Applying Interval Arithmetic to Real, Integer and Boolean Constraints

Frédéric Benhamou*
William J. Older†
benhamouuniv-orleans.fr, wolderbnr.ca

Bell Northern Research

Accepted for publication in JLP 94/95

Abstract

We present in this paper a general narrowing algorithm, based on relational interval arithmetic, which applies to any n-ary relation on $\Re$. The main idea is to define, for every such relation ρ, a narrowing function $\overrightarrow{\rho}$ based on the approximation of ρ by a block which is the cartesian product of intervals. We then show how, under certain conditions, one can compute the narrowing function of relations defined in terms of unions and intersections of simpler relations. We apply the use of the narrowing algorithm, which is the core of the CLP language BNR-Prolog, to integer and disequality constraints, to boolean constraints and to relations mixing numerical and boolean values. The result is a language, called CLP(BNR), where constraints are expressed in a unique structure, allowing the mixing of real numbers, integers and booleans. We end by the presentation of several examples showing the advantages of such an approach from the point of view of the expressiveness, and give some computational results from a first prototype.

1 Introduction

The introduction of relational arithmetic within the Prolog language is strongly related to the Constraint Logic Programming scheme ([4, 6, 9, 10, 8, 23]). As

*Permanent address:Groupe d'Intelligence Artificielle, Faculté des Sciences de Luminy, case 901,163, avenue de Luminy,13288 Marseille Cedex 9 FRANCE, benham@gia.univ-mrs.fr, currently visiting BNR until January 1993

†Bell Northern Research, Computing Research Laboratory, PO Box 3511, Station C, K1Y 4H7 Ottawa, Ontario, Canada

it is now well-known, the CLP paradigm replaces the unification concept of the Prolog language by the notion of constraint resolution. Different algebraic structures have been tackled in the principal available CLP systems in order to improve Prolog's expressiveness and efficiency by adding constraint solving on specific domains. These systems provide processing of linear equations on rational and floating point numbers (Prolog III, CLP($\Re$)), polynomial constraints over real and complex numbers (CAL), non-linear and transcendental contraints applying to real intervals (BNR-Prolog), boolean constraints (CHIP, Prolog III), constraints on lists with concatenation (Prolog III), and finally constraints on finite domains (CHIP).

Some years after the birth of the concept, and as the interest in CLP applications is growing, some general remarks can be made. The first one is that the majority of the problems which seem to take advantage of the use of CLP comes from Operations Research. These problems generally include combinatorial aspects and the CLP approach requires the use of efficient constraint solvers over finite domains, especially on bounded integers. The second remark is that most of the time, the expressive power and efficiency of CLP systems is reduced by the strong partitioning of the structures in which constraints can be expressed. This means that one cannot express constraints involving discrete and continuous domains, that a boolean value cannot be involved in any numerical constraint, and that it is not possible to use the boolean value associated with a numerical relation in any boolean constraint.

We are interested here in the use of interval arithmetic in CLP. Functional interval arithmetic has been introduced by R. Moore [17] to deal with the incorect behaviours of finite precision arithmetic. To provide a relational model for numeric processing on intervals in Prolog, relational arithmetic on real intervals has been proposed by John Cleary in [5]. The two major drawbacks of Cleary's model are the constraint solving restriction to interval-convex relations (relations built from continuous, monotonic functions) and the use of non-logical variables which tends to separate constraint solving on intervals from the CLP scheme. W. Older and A. Vellino, in [19], discuss the introduction in BNR-Prolog of relational arithmetic on real intervals and propose a general theoretical framework which makes use of lattice theory to propose a fixed point semantics for the processing of interval constraint networks and generalizes interval narrowing to any relations. More recently, J.Lee and M. Van Emden ([11]) have focused on a logical semantics for interval narrowing by establishing links between relational interval arithmetic and existing CLP systems such as CLP($\Re$) and CHIP. Finally, G. Sidebottom and W. Havens propose in [21] to use Hierarchical Arc Consistency [13] to deal with constraint propagation on disjoint intervals.

The aim of this paper is to show that interval arithmetic can be used to define a CLP language in which expressiveness is significantly extended by allowing the user to express constraints on reals, integers and booleans (including booleans representing numerical relations) in a unified framework.

In section 2, we introduce the set $\mathcal{F}$ of F-intervals and show how any subset of $\Re$ can be approximated by an F-interval. We then extend the notion of approximation to subsets of $\Re^n$ and define, for every n-ary relation on $\Re$, a narrowing function which maps $\mathcal{F}^n$ to $\mathcal{F}^n$. We follow by proving the correctness, contractance, monotonicity and idempotence of these functions.

In section 3, we introduce constraints on real numbers, define the notion of stable set of constraints, and give an algorithm, which, given a finite set of constraints, terminates and produces either inconsistency or a stable set of constraints.

In section 4, we give two important properties of the narrowing function with respect to union and intersection and show on precise relations how this allows to compute effectively complex and non interval-convex relations. This is applied to disequations and integer constraints, and to Boolean constraints which can make use of numerical operations such as addition or multiplication. Amongst others, this gives a natural way to express cardinality constraints (see [24]). We end this section by introducing extended comparison relations which establish the opposite link between numerical constraints and Boolean constraints, i.e. allowing the programmer to use numerical relations in Boolean constraints.

Section 5 defines rapidly CLP(BNR), an extension of the Constraint Logic Programming language BNR-Prolog which includes the different types of constraint described above and illustrate the use of CLP(BNR) by presenting a certain number of program examples and computational results.

We conclude in section 6 and discuss future work on the subject.

2 Interval arithmetic

2.1 Preliminaries

We consider $\Re \cup \{-\infty, +\infty\}$, the set of real numbers augmented with the two infinity symbols, and the natural extension of the relation $\leq$ to this set. For every $a, b \in \Re \cup \{-\infty, +\infty\}, a \leq b$, we will use the following notations for intervals:

$$[a,b] = \{x \in \Re | a \leq x \leq b\}$$

$$[a,b) = \{x \in \Re | a \leq x < b\}$$

$$(a,b] = \{x \in \Re | a < x \leq b\}$$

$$(a,b) = \{x \in \Re | a < x < b\}$$

We will also use the notation $\langle a,b \rangle$ to denote an interval of any of the above defined forms.

Let $\mathcal{I}$ be the set of all intervals. Given a set $F = E \cup \{-\infty, +\infty\}$, where E is a finite subset of $\Re$, we call *F-interval* any element $\langle a,b \rangle$ of $\mathcal{I}$ such that $a \in F$

and $b \in F$. Let $\mathcal{F}$ be the set of F-intervals. We recall that the set inclusion relation is a partial ordering on real intervals, and thus on F-intervals. We call *vector* any finite sequence $(u_1, \ldots, u_n)$ of subsets of $\Re$. The ith component of any vector u is denoted by u_i. We call *interval vector* any vector such that every $u_i \in \mathcal{I}$, and *F-interval vector* any vector such that every $u_i \in \mathcal{F}$. Let $\mathcal{V}$ be the set of all F-interval vectors.

For every n-ary relation ρ on $\Re$, considered as a subset of $\Re^n$, the *ith projection* of ρ, denoted $\pi_i(\rho)$ is defined as follows:

$$\pi_i(\rho) = \{x_i \in \Re \mid (\exists x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n \in \Re) \text{ such that } (x_1, \ldots, x_n) \in \rho\}$$

We call *block* (resp. *F-block*) any n-ary relation ρ on $\Re$ such that there exists an interval vector (resp an F-interval vector) $(u_1, \ldots, u_n)$ verifying

$$\rho = u_1 \times \ldots \times u_n.$$

Let $\mathcal{B}$ be the set of all F-blocks.

For the sake of clarity, we will often thereafter denote any F-block as the F-interval vector made of its projections[1].

2.2 Approximations

In order to introduce the narrowing function associated with any n-ary relation on $\Re$ we define the approximation of any subset of $\Re$ by an F-interval, and extend it to the approximation of any subset of $\Re^n$ by a block.

For every relation ρ on $\Re$ considered as a subset of $\Re$, the approximation of ρ, denoted $\underline{\text{approx}}(\rho)$, is the smallest (w.r.t. the inclusion relation) F-interval containing the relation ρ.

