0$ that were not in S). If there is no fixed variable, it is only necessary to preserve the invariant to check the disequations in $\mathcal{D}_{C2}$ and to revise $\mathcal{E}_{C2}$. Revising $\mathcal{E}_{C2}$ is necessary to detect fixed a-variables. Otherwise, in addition to the above check and revision, it is necessary for each of the fixed variables to revise $\mathcal{E}_{C1}$ and to check $\mathcal{D}_{C1}$.

```
AddSlack(e): Boolean;
begin
< E_S, FixedSet, Satisfiable > := SIMPLEX(E_S,e);
if not satisfiable then
   return FALSE
else if FixedSet = ∅ then
   return SatisfiableDC2(D_C2,E_S) and RevisedEC2(E_C2,E_S)
else begin
   for each f ∈ FixedSet do
      if not RevisedEC1(f) and not SatisfiableOneVar(f,D_C1) then
         return FALSE;
   return SatisfiableDC2(D_C2,E_S) and RevisedEC2(E_C2,E_S)
   end
end;
```

A possible variation to the above code could be to reconsider $\mathcal{D}_{A2}$ (resp. $\mathcal{E}_{A2}$) when fixed variables are discovered. This is not necessary but may transfer constraints from $\mathcal{D}_{A2}$ (resp. $\mathcal{E}_{A2}$) to $\mathcal{D}_{A1}$ (resp. $\mathcal{E}_{A1}$).

```
ReviseEC1(x = t): Boolean;
begin
revised := {x_1 = t_1 ∈ E_C1 | t_1 contains x};
E_C1 := {x_1 = t_1 ∈ E_C1 | t_1 does not contain x};
for each c ∈ revised do
   if not RevisedEF(Deref(c,{x = t})) then
      return FALSE;
return TRUE
end;
```

There are several possible variations in the above algorithms. For instance, CLP($\mathcal{Q}$) languages usually do not give access to s-variables which are only seen by the system. Hence there is no

need to keep fixed s-variables in $\mathcal{E}_S$ but it is then necessary to update the sets $\mathcal{D}_{A2}$ and $\mathcal{E}_{A2}$ when discovering fixed variables. The advantage is that there is no need to dereference constraints in $\mathcal{E}_{C1}$ (resp. $\mathcal{D}_{C1}$) in Function `ReviseE2` (resp. `SatisfiableDA2`).

The algorithm for adding a disequation is straightforward. The constraint has to be dereferenced first wrt all the sets but $\mathcal{E}_s$. If it contains only s-variables, dereferencing wrt $\mathcal{E}_s$ is then necessary.

5 Improvements to the Basic Algorithm

In this section, we consider two improvements to the basic algorithm and prove their correctness. The improvements, which concerns only s-variables, aims at removing unnecessary computation in frequent special cases. As the reader will notice, a constant difficulty is due to the possibility of fixed variables.

5.1 New Variables

5.1.1 Motivation

The first improvement concerns the case where the new equation with only slack variables contains a new variable, that is a variable not appearing in any of the sets of equations and disequations . We shall prove that, unless the new variable is fixed in the resulting system of equations, there is no need to check the disequations. Note that, in the case of a-variables, this improvement is performed implicitly by the above algorithm and this subsection originates from an attempt to see if that result holds for s-variables as well.

5.1.2 Basic Theorems

We first prove a useful theorem that establishes some form of connection between fixed variables and redundant constraints. In the following, we consider a system S of inequalities

$$AX \leq B \quad X \geq 0, B \geq 0$$

to which we associate a system E of equations

$$Y = B - AX \quad X, Y \geq 0, B \geq 0.$$

We denote by $S \models c$ (resp. $S \not\models c$) the fact that constraint c is (resp. is not) redundant wrt (or implied by) the system S.

Theorem 21 Assume that E is satisfiable without fixed variables and that $S \not\models t = 0$. Assume also that $E' = E \cup \{t = 0\}$ is satisfiable. Then $S \models t \geq 0$ or $S \models t \leq 0$ if and only if E' has fixed variables.

Proof $\Rightarrow$: Assume that $S \models t \geq 0$. Because E' is satisfiable, the minimum value of t is 0. Using the simplex algorithm to minimize t leads to an objective function with only nonnegative coefficients, at least one of them being non-zero (otherwise $S \models t = 0$ which contradicts the hypothesis). Hence there are fixed variables in E'. The case where $S \models t \leq 0$ is similar.

