Abstract Constraint Solving

Frédéric Benhamou* and Laurent Granvilliers**

Abstract. In this paper we propose a generic framework for constraint programming based on extensions of the notions of approximate domains and constraint narrowing operators. We propose a new local consistency notion, called weak arc consistency and prove that most local consistency notions used for finite domain and interval constraints, including box consistency, are instances of weak arc consistency. We provide a fixed point semantics for heterogeneous constraint solving and establish necessary coinditions on the solvers to guarantee correctness, termination and confluence of the solving process. This framework is designed to act as a basic tool for the design of debugging facilities for constraint logic programming languages, independantly from the constraint domains and solving techniques involved in the computations.

1 Introduction

One of the main practical achievements of Constraint Logic Programming has been to prove its ability to solve problems involving numerical and/or symbolic constraints in a way which, often enough to be mentioned, is competitive or even overruns traditional methods. In the Discipl ESPRIT project, we are interested in one of the bottlenecks of the technology: debugging aspects. In effect, debugging constraint logic programs is a very difficult task for a number of reasons. Among theses reasons (declarative aspect, complexity of the program executions, difficulties to trace the execution, lack of appropriate tools, etc.) we are especially interested here in what could be called *constraint debugging*.

By constraint debugging, we mean all programmer's activities that are linked to the detection of errors by investigating the constraint store. The difficulty of this investigation has two principal causes: the size and complexity of the store and the various techniques used in the different available systems to process the constraints. Not only these constraints can be defined over a number of domains, but the same constraints are processed, in different systems in various ways, which can be very different both in the techniques that are used and with respect to the computed results.

In effect, the core of these systems is made of algorithms, called solvers, designed to process various types of constraints. Essentially, these algorithms can

* IRIN, Université de Nantes, 2, rue de la Houssinière BP 92208 France `Frederic.Benhamou@irin.univ-nantes.fr`
** LIFO, Université d'Orléans, I.I.I.A., Rue Léonard de Vinci B.P. 6759 45067 ORLEANS Cedex 2 France `granvil@lifo.univ-orleans.fr`

be divided in two main categories: incomplete algorithms based on local consistency and propagation and complete algorithms, often combined with delay mechanisms.

Algorithms in the first category achieve approximate constraint solving by introducing various levels of local consistency checking. Local consistency was first introduced in Artificial Intelligence [31] and then adapted to the constraint programming framework in the language CHIP [1, 28] to solve discrete, combinatorial problems. The introduction of interval arithmetic in logic programming [10] extended the notion to numerical (including non-linear) constraints. These ideas were developed in the BNR-Prolog system [25] and extended to the processing of combined constraints over the Booleans, the naturals and the reals in CLP(BNR)[9].

Solvers from the second category generally divide constraint systems in two subsets. The first (e.g. linear constraints) is processed with complete algorithms while the treatment of the other is based on delay mechanisms. This is the case in CLP($\Re$) [16] and Prolog III [12].

Finally, in the most recent systems, these techniques are combined in various ways (Simplex and interval-based propagation in Prolog IV [14], atomic and global constraint processing over finite domains in CHIP [1], atomic and global continuous constraints in Declic [6], etc.) Other recent work on cooperation of heterogeneous constraint solvers can be found in [20, 15, 3, 17].

Given this state of affairs, the main problem is to design generic (that is domain and technique independant) debugging tools that can be applied to (at least almost) all constraint logic programming languages. Furthermore, to keep the debugging process at the same level of declarativity as the considered family of languages, the description of the resulting store (what is computed) should be as independant as possible from the solving algorithms.

This paper is the result of our work in this area during the last two years and is both a preliminary step in this direction and an attempt to propose a uniform framework for heterogeneous constraint solvers. Some preliminary results were reported in [5]. The main idea is to further generalize the main concepts of approximation and constraint narrowing introduced for interval constraints in [4] to model most known constraint solving algorithms, either complete or propagation-based. In this framework, solvers are represented thanks to closure operators called constraint narrowing operators. These operators are defined over subsets of the power set of the constraint domain. These subsets, called approximate domains, represent exactly the approximations performed by the algorithms. A fixed point semantics for these algorithms is then provided.

We then extend the description of the constraint solving mechanisms by defining a framework in which various solvers can collaborate, share common variables and communicate their results to each other. This framework is based on the notion of extended constraint systems. These systems are objects representing the syntactical, operational and approximation aspects of constraint processing. For any given constraint programming language, these systems form a complete lattice and we show that the ultimate goal of complex combinations of solving techniques can be expressed in terms of computing specific fixed points of various monotone operators defined over this family of lattices.

We provide necessary conditions required for the constraint solvers to guarantee termination, correctness and confluence of the central propagation algorithm. Finally, we introduce the concept of constraint rewriting operators to model symbolic transformations of constraint systems (e.g. pre-processing, delayed constraints and incremental constraint solving). To illustrate this framework, we show that complete methods for rational linear equations and inequations, delayed multiplications, propagation-based finite domains and interval techniques are instances of the proposed scheme. In the case of interval constraints, a domain of special interest for the project, we detail the various techniques which were recently developped in various constraint programming systems (e.g. CLP(BNR) [9], Newton [30, 7], Prolog IV [14], Declic [6]) and show how they can be simply expressed in the proposed framework.