The purpose of this definition is twofold. On one hand, if F-intervals are defined as being floating point intervals, this definition is closely akin to the ideas at the basis of relational interval arithmetic and, in the case where ρ is reduced to a singleton, introduces the approximation of real numbers by floating point intervals (see [5]). On the other hand, the generalization which allows ρ to be any relation on $\Re$ (ρ is not restricted to be an interval) will be used to deal with non interval-convex relations[2]. The $\underline{\text{approx}}$ function is then naturally extended to any n-ary relation on $\Re$ in the following way:

Definition 1 *For every $\rho \subset \Re^n$, the approximation of ρ, denoted $\underline{\text{approx}}(\rho)$ is the smallest (wrt the inclusion relation) F-block containing ρ.*

The existence of the approximation of any relation is based on the closure of $\mathcal{B}$ under intersection. It can also be shown that :

$$\underline{\text{approx}}(\rho) = (\underline{\text{approx}}(\pi_1(\rho)), \ldots, \underline{\text{approx}}(\pi_n(\rho))).$$

[1] However, let us mention that $\mathcal{B}$ and $\mathcal{V}$ are not isomorphic since for any F-interval vector $x = (x_1, \ldots, x_n)$, if any of the x_i's is the empty set, then x maps to the empty block.

[2] An interval-convex relation, as defined in [5], is a relation whose projections are intervals.

The principal properties of the $\underline{\text{approx}}$ function are monotonicity and idempotence, shown in the two following Lemmas.

Lemma 1 *Let ρ and ρ' be two n-ary relations on $\Re$. Then,*

$$\rho \subset \rho' \text{ implies } \underline{approx}(\rho) \subset \underline{approx}(\rho').$$

Proof: Since $\rho \subset \rho'$ and $\rho' \subset \underline{\text{approx}}(\rho')$, then $\rho \subset \underline{\text{approx}}(\rho')$.
Thus, by Definition 1, since $\underline{\text{approx}}(\rho)$ is the smallest block containing ρ and $\underline{\text{approx}}(\rho')$ is a block, then $\underline{\text{approx}}(\rho) \subset \underline{\text{approx}}(\rho')$. □

Lemma 2 *Let ρ be an n-ary relation on $\Re$. Then,*

$$\underline{approx}(\underline{approx}(\rho)) = \underline{approx}(\rho)$$

Proof: Straightforward, since an immediate consequence of Definition 1 is that for any F-block u, $\underline{\text{approx}}(u) = u$. □

Here follows another Proposition which establishes some properties of the approximation with respect to union and intersection[3]:

Property 1 *Let ρ, ρ' be two n-ary relations on $\Re$. Then,*

$$\underline{approx}(\rho \cup \rho') = \underline{approx}(\underline{approx}(\rho) \cup \underline{approx}(\rho')), \quad (1)$$

$$\underline{approx}(\rho \cap \rho') \subset \underline{approx}(\rho) \cap \underline{approx}(\rho'). \quad (2)$$

Proof: The left-right inclusion proof is straightforward in both cases[4]. Here follows the right-left proof for (1):
Since $\rho \subset \rho \cup \rho'$ and $\rho' \subset \rho \cup \rho'$, then, by Lemma 1,
$\underline{\text{approx}}(\rho) \subset \underline{\text{approx}}(\rho \cup \rho')$ and $\underline{\text{approx}}(\rho') \subset \underline{\text{approx}}(\rho \cup \rho')$,
thus, $\underline{\text{approx}}(\rho) \cup \underline{\text{approx}}(\rho') \subset \underline{\text{approx}}(\rho \cup \rho')$, and by Lemma 1 and 2,
$\underline{\text{approx}}(\underline{\text{approx}}(\rho)) \cup \underline{\text{approx}}(\underline{\text{approx}}(\rho')) \subset \underline{\text{approx}}(\rho \cup \rho')$. □

2.3 Narrowing

The definition of the narrowing function associated to every n-ary relation of $\Re$ is the basis of Relational Interval Arithmetic. Informally, given a relation ρ and an F-block u, the result of the narrowing function of ρ applied to u is the smallest F-block containing $u \cap \rho$. Here follows the definition and basic properties of narrowing functions:

[3] One can note that $\mathcal{B}$ is not closed under set union.

[4] Since $\rho \subset \underline{\text{approx}}(\rho)$ and $\rho' \subset \underline{\text{approx}}(\rho')$ implies $\rho \cup \rho' \subset \underline{\text{approx}}(\rho) \cup \underline{\text{approx}}(\rho')$ and $\rho \cap \rho' \subset \underline{\text{approx}}(\rho) \cap \underline{\text{approx}}(\rho')$. Applying Lemma 1 gives the result

Definition 2 *Let ρ be an n-ary relation on $\Re$. The* narrowing function *of ρ is the function $\overrightarrow{\rho}, \mathcal{F}^n \longrightarrow \mathcal{F}^n$, such that for every F-block u,*

$$\overrightarrow{\rho}(u) = \underline{approx}(u \cap \rho).$$

The main properties of the narrowing functions are Contractance (the narrowed intervals are smaller than the initial intervals), Correctness (every real solution lies in the narrowed intervals), Monotonicity (the narrowing preserves the inclusion) and Idempotence (the narrowed intervals have to be computed but once), as expressed by the following theorem.

Theorem 1 *For every $\rho \in \Re^n$, and every F-blocks u, v,*

$$(1) \quad \overrightarrow{\rho}(u) \subset u, \qquad \textit{(Contractance)}$$

$$(2) \quad u \cap \rho = \overrightarrow{\rho}(u) \cap \rho, \qquad \textit{(Correctness)}$$

$$(3) \quad u \subset v \textit{ implies } \overrightarrow{\rho}(u) \subset \overrightarrow{\rho}(v), \qquad \textit{(Monotonicity)}$$

$$(4) \quad \overrightarrow{\rho}(\overrightarrow{\rho}(u)) = \overrightarrow{\rho}(u). \qquad \textit{(Idempotence)}$$

Proof: (1) Contractance
$u \cap \rho \subset u$, then
$\underline{\text{approx}}(u \cap \rho) \subset \underline{\text{approx}}(u)$ (Lemma 1), and
$\overrightarrow{\rho}(u) \subset u$ (since u is an F-block).

(2) Correctness
Let us show first that $u \cap \rho \subset \overrightarrow{\rho}(u) \cap \rho$.
$\overrightarrow{\rho}(u) = \underline{\text{approx}}(u \cap \rho)$, (Definition 2),
$\underline{\text{approx}}(u \cap \rho) \supset u \cap \rho$, (Definition 1), thus,
$\underline{\text{approx}}(u \cap \rho)) \cap \rho \supset u \cap \rho$.

Then, we show that $u \cap \rho \supset \overrightarrow{\rho}(u) \cap \rho$.
$\underline{\text{approx}}(u \cap \rho) \subset u$. (Contractance), thus,
$\overrightarrow{\rho}(u) \cap \rho \subset u \cap \rho$.

(3) Monotonicity
This property is a direct consequence of Lemma 1. In effect, we have
$u \subset v$, thus $u \cap \rho \subset v \cap \rho$, then by Lemma 1,
$\underline{\text{approx}}(u \cap \rho) \subset \underline{\text{approx}}(v \cap \rho)$

(4) Idempotence
A direct consequence of the Contractance property is that:
$\overrightarrow{\rho}(\overrightarrow{\rho}(u)) \subset \overrightarrow{\rho}(u)$

Thus, we have to show that $\overrightarrow{\rho}(\overrightarrow{\rho}(u)) \supset \overrightarrow{\rho}(u)$
Since $u \cap \rho \subset \underline{\text{approx}}(u \cap \rho)$ (Definition 1), then
$u \cap \rho \subset \underline{\text{approx}}(u \cap \rho) \cap \rho$. Thus, by Lemma 1,
$\underline{\text{approx}}(u \cap \rho) \subset \underline{\text{approx}}(\underline{\text{approx}}(u \cap \rho) \cap \rho)$, and finally
$\overrightarrow{\rho}(u) \subset \overrightarrow{\rho}(\overrightarrow{\rho}(u))$. □

We can now use the narrowing of one particular relation to simplify sets of relations, as shown in the following section.

3 Applying narrowing to constraint systems

3.1 Constraint systems

Let V be an infinite countable set of variables representing real numbers, and $F = E \cup \{-\infty, +\infty\}$, where E is a finite subset of $\Re$.