$\Leftarrow$: We have that E' has fixed variables while E does not. Consider the equation $t = 0$ and assume that t is lexicographically positive (i.e. its constant or its first non-zero coefficient is

positive). We can include in S the constraint $t \geq 0$ which is equivalent to adding the constraint $z = t$ in E giving E'' where z is a new s-variable. Without loss of generality, we can assume that z is lexicographically maximal among all s-variables. Now we try to minimize t which is bounded by 0 and two cases may arise. If the new row is never used during pivoting then clearly $S \models t \geq 0$ since we could have achieved the same pivoting without the new row. Otherwise, assume that the system chooses the row $z = t'$ to pivot after some steps. The objective function now becomes z and the system is still in solved form. Since E'' contains fixed variables there must exist a row of the form $y = z$ (otherwise we can remove z while still remaining in solved form which contradicts the hypothesis). But this means that we could have used that row to pivot giving y to optimize. The simplex algorithm would have terminated after the pivoting showing that $S \models t \geq 0$. The case where t is lexicographically negative is similar. □

The second theorem is in fact the basic result behind both improvements.

Theorem 22 Assume that

1. S is satisfiable and denote by $E \equiv \{y_i = t_i\}_{i=1}^m$ the system of equations over $x_1, \ldots, x_n$, associated with S and satisfiable without fixed variables.
2. $S \not\models t \geq 0$ and $S \not\models t \leq 0$;
3. $S' \equiv S \cup \{t = 0\}$ and E', its associated system of equations, is satisfiable without fixed variables.

Then $S \not\models d = 0$ and $S' \models d = 0$ implies that $d = 0$ and $t = 0$ represent the same hyperplane.

Proof By the redundancy theorem [6], $S' \models d = 0$ implies the existence of nonnegative rational numbers $\alpha^+, \alpha^-, \alpha_i, \alpha_i', \alpha_0, \beta^+, \beta^-, \beta_i, \beta_i', \beta_0$ such that

$$d = \alpha^+ t - \alpha^- t + \sum_{i=1}^{m} \alpha_i t_i + \sum_{i=1}^{n} \alpha_i' x_i + \alpha_0$$

$$-d = \beta^+ t - \beta^- t + \sum_{i=1}^{m} \beta_i t_i + \sum_{i=1}^{n} \beta_i' x_i + \beta_0.$$

It follows that

$$0 = \gamma^+ t - \gamma^- t + \sum_{i=1}^{m} \gamma_i t_i + \sum_{i=1}^{n} \gamma_i' x_i + \gamma_0$$

where $\gamma^+ = \alpha^+ + \beta^+$, $\gamma^- = \alpha^- + \beta^-$, $\gamma_i = \alpha_i + \beta_i$ and $\gamma_i' = \alpha_i' + \beta_i'$. By taking $\gamma^* = \gamma^- - \gamma^+$, it comes

$$\gamma^* t = \sum_{i=1}^{m} \gamma_i t_i + \sum_{i=1}^{n} \gamma_i' x_i + \gamma_0.$$

If γ^* is non-zero, it follows that $S \models t \geq 0$ or $S \models t \leq 0$ which contradicts the hypothesis. Hence, $\gamma^+ = \gamma^-$ which entails that all γ_i, γ_i' are zero. It remains that

$$d = \alpha^* t$$

with $\alpha^* = \alpha^+ - \alpha^-$ which implies that $d = 0$ and $t = 0$ represent the same hyperplane. □

We are now in position to justify the first improvement.

Theorem 23 Assume that

1. $S \cup \{d \neq 0\}$ is satisfiable and denote by $E \equiv \{y_i = t_i\}_{i=1}^m$ the system of equations over $x_1, \ldots, x_n$, associated with S and satisfiable without fixed variables.
2. $t = 0$ is a constraint containing a new variable x_{n+1} (i.e. x_{n+1} does not appear in E and d) with a non-zero coefficient.
3. $S' \equiv S \cup \{t = 0\}$ and E', its associated system of equations, is satisfiable without fixed variables.

Then, $S' \cup \{d \neq 0\}$ is satisfiable.

Proof Assume that $S' \models d = 0$. We are now in position to apply Theorem 22. Indeed, the first condition of Theorem 22 is implied by our first condition, the third conditions are the same, and, by Theorem 21, the second condition of Theorem 22 follows from our third condition. Moreover our first condition implies that $S \not\models d = 0$. Hence $d = 0$ and $t = 0$ represent the same hyperplane. But this is impossible since d does not contain x_{n+1} and x_{n+1} is not fixed. □

The above theorem, in conjunction with the independence of negative constraints theorem [1, 8][6], fully justifies the improvement.