Definition 3 *A* constraint *is an expression of the form $\rho(x_1, \ldots, x_n)$, where ρ is a n-ary relation on $\Re$, and every x_i is either a variable from V or a constant from E.*

Choosing $\Re$ and not $\mathcal{F}$ as the domain on which the constraints are defined allows us to make a natural link with Prolog, without the various drawbacks expressed in [5] (use of non-logical variables, modification of the unification algorithm, etc.).

Definition 4 *A* system Σ *is a pair (i, S), where i is a mapping from $V \cup E$ into $\mathcal{F}$, and S is a finite set of constraints.*

Definition 5 *A* solution σ *of a constraint system $\Sigma = (i, S)$ is a mapping from $V \cup E$ into $\Re$ verifying:*

$$\forall x \in E, \sigma(x) = x$$
$$\forall x \in V, \sigma(x) \in i(x)$$
$$\forall \rho(x_1, \ldots, x_n) \in S, (\sigma(x_1), \ldots, \sigma(x_n)) \in \rho$$

It can be noted that a constraint system can also be defined as a mere finite set of constraints, since i can be defined as a set of unary relations stating that a variable must lie between two given bounds. However, although simplifying the definition, this approach complicates the algorithm and is quite far from the actual implementation. The purpose of the narrowing algorithm is, given a constraint system, to compute a fixed point called *stable system* whose definition follows:

Definition 6 *A system $\Sigma = (i, S)$ is* stable *iff for every constraint $\rho(x_1, \ldots, x_n) \in S$, if $u = (i(x_1), \ldots, i(x_n))$, then*

$$\overrightarrow{\rho}(u) = u$$

Algorithm

input: a stable system $\Sigma = (i, S)$,
a constraint c

$S \leftarrow S \cup \{c\}, C \leftarrow \{c\}, i' = i$
While $C \neq \emptyset$ do
  Choose any constraint $c' = \rho(x_1, \ldots, x_n)$ from C
  $u \leftarrow (i'(x_1), \ldots, i'(x_n))$
  $v \leftarrow \overrightarrow{\rho}(u)$.
  If any of the v_i is empty then stop, the system $(i, S \cup \{c\})$ is inconsistent.
  For every variable x_j in $\{x_1, \ldots, x_n\}$ Do
    If $v_j \neq i(x_j)$ then
      $i'(x_j) \leftarrow v_j$
      For every constraint c'' in S in which x_j appears Do $C \leftarrow C \cup \{c''\}$.
    EndIf
  EndFor
  $C \leftarrow C - \{c'\}$
EndWhile

output: inconsistency or
a stable system $\Sigma' = (i', S \cup \{c\})$

Figure 1: A narrowing algorithm

3.2 A narrowing algorithm

The algorithm which is used to compute stable sets of constraints is basically the one which is used in BNR-Prolog(see [19]), and is quite close to local consistency algorithms (see [12, 15, 16]) applied to infinite bounded domains and n-ary relations. The incremental version of the algorithm is described in figure 1.

This algorithm verifies the following properties:

1. The algorithm trivially terminates, since narrowing functions are contractant and the number of computable F-intervals is finite.

2. The algorithm, as shown in [19] reaches a fixed point which, due to the monotonicity of narrowing functions is unique and does not depend on the order in which the constraints are chosen.

3. Due to the correctness of narrowing functions, every solution of $(i, S \cup \{c\})$ is a solution of Σ'.

However, this narrowing procedure, like other local consistency checks is not strong enough to guarantee, in the general case, the completeness of the

algorithm which can compute a stable system $\Sigma' = (i', S \cup \{c\})$ where $S \cup \{c\}$ is inconsistent.

4 Constraints on reals, integers and Booleans

The previous sections define narrowing functions but do not give any indications on the type of relations they can handle usefully and do not provide any way to compute them. This is the object of this section. Since we are interested here in more practical issues, we will consider F-intervals as being floating point intervals, for any given floating point representation.

Interval arithmetic, as presented in [5], generally restricts the definition of the narrowing function to these relations ρ such that for every floating point vector u, every projection of $\rho \cap u$ is an interval vector. These relations are called *interval convex* relations. Here follows the definitions for interval-convexity and its natural extension to F-interval convexity.

Definition 7 *A n-ary relation ρ on $\Re$ is* interval convex *(resp. F-interval convex) if for every block (resp F-block) u and every i in $\{1, \ldots, n\}$, $\pi_i(\rho \cap u)$ is an interval (resp. an F-interval).*

When restricted to this case, the equivalent of the approximation function is based on an "outward rounding" function which associates to any interval I the smallest floating point interval J such that $I \subset J$. Examples of interval convex relations are:

$$\text{add} = \{(x, y, z) \in \Re^3, x + y = z\},$$

$$\text{leq} = \{(x, y) \in \Re^2, x \leq y\},$$

$$\text{le} = \{(x, y) \in \Re^2, x < y\},$$

$$\text{eq} = \{(x, y) \in \Re^2, x = y\}$$

For example, for any floating point vector $u = (u_1, u_2, u_3)$ the resulting floating point vector $v = (v_1, v_2, v_3)$ after applying $\overrightarrow{\text{add}}$ is given below:

$$v_1 = u_1 \cap (u_3 \ominus u_2),$$

$$v_2 = u_2 \cap (u_3 \ominus u_1),$$

$$v_3 = u_3 \cap (u_1 \oplus u_2),$$

Where $\oplus$ and $\ominus$ are the regular interval addition and interval substraction[5]. Similar formulas allow to compute the narrowing functions of the other relations

[5]For a more detailed definition of these functions one can see [17].

cited above. Some usual relations are much more difficult to deal with, for example multiplication. The natural relational definition of multiplication is:

$$\text{mult} = \{(x, y, z) \in \Re^3, z = x \times y\}.$$

The mult relation is not interval convex and for every F-block u, the projections of mult $\cap\, u$ are generally not intervals, but disjunctions of intervals, as it can be verified in the following example cited in [5]:

if $u = ([-2, 3], [-\infty, +\infty], [1, 1])$, then
$\pi_1(\text{mult} \cap u) = [-2, 3]$,
$\pi_2(\text{mult} \cap u) = [-\infty, -1/2] \cup [1/3, +\infty]$,
$\pi_3(\text{mult} \cap u) = [1, 1])$

However, as it is suggested in [5], one can express *mult* as the union of two interval convex relations noted mult^+ and mult^-, whose definition is given above and whose narrowing projections algorithms are precisely presented in [5]:

$$\text{mult}^+ = \{(x, y, z) \in \Re^3, x \geq 0, x \times y = z\},$$

$$\text{mult}^- = \{(x, y, z) \in \Re^3, x < 0, x \times y = z\}.$$

The solution proposed in [5] and also in [11] is to deal with this union of interval-convex relations by choosing one of these sub-relations, and postpone the processing of the other relation by effectively creating a Prolog choice point, or any other explicit backtracking procedure. In contrast, making use of the definitions proposed above, the computation of the narrowing function on the same example gives the following result:

$$\overrightarrow{\text{mult}}(u) = ([-2, 3], [-\infty, +\infty], [1, 1])$$

The aim of the following section is to formally describe how one can compute such relation decompositions.

4.1 Union and intersection of interval-convex relations

As it will be shown in the next sections, it is crucial to be able to make use of the expression of any relation in terms of unions and intersections of simpler (i.e. interval convex) relations. In this section, we present two results. The first gives a way to compute the union of two relations, and the second expresses conditions under which a similar processing can be done with respect to intersection. Here follows the decomposition Property:

Property 2 (Decomposition) *Let ρ and ρ' be two n-ary relations on $\Re$. Then, for every block u:*

$$\overrightarrow{\rho \cup \rho'}(u) \quad = \quad \underline{approx}(\overrightarrow{\rho}(u) \cup \overrightarrow{\rho}'(u)) \tag{3}$$

Proof:

$$\begin{aligned}
\overrightarrow{\rho \cup \rho'}(u) &= \underline{\text{approx}}(u \cap (\rho \cup \rho')) \\
&= \underline{\text{approx}}((u \cap \rho) \cup (u \cap \rho') \\
&= \underline{\text{approx}}(\underline{\text{approx}}(u \cap \rho) \cup \underline{\text{approx}}(u \cap \rho')), (Property\ 1) \\
&= \underline{\text{approx}}(\overrightarrow{\rho}(u) \cup \overrightarrow{\rho'}(u))
\end{aligned}$$

□

With respect to intersection, a similar reasoning leads tothe following property (the equality does not hold in the general case) :

$$\overrightarrow{\rho \cap \rho'}(u) \subset \overrightarrow{\rho}(u) \cap \overrightarrow{\rho'}(u) \tag{4}$$

It is often mentioned that to deal with relations which are expressed in terms of intersection of interval-convex relations, one can decompose them and apply the narrowing algorithm. This way of doing does not guarantee to compute the narrowing of the intersection in the general case, as it is shown in the example below. Consider the relation $\rho = \rho_1 \cap \rho_2$, where:

$$\rho_1 = \{(x, y) \in \Re^2 \mid x = y\} \text{ and } \rho_2 = \{(x, y) \in \Re^2 \mid x = -y\}$$

Let us now consider $u = ([-1, 1], [-1, 1])$. It is clear that $\overrightarrow{\rho}(u) = ([0, 0], [0, 0])$, but applying the narrowing algorithm on the system $S = \{x \in [-1, 1], y \in [-1, 1], x = y, x = -y\}$, we are not able to narrow any of the two intervals.

However, in an interesting number of special cases, this way of computing the intersection is correct, as it is shown in the following. We first introduce the notion of i-dependency.

Definition 8 (i-dependancy) *A n-ary relation ρ on $\Re$ is* i-dependant *($i \in \{1, \ldots, n\}$), iff, $\forall (x_1, \ldots, x_n) \in \Re^n, \forall y \in \Re$,*

$$(x_1, \ldots, x_n) \in \rho \Longleftrightarrow (x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n) \in \rho$$

The negation of i-dependancy is called *i-independancy*. An immediate consequence of this definition is the following property we give without proof.

Property 3 *Let ρ be an n-ary relation on $\Re$, then, for all i in $\{1, \ldots, n\}$, the two following propositions are equivalent:*

1. *ρ is i-dependant*
2. *there exists a block u such that $\pi_i(\rho \cap u) \neq \emptyset$ and $\pi_i(\rho \cap u) \neq \pi_i(u)$.*

Then we show the following theorem (the proof is given in Appendix):

Theorem 2 (Composition) *Let ρ and ρ' be two F-interval convex, n-ary relations on $\Re$. If there exists at most one i in $\{1,\ldots,n\}$ such that ρ and ρ' are i-dependant then for every F-Block u:*

$$\overrightarrow{\rho \cap \rho'}(u) = \overrightarrow{\rho}(\overrightarrow{\rho'}(u)) \cap \overrightarrow{\rho'}(\overrightarrow{\rho}(u))$$

A simple example of application of this theorem is the computation of the following relation[6]:

$$\rho = \{(x,y) \in \Re^2 \mid x = y\} \cap \{(x,y) \in \Re^2 \mid x \geq 0\}$$

Let us call $\rho_1 = \{(x,y) \in \Re^2 \mid x = y\}$ and $\rho_2 = \{(x,y) \in \Re^2 \mid x \geq 0\}$. If we consider $u = ([-2,1],[-1,2])$ then

$$\begin{aligned}
\overrightarrow{\rho_1}(u) &= ([-1,1],[-1,1]),\\
\overrightarrow{\rho_2}(\overrightarrow{\rho_1}(u)) &= ([0,1],[-1,1]),\\
\overrightarrow{\rho_2}(u) &= ([0,1],[-1,2]),\\
\overrightarrow{\rho_1}(\overrightarrow{\rho_2}(u)) &= ([0,1],[0,1]),\\
\overrightarrow{\rho_2}(\overrightarrow{\rho_1}(u)) \cap \overrightarrow{\rho_1}(\overrightarrow{\rho_2}(u)) &= ([0,1],[0,1])
\end{aligned}$$

It is easy, in this case, to verify that $\overrightarrow{\rho}(u) = ([0,1],[0,1])$

4.2 Integer constraints and disequality

The efficient processing of constraints on finite domains, and thus on bounded integers is one of the most important functionalities required in CLP to solve many constraint problems. A narrowing algorithm for integer and disequality relations is suggested by John Cleary in [5], with the conclusion that the application of interval narrowing to these fundamentaly non interval-convex relations is probably worthless due to the great number of created choice points, and the weakness of the narrowing applied to disequality.

However, the narrowing approach presented here computes integer constraints quite efficiently. Informally, the effect of narrowing on variables that are constrained to represent integer values is to reduce their domain to closed intervals whose bounds are integers. Here follows the description of the algorithms to compute the narrowing function for integer (denoted by *int*). If $u = \langle a,b \rangle$ is an F-interval, then:

$$\overrightarrow{\text{int}}(u) = [\lceil a' \rceil, \lfloor b' \rfloor]$$

[6] this is, in fact one half of the relation "absolute value",

$$\begin{aligned}
\text{abs} \quad = \quad & (\{(x,y) \in \Re^2 \mid x \geq 0\} \cap \{(x,y) \in \Re^2 \mid x = y\}) \cup\\
& (\{(x,y) \in \Re^2 \mid x < 0\} \cap \{(x,y) \in \Re^2 \mid x = -y\})
\end{aligned}$$

where $\lceil a' \rceil$ is the smallest integer greater or equal than a', $\lfloor b' \rfloor$ is the greatest integer smaller or equal than b', and
$a' = a + 1, b' = b$ if $u = (a, b]$ and a and b are integers,
$a' = a, b' = b - 1$ if $u = [a, b)$ and a and b are integers,
$a' = a + 1, b' = b - 1$ if $u = (a, b)$ and a and b are integers,
$a' = a, b' = b$ if $u = \langle a, b \rangle$ and u is not of one of the above forms.

In contrast with the continuous domains, where the approximation of reals by intervals makes the disequality relation practically useless, disequations are of great importance in the case of finite domains as it has been shown on many examples. The disequality relation is given by neq= $\{(x, y) \in \Re^2 \mid x \neq y\}$. It can also be expressed as the union of two already defined interval convex relations as follows:

$$\text{neq} = \{(x, y) \in \Re^2 \mid x < y\} \cup \{(x, y) \in \Re^2 \mid x > y\}$$

Thus, the Theorem 2 can be applied to compute $\overrightarrow{\text{neq}}$. Practically, let us consider two variables in the relation neq. If one of their domains is reduced to one value which is a closed bound of the other domain, then this second domain is narrowed by removing that value. For example:

$$\begin{aligned} x \in (0, 2.12], \text{integer}(x), x \neq 2 &\implies x = 1, \\ x \in (0, 3.99], \text{integer}(x), x \neq 2 &\implies x \in [1, 3]. \end{aligned}$$

4.3 Boolean constraints

The introduction of Boolean constraints into this framework is done by considering Boolean variables as integers whose possible values are taken in $[0, 1]$. The Boolean relations are defined as follows:

$$\text{and} = \min = \{(x, y, z) \in \Re^3, \min(x, y) = z\}$$

$$\text{or} = \max = \{(x, y, z) \in \Re^3, \max(x, y) = z\}$$

$$\text{not} = \{(x, y) \in \Re^3, y = 1 - x\}$$

The design of an algorithm to compute *min* and *max* is not trivial. A case analysis based on the comparisons of the six involved bounds leads to the study of ninety different cases. However, we can apply Property 2 and Theorem 2 since *min* and *max* can be expressed in the following way:

$$\begin{aligned} \min &= \big(\{(x, y, z) \in \Re^3, x \leq y\} \cap \{(x, y, z) \in \Re^3, z = x\}\big) \cup \\ &\quad \big(\{(x, y, z) \in \Re^3, x > y\} \cap \{(x, y, z) \in \Re^3, z = y\}\big) \\ \max &= \big(\{(x, y, z) \in \Re^3, x \geq y\} \cap \{(x, y, z) \in \Re^3, z = x\}\big) \cup \\ &\quad \big(\{(x, y, z) \in \Re^3, x < y\} \cap \{(x, y, z) \in \Re^3, z = y\}\big) \end{aligned}$$

The narrowing algorithm applied to such defined Boolean constraints, when associated with Boolean enumeration, is comparable to local consistency algorithms based on the Davis and Putnam procedure (see for example [14]).