5.1.3 Modification of the Algorithm

The above result can be exploited in the basic algorithm in a straightforward manner. If the simplex algorithm does not return any fixed variable, there is no need to check the set $\mathcal{D}_{C2}$[7]. Also there is no need to revise the set $\mathcal{E}_{C2}$ since none of the right-hand side expressions has reached a fixed value (this is a direct consequence of the above theorem). Hence we can update the basic algorithm to include the code.

```
AddSlack(e): Boolean;
begin
< E_S, FixedSet, Satisfiable > := SIMPLEX(E_S,e);
if not satisfiable then
   return FALSE
else if FixedSet = ∅ then
   if e does not contain a new variable then
      return SatisfiableDC2(D_C2,E_S) and RevisedEC2(E_C2,E_S)
   else
      return TRUE
else begin
   for each f ∈ FixedSet do
      if not RevisedEC1(f) and not SatisfiableOneVar(f,D_C1) then
         return FALSE
   return SatisfiableDC2(D_C2,E_S) and RevisedEC2(E_C2,E_S)
   end
end;
```

[6] This theorem intuitively means that disequations can be handled independently.

[7] Note that the explicit fixed variable in $\mathcal{E}_S$ do not matter since they do not appear in $\mathcal{D}_{C2}$.

Note that it is necessary to revise the invariants on the sets $\mathcal{E}_{C2}$ and $\mathcal{D}_{C2}$ which we now denote $\mathcal{E}'_{C2}$ and $\mathcal{D}'_{C2}$. The invariants on these last two sets are

$$c \in \mathcal{E}'_{C2} \Rightarrow Deref(c, \mathcal{E}_S) \text{ is of the type } \mathcal{E}_{C2} \text{ or } \mathcal{E}_{C1}$$

$$c \in \mathcal{D}'_{C2} \Rightarrow Deref(c, \mathcal{E}_S) \text{ is of the type } \mathcal{D}_{C2} \text{ or } \mathcal{D}_{C1}.$$

None of the other functions needs to be modified since $\mathcal{E}_{C2}$ and $\mathcal{D}_{C2}$ are considered each time $\mathcal{E}_{C1}$ and $\mathcal{D}_{C1}$ are considered.

5.2 Early Detection of Failures

In general, it would be of great benefit if the disequations could be checked wrt the new equation before the call to the simplex algorithm. In case of failures, this would reduce substantially the computation time and provide a more active use of disequations. The main problem we face here is that the set $\mathcal{E}_S$ has not yet assimilated the new constraint and hence it may seem difficult to check the disequations. However it turns out to be the case that, provided that the new equation does not introduce fixed variables, a disequation will fail in the updated system if and only if the disequation is syntactically identical (when dereferenced and to a normalization of the coefficients) to the new constraint. Hence it is possible to check, in a simple way (called "syntactic matching" in the following), the disequations before calling the simplex algorithm. The result is justified by Theorem 22.

The basic algorithm can be extended to include the above result as well. The procedure AddSlack should be divided into two parts, a part handling with the case of a new variable (the checking of disequations is not necessary provided that the resulting system of disequations do not contain additional fixed variables) and a part handling the case where all variables already appear in the constraints. In this second part, the call to the simplex algorithm can be postponed after the checking of disequations as in the following code.

```
AddSlack(e): Boolean;
begin
if e contains a new variable then
   begin
   < E_S, FixedSet, Satisfiable > := SIMPLEX(E_S,e);
   if not satisfiable then
      return FALSE
   else if FixedSet ≠ ∅ then
      begin
      for each f ∈ FixedSet do
         if not RevisedEC1(f) and not SatisfiableOneVar(f,D_C1) then
            return FALSE
      return SatisfiableDC2(D_C2,E_S) and RevisedEC2(E_C2,E_S)
      end
   else
      return TRUE
   end
else
```

```
  begin
  if SyntacticMatching(e,$\mathcal{D}_{C2} \cup \mathcal{D}_{C1}$) then
     return FALSE;
  $< \mathcal{E}_S, FixedSet, Satisfiable >$ := SIMPLEX($\mathcal{E}_S$,e);
  if not satisfiable then
     return FALSE
  else if FixedSet $\neq \emptyset$ then
     begin
     for each f $\in$ FixedSet do
        if not RevisedEC1(f) and not SatisfiableOneVar(f,$\mathcal{D}_{C1}$) then
           return FALSE
     return SatisfiableDC2($\mathcal{D}_{C2}$,$\mathcal{E}_S$) and RevisedEC2($\mathcal{E}_{C2}$,$\mathcal{E}_S$)
     end
  else
     return RevisedEC2($\mathcal{E}_{C2}$,$\mathcal{E}_S$)
  end
end;
```