Basic examples of Boolean constraint narrowing are:

$$\begin{aligned} x \vee 1 = y &\implies y = 1 \\ x \vee 0 = 1 &\implies x = 1 \\ x \wedge y = 1 &\implies x = 1, y = 1 \\ x \vee y = 0 &\implies x = 0, y = 0 \end{aligned}$$

Moreover, based on the fact that Booleans are defined as numbers, one can express relations between Boolean variables by using numerical relations. For example, to state that in a sequence $(x_1, \ldots, x_n)$ of Boolean values, at least a of them and at most b of them must be true, one can write the following constraint[7]:

$$a \leq x_1 + \ldots + x_n \leq b$$

Finally, here are some other usual Boolean functions[8]:

$$\begin{aligned} a \Rightarrow b &: a \leq b, \\ \text{if } a \text{ then } b \text{ else } c &: (2 - a - b) \times (1 - c + a) = 0, \\ a \text{ xor } b &: a \neq b. \end{aligned}$$

4.4 Extended numerical relations

One other important feature missing in the standard CLP systems is the possibility to use comparison relations in Boolean constraints and thus to express constraints like: given three numbers x, y and z, if $x \leq y$ then $x \leq z \leq y$ else $(x \leq z) \vee (z \leq y)$, which can be expressed as[9]:

$$(\neg(x \leq y)) \vee (x \leq z \leq y)) \wedge ((x \leq y) \vee ((x \leq z) \vee (z \leq y))) = 1,$$

or more simply:

$$(z \geq x) + (z \leq y) = (x \leq y) + 1$$

The processing of such constraints is useful as soon as one considers problems mixing numbers and Boolean values. One way to introduce these constraints is to modify the definition of such relations as equality, inequality and disequality by considering ternary relations involving one Boolean parameter. Here are

[7] This is the Boolean expression of the cardinality operator proposed in [24].

[8] Assuming that $\leq$ and $\neq$ are ternary relations whose third parameter is a Boolean value, as it will be developped in the next sections.

[9] See previous note.

some of these relations:

$$\begin{aligned} \text{eq} &= \big(\{(x,y,z) \in \Re^3, x = y\} \cap \{(x,y,z) \in \Re^3, z = 1\}\big) \cup \\ &\quad \big(\{(x,y,z) \in \Re^3, x \neq y\} \cap \{(x,y,z) \in \Re^3, z = 0\}\big) \\ \text{geq} &= \big(\{(x,y,z) \in \Re^3, x \geq y\} \cap \{(x,y,z) \in \Re^3, z = 1\}\big) \cup \\ &\quad \big(\{(x,y,z) \in \Re^3, x < y\} \cap \{(x,y,z) \in \Re^3, z = 0\}\big) \end{aligned}$$

Here again, we can apply Property 2 and Theorem 2 to compute correctly this type of constraints.

5 Examples and computational results.

In this section, we present some examples of the possible use of the different constraints described above, and we give some computational results, based on a first prototype of a language called CLP(BNR), which is an extension of BNR-Prolog (see [19]) which includes the processing of the integer and Boolean constraints described in the first part of the paper. The syntax is a superset of the standard Edinburgh Prolog syntax and we will assume the reader is familiar with the notions of variables, constants, terms, lists, rules, programs, queries and their usual notations. The additional constraints include unary type constraints, Boolean relations, and ternary numerical relations whose third parameter is a Boolean. A functional notation of the constraints is syntacticaly provided to simplify the writing of programs. A more detailed presentation of CLP(BNR)from the user point of view can be found in [20].

From the implementation point of view, this prototype consists mainly of a BNR-Prolog top-level implementation of interval arithmetic with specific assembler subroutines to compute basic narrowing functions. Among other implementation imperfections, there is no specific way to compute inequalities and the actual processing of integers, and thus Booleans, is based on the general BNR-Prolog floating-point representation. This explains why the following computational results are much slower than what we could expect from a final integrated implementation of the system. These results are mainly given here to provide a general idea on performances and experimental complexity analysis.

The results have been computed on a standard Macintosh II (2 Mips). For the problems presented below, we have separated the set-up part, where Prolog behaves like a macro-processor and builds constraint systems, and the enumeration part, where the actual solutions are computed. Most of the time, the result tables indicate the set-up time and the execution times in seconds and the number of necessary backtrackings to find the first solution and all solutions (including the proof that they are all computed). However, on some examples some of these results have been omited for practical reasons.

We end this preamble by saying a few words on enumeration. As it is well-known, local consistency algorithms on finite domains are complete when they

are used together with enumeration procedures which compute non deterministically the actual solutions of the considered systems. In fact, a massive part of the interest of such approaches relies on the efficiency of the enumeration, and thus on a number of particular heuristics. Among these, the most used is the "first-fail principle" which consists in enumerating first the more constrained variables (the variables whose domain is the smallest and/or the variables appearing in the greatest number of constraints). As the processing of disequalities, as described here, is weaker than the mere deletion of the prohibited values from the corresponding domains[10] and due to our top-level implementation, it is more difficult in our case to select the more constrained variables. We have thus implemented an approximation of the first-fail principle which consider first the variables with the smallest difference between their upper and lower bounds. A more sophisticated implementation could perfectly involve a domain bit-map in the integer case and apply this heuristic more accurately.

5.1 Linear arithmetic on integers

Even though the originality of the system presented here is ita ability to tackle constraints which do not fit in the usual specific integer linear case, we just give here some benchmarks for linear problems to give a general idea of its performances. As it has been stated before, it is also clear that our aim here is not to compete, from the efficiency point of view, with specialized systems, like CHIP, whose implementation has been particularly studied (see for example [1]). The first example with which we propose to illustrate integer constraints is the well-known cryptarithmetic problem DONALD +GERALD = ROBERT, whose purpose is to give a different value, taken between 0 and 9 to each letter, in order to verify the corresponding additions. The second one is the too famous N queens problem, which, totally based on integer disequalities, is certainly not one of our best cases. Here follows the computation results for these problems:

Problem	Set-up time	First solution		All solutions	
		Back	Enum time	Back	Enum time
DONALD	1.75 s	2	0.38 s	8	0.90 sec
12 Queens	6.73 s	58	26.26 s	n/a	n/a
14 Queens	10.05 s	49	24.73 s	n/a	n/a
16 Queens	14.33 s	50	26.76 s	n/a	n/a

Due to the very big number of acceptable solutions to the N-Queens problem, as soon as $N \geq 12$, the computation of all of them is quite unrealistic[11].

[10] The computed intervals are only narrowed when the value is also one of the interval's bounds.

[11] To give an idea, the algorithm proposed in [14] finds the 14200 solutions for N = 12 in 4:30 hours.

5.2 Non-linear arithmetic on integers

As a first example of non-linear constraints on integers, we propose the following problem. Find n integers $x_1, \ldots, x_n, 1 \leq x_i \leq n$, verifying the two following conditions:

$$\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} i \quad , \quad \prod_{i=1}^{n} x_i = \prod_{i=1}^{n} i.$$

Any permutation of $(1, \ldots, n)$ is an obvious solution, but as n is growing, there are other solutions. To avoid the computation of symmetrical solutions, we impose the following constraint: $x_1 \leq x_2 \leq \ldots, \leq x_n$. The first n for which there is more than one solution is 9. Here follow the results for this program:

N	Set-up	First solution		All solutions		Nb Sol
		Back	Enum time	Back	Enum time	
9	0.98 s	115	6.16 s	392	19.73 s	2
10	1.11 s	339	17.61 s	1085	55.83 s	6
11	1.25 s	1025	53.35 s	3179	163.80 s	6
12	1.31 s	1423	77.83 s	9323	487.30 s	22

5.2.1 Pythagorean triples and other Diophantine equations

Two other interesting problems are the following ones:

1. Find three positive integers x, y, z such that:
$$x^2 + y^2 = z^2 \tag{5}$$
2. find four positive integer x, y, w, z which are solutions of the equation:
$$x^2 + y^2 + w^2 = z^2, \tag{6}$$

The CLP(BNR) program to solve these two equations, assuming the considered integers are bounded, is straightforward. In the first case (Equation 5), it is well known that the solutions are infinitely many and can be generated by the following equations:

$$x = u^2 - v^2, y = 2uv, z = u^2 + v^2,$$

where u and v are any integers whatsoever satisfying the conditions that $u > v > 0, (u, v) = 1$, and one of u and v is even (see [22]). The second problem is more interesting since, as far as we know, the infinitely many solutions cannot be generated with any other equations. Figure 2 shows the program to find all integers (bounded by a positive integer n) verifying Equation 6. We can assume, without loss of generality that: $x \leq y \leq w$. Here follow the computation results for these two problems:

```
diophantine(N):-
[X,Y,Z,W]:integral(1,N),
X**2+Y**2+W**2==Z**2,
W >= Y, Y >= X,
enumerate([X,Y,W]).
```

Figure 2: Program for $x^2 + y^2 + w^2 = z^2$

N	Equation 5			Equation 6		
	Enum time	Back.	Nb Sol.	Enum time	Back.	Nb Sol.
20	0.65 s	13	6	3.36 s	89	22
50	2.78 s	37	20	30.00 s	564	141
100	9.23 s	87	52	200.60 s	2299	573
200	31.90 s	192	127	n/a	n/a	n/a
500	179.96 s	559	386	n/a	n/a	n/a

5.3 Boolean constraints

We have tested our prototype on classical Boolean benchmarks. It should be noted that we have used, as often as possible, the addition on boolean variables to express cardinality constraints. Queens is the Boolean version of the N-Queens problem previously described. Schur refers to the Schur Lemma: Considering the N first integers, if one wants to place them in three boxes in such a way that, for every box, and for every integers a and b, a and $2a$ are not in the same box, and if a and b are in the same box, then $a + b$ is not. Pigeon is the pigeon-hole problem, where n pigeons have to be placed in p holes, with the condition that only one pigeon can be placed in one hole. Here follows the computational results for these benchmarks:

Problem	Set-up	First solution		All solutions		Nb Sol
		Back	Enum	Back	Enum	
queens 8	1.93 s	21	0.94 s	391	19.56 s	92
Schur 13	2.45 s	9	0.46 s	179	6.76 s	18
Schur 14	2.91 s	-	-	179	6.88 s	0
pigeons 7/7	0.84 s	0	0.93 s	5039	146.65 s	5040
pigeons 8/7	0.98 s	-	-	5039	105.88 s	0

If we refer, for example to recent benchmarks provided in ([14]), the efficiency of our prototype is nearly equivalent or slightly better. Considering the

strong implementation restrictions cited above, we thus can hope our system will provide, in addition to, and with the help of numerical constraint processing, a fairly efficient Boolean solver.

5.4 Extended numerical relations

An example of the possible use of these relations is cited both in [6] and [23] as the "magic series" problem. The purpose of this problem is to find a sequence of n non-negative integers $(x_0, \ldots, x_{n-1})$ such that for every $i \in \{0, \ldots, n-1\}, x_i$ is the number of occurences of the integer i in the sequence. In other words, for every $i \in \{0, \ldots, n-1\}$,

$$x_i = \sum_{j=0}^{n-1} (x_j = i),$$

where the value of $(x = y)$ is 1 if $(x = y)$ is true and 0 if $(x \neq y)$ is true. Moreover, it can be shown that the two following properties are true[12]:

$$\sum_{i=0}^{n-1} x_i = n, \tag{7}$$

$$\sum_{i=0}^{n-1} i\, x_i = n. \tag{8}$$

We have programmed this problem in CLP(BNR), without and with the additional constraints. Figure 3 shows the program without any additional redundant constraints and here follows the computation results for this problem:

N	Set-up	Without add. cstrt		With Eq. 7		With Eq. 7 & 8	
		back.	enum	back.	enum	back.	enum
5	1.60 s	16	17.33 s	5	0.40 s	2	0.20 s
10	6.48 s	139	17.33 s	15	1.96 s	8	1.00 s
15	14.43 s	466	112.56 s	24	8.85 s	11	3.66 s
20	25.66 s	n/a	n/a	35	34.68 s	18	11.03 s
25	40.06 s	n/a	n/a	44	108.51 s	21	24.66 s
30	57.71 s	n/a	n/a	55	281.16 s	28	51.03 s

[12] in effect computing the total number of occurences of numbers gives Equation 7, while the study of the sum of the elements of the sequence gives Equation 8

```
magic(N,L):-
  length(L,N),
  L : integer(0,_),
  constraints(L,L,0),
  enumerate(L).

constraints(L,[],N). constraints(L,[X|Xs],I):-
  sum(L,I,X),
  J is I+1,
  constraints(L,Xs,J).

sum([],I,0). sum([X|Xs],I,S):-
  S == (X==I) + S1
  sum(Xs,I,S1).
```

Figure 3: Program for the magic series problem

5.5 Scheduling with disjunctive constraints

This type of constraints can also be used to deal with scheduling problems with resource allocation. Let us consider any scheduling problem involving a set of tasks $T = \{t_1, \ldots, t_n\}$. To each task t_i is associated a pair (b_i, d_i), where b_i is the beginning and d_i is the duration of task t_i. The values for beginnings and durations are taken in a finite set of time segments[13]. To this set of tasks is associated a set of precedence and distance constraints[14]. Let us now introduce a set of resources and consider, amongst others, two tasks, $t1$ and $t2$ which share the same resource. Then the following constraint expresses that tasks t_1 and t_2 cannot be executed concurrently:

$$(b1 + d1 \leq b2) \vee (b2 + d2 \leq b1) = 1 \tag{9}$$

One way to treat these constraints, as described in [23], is to use the constraints as choice points and to find non deterministically an ordering of the tasks which satisfies the other precedence and distance constraints. In our system, we can make use of the extended numerical relations to express directly the constraint 9 as shown in the following CLP(BNR) rule:

[13] Although the set of possible segment values is finite, one is not constrained to use a finite domain representation for them, as solutions can be found using narrowing, without any enumeration step.

[14] Precedence constraints are of the type $b_i + d_i \leq b_j$, meaning that task t_i has to be completed before the beginning of task t_j. Distance constraints are any constraints which can be represented as precedence constraints by introducing new tasks. An example of distance constraint is: task t_i must begin at least n segments after task t_j has been completed.

```
disj_constraint(B1,D1,B2,D2):-
  (B1+D1 =< B2) + (B2+D2 =< B1) == 1.
```

The addition, as stated in the previous sections, is used to express the fact that the disjunction is exclusive. After having set deterministically the whole constraint system, the system can find a solution thanks to boolean enumeration. It has to be noted that as soon as either a boolean or a numerical value is known, the computation of the new stable system involves narrowing of other boolean and numerical values, possibly pruning dramatically the search tree.

We have applied this technique to the bridge problem, cited in [23]. This problem involves fourty six tasks, and more than six hundred constraints. In order to compute the solution which guarantees the minimum cost, we have implemented on top of the system a control predicate which computes a branch and bound procedure. Here are the computational results for this problem:

Bridge	Set-up time	Nb of back.	Enum. time
First solution (cost 110)	12.03 s	0	1.61 s
Best solution (cost 104)	-	4	5.28 s
Proof of optimality	-	149	28.66 s

6 Conclusion

We have shown in this paper that interval arithmetic is a good candidate for constraint solving in a CLP language including constraints on real numbers, integers and Booleans. This approach provides a unified framework in which all these different types of constraints can be freely mixed. This allows the programmer to deal with problems where the combinatorial part is coded with integer constraints and involving real coefficients, to include numerical relations in Boolean systems, and to improve the expressiveness of Boolean constraints by making use of addition, multiplication and numerical relations. The expressive power of such a language is thus an extension of the possibilities of other CLP systems, with the two important exceptions of constraints on lists (Prolog III, see [6]) and linear resolution on rational/reals numbers where Prolog III and CLP($\Re$) propose specific and complete algorithms based on Gaussian elimination and Simplex-like methods. The range of possible applications remains close to what has been already tackled with CLP methods (planning, scheduling, configuration, resource allocation, circuit design and testing, engineering-oriented KBS, etc.), while strongly tightening the links between the combinatorial and (generally non-linear) numerical aspects of the kind of problems cited above. Future work concerns implementation improvements, development of applications for "real-life problems", and design of eventual communications with complete algorithms for special cases such as rational linear programming.

Aknowledgements

We would like to express our gratitude to Peter Cashin who gave the authors the opportunity to collaborate by supporting a one year visit of one of them to the BNR Software Engineering Center. We would also thank Alain Colmerauer who has suggested a number of improvements and simplifications in the theoretical framework and Rick Workman and André Vellino for their careful reading and their comments on previous versions of this paper. We also thanks Maarten Van Emden, Henk Vandecasteele, Martin Nilson and Olivier Lhomme for interesting discussions and email correspondance on the matter presented here.

References

[1] A. Aggoun and N. Beldiceanu. "Overview of the CHIP compiler system". In *Constraint Logic Programming: Selected Research*, F. Benhamou and A. Colmerauer (eds), MIT Press, 1992 (to appear). This paper is a modified version of the paper in *Proceedings of ICLP '91*, MIT Press, p 775–789, Paris 1991.