Note the need to revise $\mathcal{E}_{C2}$ in almost all cases and the need to consider the disequations in case of fixed variables. The latter comes from the second assumption in the above theorem. From Theorem 21, we can conclude that the assumption is violated if there are fixed variables. This occurs in the case where the solution space loses two dimensions or more due to the new equation. This overhead is likely to be small since, on the one hand, the "syntactic matching" can be extremely fast if suitable data-structures are used and, on the other hand, the system of disequations is probably much smaller than the system of equations. It could be further reduced by a closer integration of the "syntactic matching" in the simplex function as many fixed variables are discovered after the dereferencing wrt $\mathcal{E}_s$. Finally let us mention that the "syntactic matching" requires the constraints to be dereferenced. Since the first improvement may leave the constraints undereferenced, it is useful to have a Boolean variable stating if the constraints are dereferenced or not in order to avoid unnecessary dereferencing

6 Conclusion

This paper was devoted to the (efficient) handling of disequations in a CLP language over linear Rational Arithmetic. We have presented a syntactic solved form for the constraints and described how to specialize it to avoid unnecessary dereferencing. An incremental algorithm for the improved solved form has been given that exploits that fact. Two significant improvements of the basic algorithm have been proposed. The first improvement amounts to avoiding checking the disequations and revising the equations when an equation with a new variable is added. The second improvement allows for checking the disequations before the call to simplex, providing an early detection of failures and a more active use of constraints. In both cases, fixed variables raise problems.

Further work include a substantial experimental evaluation of the above algorithm (which is the basis of a new implementation of CLP($\mathcal{Q}$)) and comparison with previous algorithms. Simultaneously, the detection of redundant constraints inside the solved form will be investigated. It turns out that the solved form has some inherent advantages over the usual simplex form to detect

redundant constraints due to its lexicographic ordering. Progress on this issue may further improve the efficiency of the above algorithms.

Acknowledgments

Initial work on the solved form took place at ECRC and G.I.A. Marseille-Luminy. We are grateful to A. Colmerauer, M. Dincbas, T. Graf, M. Henrion for fruitful discussions on this topic. The work reported here has been possible thanks to J. Cohen who provided the opportunity for the authors' collaboration.

References

[1] A. Colmerauer. Equations and Inequations on Finite and Infinite Trees. In *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS-84)*, pages 85–99, Tokyo, Japan, November 1984. ICOT.

[2] A. Colmerauer. An Introduction to Prolog III. *CACM*, 28(4):412–418, 1990.

[3] G.B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, Princeton, New Jersey, 1963.

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

[5] J. Jaffar, S. Michaylov, P.J. Stuckey, and R. Yap. The CLP($\Re$) Language and System. Research report, IBM, 1990.

[6] M.H. Karwan, V. Lofti, J. Telgen, and S. Zionts. *Redundancy in Mathematical Programming: a State-of-the-Art Survey*, volume 206 of *Lecture Notes in Economics and Mathematical Systems.* Springer Verlag, 1983.

[7] J.L. Lassez and K. Mc Aloon. Applications of a Canonical Form for Generalized Linear Constraints. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, Tokyo, Japon, December 1988.

[8] J.L. Lassez and K. McAloon. Independence of Negative Constraints. To appear.

[9] P.J. Stuckey. Incremental Linear Arithmetic Constraint Solving and Detection of Implicit Equalities. Submitted for publication, 1990.

[10] P. Van Hentenryck and T. Graf. Standard Forms for Rational Linear Arithmetics in Constraint Logic Programming. In *International Symposium on Artificial Intelligence and Mathematics*, Fort Lauderdale, Florida, January 1990. Also ECRC internal Report IR-LP-2217.

[11] P. Van Hentenryck and J.L. Imbert. On Redundant Constraints in Solved Forms for Linear Rational Arithmetics. (Forthcoming).