[2] 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)*, ICOT, Tokyo, pages 263-276, December 1988.

[3] F. Benhamou, "Boolean Algorithms in Prolog III", in *Constraint Logic Programming: Selected Research*, F. Benhamou and A. Colmerauer (eds), MIT Press, 1992 (to appear).

[4] F. Benhamou and A. Colmerauer (eds), *Constraint Logic Programming: Selected Research*, MIT Press, 1992 (to appear).

[5] J. G. Cleary, "Logical Arithmetic", *Future Computing Systems*, Vol 2,No 2, p 125–149, 1987.

[6] A. Colmerauer, "An introduction to Prolog III", in *Communications of the ACM*, 33(7):69, July 1990.

[7] A. Colmerauer, "Naive Resolution of Non-linear Constraints", *Technical Report*, GIA, Marseille, France, 1992. To appear in *Constraint Logic Programming: Selected Research*, F. Benhamou and A. Colmerauer (eds), MIT Press, 1992

[8] M. Dincbas, H. Simonis and P. Van Hentenryck, "Extending equation solving and constraints handling in logic programming", in *Proc. Colloquium CREAS MCC*, Austin, Texas, May 1987.

[9] J. Jaffar and J.L. Lassez, "Constraint Logic Programming", in *Proc. POPL*, ACM, 1987.

[10] Joxan Jaffar, Spiro Michaylov, P. J. Stuckey and R. H. C. Yap, "The CLP($\Re$) Language and System", in *ACM Transactions on Programming Languages and Systems*, vol14, no 3, July 1992, Pages 339–395.

[11] J.H.M. Lee and M.H. van Emden, "Adapting CLP($\Re$) to Floating Point Arithmetic", in *Proceedings of the Fifth Generation Computer Systems Conference*, Tokyo, Japan, 1992

[12] A.K. Mackworth, "Consistency in Networks of Relations", in *Artificial Intelligence 8*, p 99-118, 1977.

[13] A.K. Mackworth, J.A. Mulder and W.S. Havens,"Hierarchical Arc Consistency: Exploiting Structured Domains in Constraint Satisfaction Problems", in *Computational Intelligence 1*, p 118-126, 1985.

[14] J.L. Massat, "Using Local Consistency Techniques to Solve Boolean Constraints", to appear in *Constraint Logic Programming: Selected Research*, F. Benhamou and A. Colmerauer (eds), MIT Press, 1992.

[15] U. Montanari, "Networks of Constraints: Fundamental Properties and Application to Picture Processing", in *Information Science*, vol 7, 1992.

[16] U. Montanari and F. Rossi,"Finite Domain Constraint Solving and Constraint Logic Programming", to appear in *Constraint Logic Programming: Selected Research*, F. Benhamou and A. Colmerauer (eds), MIT Press, 1992.

[17] R.E. Moore, "Interval Analysis". Prentice Hall, 1966.

[18] W. Older and A. Vellino, "Extending Prolog with Constraint Arithmetic on Real Intervals",in *Proceedings of the Canadian Conference on Electrical and Computer Engineering*, 1990.

[19] W. Older and A. Vellino, "Constraint Arithmetic on Real Intervals",to appear in *Constraint Logic Programming: Selected Research*, F. Benhamou and A. Colmerauer (eds), MIT Press, 1992.

[20] W. Older and F. Benhamou, *Programming in CLP(BNR)*, BNR Research report, 1993.

[21] G. Sidebottom and W. Havens, "Hierarchical Arc Consistency Applied to Numeric Processing in Constraint Logic Programming", *Technical Report CSS-IS TR 91-06*, Simon Fraser University, Burnaby, Canada, 1991. To appear in *Computational Intelligence* 8(4), Blackwell Publishers, 1992.

[22] H.M. Stark, "An Introduction to Number Theory", *MIT Press*, Cambridge, 1978.

[23] P. Van Hentenryck, "Constraint Satisfaction in Logic Programming", *MIT Press*, Cambridge, 1989.

[24] P. Van Hentenryck and Yves Deville"The Cardinality Operator: A new Logical Connective for Constraint Logic Programming", to appear in *Constraint Logic Programming: Selected Research*, F. Benhamou and A. Colmerauer (eds), MIT Press, 1992. A preliminary version is also in *Proceedings of ICLP '91*, MIT Press, p 745–759, Paris 1991.

Appendix: Composition Theorem

Theorem 3 (Composition) *Let ρ and ρ' be two F-interval convex, n-ary relations on $\Re$. If there exists at most one i in $\{1,\ldots,n\}$ such that ρ and ρ' are i-dependant then we have the two following properties:*

1. *$\rho \cap \rho'$ is F-interval convex*
2. *for every F-Block u, $\overrightarrow{\rho \cap \rho'}(u) = \overrightarrow{\rho}(\overrightarrow{\rho'}(u)) \cap \overrightarrow{\rho'}(\overrightarrow{\rho}(u))$*

We first give without proofs a certain number of technical Lemmas:

Lemma 3 *Let ρ be an n-ary relation on $\Re$, then for every i in $\{1,\ldots,n\}$,*

$$\pi_i(\underline{approx}(\rho)) = \underline{approx}(\pi_i(\rho))$$

Lemma 4 *Let ρ and ρ' be two n-ary relations on $\Re$, then for every i in $\{1,\ldots,n\}$, if ρ and ρ' are both i-independant, then $\rho \cap \rho'$ is i-independant.*

Lemma 5 *Let u and v be two F-blocks. If $u \cap v \neq \emptyset$ then*

$$\forall i \in \{1,\ldots,n\}, \pi_i(u \cap v) = \pi_i(u) \cap \pi_i(v).$$

Lemma 6 *Let ρ be an n-ary relation on $\Re$ and u a F-block, then*

$$\underline{approx}(\rho \cap u) \subset u$$

Lemma 7 *Let ρ and ρ' be two n-ary relations on $\Re$,u a F-block. Then the two following propositions are true:*

1. *$\rho' \cap \underline{approx}(\rho \cap u) \subset \underline{approx}(\rho \cap u) \cap \underline{approx}(\rho' \cap u)$*
2. *$\overrightarrow{\rho}(\overrightarrow{\rho'}(u)) \cap \overrightarrow{\rho'}(\overrightarrow{\rho}(u)) \subset \overrightarrow{\rho'}(u) \cap \overrightarrow{\rho}(u)$*

We finally establish the proof of the Composition theorem:

Proof: The proof for the case where there is no common i-dependency is a trivial specialization of the proof for the general case. We can thus consider without lack of generality that ρ and ρ' are both 1-dependant and that there exists $p, p' \in \{2,\ldots,n\}$ such that:
ρ is i-independant for all $i \in \{p+1, p'\}$,
ρ' is j-independant for all $j \in \{p'+1, n\}$.

(A) $\rho \cap \rho' \cap u = \emptyset$.

By definition of the approximation, we have:

$$\overrightarrow{\rho \cap \rho'}(u) = \emptyset$$

By Lemma 7 (2) it is sufficient to show that $\overrightarrow{\rho}(u) \cap \overrightarrow{\rho'}(u) = \emptyset$

Let us suppose that $\pi_1(\overrightarrow{\rho}(u) \cap \overrightarrow{\rho'}(u)) \neq \emptyset$, then

$\pi_1(\overrightarrow{\rho}(u)) \cap \pi_1(\overrightarrow{\rho'}(u)) \neq \emptyset$, (Lemma 5)

Let x_1 be an element of $\pi_1(\overrightarrow{\rho}(u)) \cap \pi_1(\overrightarrow{\rho'}(u))$.
Since ρ and ρ' are F-interval convex (Lemma 6),

$$\begin{array}{rcl} \exists X \in \rho \cap u & | & X = (x_1, x_2, \ldots, x_p, x_{p+1}, \ldots, x_{p'}, x_{p'+1}, \ldots, x_n) \\ \exists X' \in \rho' \cap u & | & X' = (x_1, x'_2, \ldots, x'_p, x'_{p+1}, \ldots, x'_{p'}, x'_{p'+1}, \ldots, x'_n) \end{array}$$

Let $Y = (x_1, x'_2, \ldots, x'_p, x'_{p+1}, \ldots, x'_{p'}, x_{p'+1}, \ldots, x_n)$
Since $X \in \rho$ and ρ is i-independant for all $i \in \{2, \ldots, p'\}, Y \in \rho$, (Definition 8)
Since $X' \in \rho'$ and ρ' is i-independant for all $i \in \{p', \ldots, n\}, Y \in \rho'$, (Definition 8)
Therefore:

$$\forall x_{\in} \pi_1(\overrightarrow{\rho}(u)) \cap \pi_1(\overrightarrow{\rho'}(u)) \exists Y \in (\rho' \cap \rho \cap u) \mid \pi_1(Y) = x$$

Which is in contradiction with the fact that $\rho \cap \rho' \cap u = \emptyset$ and ends the proof for this case

(B) $\rho \cap \rho' \cap u \neq \emptyset$.

Let us first remark that two F-blocks are equals iff all their projections are equals.

(I) First case: Both relations are i-dependant ($i = 1$).

Since the proof of the left-right inclusion is trivial, and applying Lemma 7 (2), it is sufficient to show that

$$\pi_i(\overrightarrow{\rho}(u) \cap \overrightarrow{\rho'}(u)) \subset \pi_i(\overrightarrow{\rho \cap \rho'}(u))$$

Since $\overrightarrow{\rho}(u) \cap \overrightarrow{\rho'}(u) \neq \emptyset$, and applying Lemma 5 we have:
$\pi_i(\overrightarrow{\rho}(u) \cap \overrightarrow{\rho'}(u)) = \pi_i(\overrightarrow{\rho}(u)) \cap \pi_i(\overrightarrow{\rho'}(u))$
Let x be an element of $\pi_i(\overrightarrow{\rho}(u)) \cap \pi_i(\overrightarrow{\rho'}(u))$, then

$$\begin{array}{rcl} \exists X \in \overrightarrow{\rho}(u) & | & X = (x, x_2, \ldots, x_p, x_{p+1}, \ldots, x_n), \\ \exists X' \in \overrightarrow{\rho}(u) & | & X' = (x, x'_2, \ldots, x'_p, x'_{p+1}, \ldots, x'_n) \end{array}$$

Since for all $j \in \{2, \ldots, p\}, \rho$ is j-independant and
for all $k \in \{p+1, \ldots, n\}, \rho'$ is k-independantand thus:

$$X'' = (x, x_2, \ldots, x_p, x'_{p+1}, \ldots, x'_n) \in \rho \cap \rho' \cap u$$

The intersection of two F-intervals being an F-interval, this ends the proof of both properties for this case.

(I) Second case: Both relations are i-independant ($i \in \{2, \ldots, p\}$)

$$\begin{aligned}\pi_i(\overrightarrow{\rho \cap \rho'}(u)) &= \pi_i(\underline{\text{approx}}(\rho \cap \rho' \cap u)), \text{ (Definition 2)}\\ &= \underline{\text{approx}}(\pi_i(\rho \cap \rho' \cap u)), \text{ (Lemma 3)}\\ &= \underline{\text{approx}}(\pi_i(u)), \text{ (Lemma 4)}\\ &= u_i.\end{aligned}$$

Applying Lemma 5, since $\rho \cap \rho' \cap u \neq \emptyset$, we have
$\pi_i(\overrightarrow{\rho}(\overrightarrow{\rho'}(u)) \cap \overrightarrow{\rho'}(\overrightarrow{\rho}(u))) = \pi_i(\overrightarrow{\rho}(\overrightarrow{\rho'}(u))) \cap \overrightarrow{\rho'}(\overrightarrow{\rho}(u))$,
We have also:

$$\begin{aligned}&\pi_i(\overrightarrow{\rho}(\overrightarrow{\rho'}(u)))\\ &= \pi_i(\underline{\text{approx}}(\rho \cap \underline{\text{approx}}(\rho' \cap u))), \text{ (Definition 2)}\\ &= \underline{\text{approx}}(\pi_i(\rho \cap \underline{\text{approx}}(\rho' \cap u))), \text{ (Lemma 3)}\\ &= \underline{\text{approx}}(\pi_i(\underline{\text{approx}}(\rho' \cap u))), (\rho \text{ is i-independant})\\ &= \underline{\text{approx}}(\underline{\text{approx}}(\pi_i(\rho' \cap u))), \text{ (Lemma 3)}\\ &= \underline{\text{approx}}(\pi_i(\rho' \cap u)), \text{ (Idempotence of approximation)}\\ &= \underline{\text{approx}}(\pi_i(u)), (\rho' \text{ is i-independant})\\ &= u_i.\end{aligned}$$

The same reasoning leads to $\pi_i(\overrightarrow{\rho'}(\overrightarrow{\rho}(u))) = u_i$, Since u_i is an F-interval, this ends the proof of both properties for this case.

(III) Third case: ρ is i-independant, ρ' is i-dependant ($i \in \{p{+}1, \ldots, p'\}$).
From Definition 2 and Lemma 3, it comes:

$$\pi_i(\overrightarrow{\rho \cap \rho'}(u)) = \underline{\text{approx}}(\pi_i(\rho \cap \rho' \cap u))$$

As shown in the previous part, we have also:

$$\pi_i(\overrightarrow{\rho}(\overrightarrow{\rho'}(u)) \cap \overrightarrow{\rho'}(\overrightarrow{\rho}(u))) = \underline{\text{approx}}(\pi_i(\rho \cap \overrightarrow{\rho'}(u))) \cap \underline{\text{approx}}(\pi_i(\rho' \cap \overrightarrow{\rho}(u))),$$

On the other hand,

$$\begin{aligned}&\underline{\text{approx}}(\pi_i(\rho \cap \overrightarrow{\rho'}(u)))\\ &= \underline{\text{approx}}(\pi_i(\underline{\text{approx}}(\rho' \cap u))), (\rho \text{ is i-independant and Prop 3})\\ &= \underline{\text{approx}}(\pi_i(\rho' \cap u)), \text{ (Lemma 3 and Idempotence of approximation)}\end{aligned}$$

Since $\underline{\text{approx}}(\pi_i(\rho' \cap \overrightarrow{\rho}(u))) \subset \underline{\text{approx}}(\pi_i(\rho' \cap u))$, (Lemma 6),

$$\pi_i(\overrightarrow{\rho}(\overrightarrow{\rho'}(u)) \cap \overrightarrow{\rho'}(\overrightarrow{\rho}(u))) = \underline{\text{approx}}(\pi_i(\rho' \cap \overrightarrow{\rho}(u)))$$

It remains to establish the following equality:

$$\underline{\text{approx}}(\pi_i(\rho \cap \rho' \cap u)) = \underline{\text{approx}}(\pi_i(\rho' \cap \overrightarrow{\rho}(u)))$$

which is true if

$$\pi_i(\rho \cap \rho' \cap u) = \pi_i(\rho' \cap \overrightarrow{\rho}(u))$$

The left-right inclusion is straightforward.
Let $x \in \pi_i(\rho' \cap \overrightarrow{\rho}(u))$. Then

$$\exists X \in \rho' \cap \overrightarrow{\rho}(u) \mid X = (x_1, x_2, \ldots, x_p, x_{p+1}, \ldots, xi-1, x, xi+1, \ldots, x_{p'}, x_{p'+1}, \ldots, x_n)$$

Since $\rho' \cap \overrightarrow{\rho}(u) \subset \overrightarrow{\rho}(u) \cap \overrightarrow{\rho'}(u)$ (Lemma 7), and applying case I, we have:

$$\exists X' \in (\rho' \cap \rho \cap u) \mid X' = (x_1, x'_2, \ldots, x'_p, x'_{p+1}, \ldots, x'i-1, x', xi'+1, \ldots, x'_{p'}, x'_{p'+1}, \ldots, x'_n)$$

Since $X' \in \rho$ and ρ is i-independant for all $i \in \{2, \ldots, p'\}$,
$Y = (x_1, x_2, \ldots, x_p, x_{p+1}, \ldots, xi-1, x, xi+1, \ldots, x_{p'}, x'_{p'+1}, \ldots, x'_n) \in \rho$
Since $X \in \rho'$ and ρ' is i-independant for all $i \in \{p', \ldots, n\}$, $Y \in \rho'$.
Therefore:

$$\forall x \in \pi_i(\rho' \cap \overrightarrow{\rho}(u)) \exists Y \in (\rho' \cap \rho \cap u) \mid \pi_i(Y) = x$$

Furthermore, since ρ' is F-interval convex, $\pi_i(\rho' \cap \overrightarrow{\rho}(u))$ is an F-interval.
A symmetric reasonning handles the case where $i \in \{p'+1, \ldots, n\}$ and concludes the proof of the theorem. □