The expression of properties of constraint languages and solving techniques in terms of closure operators over complete lattices has been explored by several authors [26, 14, 25, 27, 22, 15] in numerous contexts including cc and CLP. We believe that the contribution of this paper is to show that most complex heterogeneous constraint systems can be described with a family of such operators. We precise the nature and the main properties of these operators and we introduce a discrimination between operators based on approximation and on symbolic transformations. Finally, we detail how specific problems such as some forms of incrementality and delayed constraints can be described by appropriate combinations of such operators.

The paper is organized as follows: in section 2 we develop an example, in Prolog IV, of different possible ways of processing a simple constraint system. In section 3 we propose an abstract constraint solving framework based on a a generalization of constraint narrowing operators and give examples of common constraint solvers which implement such operators, in section 4 we define extended constraint systems and weak arc consistency, and give a propagation algorithm which computes weak arc consistent systems. In section 5 we examine a number of local consistency properties used in constraint logic programming languages and prove that weak arc consistency generalizes arc, interval, hull, 2B and box-consistency. In section 6 we extend these notions to model heterogeneous constraint systems, where various constraint solvers communicate and we propose general properties of constraint solvers which guarantee termination, correctness and confluence of the cooperation algorithm. In section 7, we introduce constraint rewriting operators and provide a generic algorithm for heterogeneous constraint solving. We conclude in section 8.

2 Example

In this section we provide an example, in Prolog IV, of various possible cooperations between two constraint solvers. One is a complete solver over linear equations and inequations on rational numbers (as introduced in the languages

Prolog III [12] and CLP($\mathbb{R}$)[16]). The other is an interval-based solver over arbitrary constraints over the reals as introduced in BNR-Prolog [24] and CLP(BNR) [8, 9]. The goal of these examples is to show the many possibilities for the cooperation. At first sight, the operational semantics of this cooperation (basically what is propagated to where) appear to be related to design decisions and unique for a given solver/constraint language. If this was the case, no generic debugger (or debugging framework) could be designed and ad hoc debugging facilities should be implemented in the different systems. The goal of the paper is to show that all such decisions can be abstracted in such a way that most debugging tools can be designed independently from the constraint domains and, more importantly, from the various notions and mechanisms invented to solve these constraints and to combine the solvers.
Let us assume that we want to solve the union of two constraint systems S_1 and S_2, where:

$$\begin{aligned}S_1 = \{\ & x + y + z = 0, x - y - z \leq 0, y + z \leq 0, z \geq 0\} \\ S_2 = \{\ & x \in [-10, 10], y \in [-10, 10], z \in [-10, 10], 3x - z^2 + 1 = 0, \\ & -6x^2 + y^2 + z^2 - 2 = 0\}\end{aligned}$$

The first linear system will be processed with the linear solver and the second with the interval solver. The corresponding Prolog IV program is

```
s1(X,Y,Z) :-
  X + Y + Z = 0,
  lelin(Y + Z,0),
  lelin(X - Y - Z,0),
  gelin(Z,0).

s2(X,Y,Z) :-
  cc(X,-10,10),
  cc(Y,-10,10),
  cc(Z,-10,10),
  3.*.X .-. square(Z) .+. 1 = 0,
  square(Y) .+. square(Z) .-. 6.*.square(X) .-. 2 = 0.
```

If we use the linear solver on the first system we obtain:

```
>> s1(X,Y,Z).

X = 0,
Y ~ real,
Z ~ real.
```

The Simplex-like solver detects that the only possible value for x is 0, and that no other variable is fixed. If we use the interval solver on the second system, we obtain:

```
>> s2(X,Y,Z).

X ~ cc(-'>0.33333334','>1.0000004'),
Y ~ cc(-'>2.0000004','>2.0000004'),
Z ~ cc(-'>2.0000004','>2.0000004').
```

or, with a different (and more usual presentation):

$x \in [-0.33333334^{+}, 1.0000004^{+}]$,
$y \in [-2.0000004^{+}, 2.0000004^{+}]$,
$z \in [-2.0000004^{+}, 2.0000004^{+}]$,

where a^{+} denotes the smallest float greater than a. Possible domains of values for the three variables, initially defined in the interval $[-10, 10]$ were reduced. At this point, if we had no cooperation between both solvers, the best result we could have would be:

$x = 0$,
$y \in [-2.0000004^{+}, 2.0000004^{+}]$,
$z \in [-2.0000004^{+}, 2.0000004^{+}]$,

Now, in Prolog IV the two solvers cooperate by sharing the fixed variables. The result is then:

```
>> s1(X,Y,Z), s2(X,Y,Z).

X = 0,
Y ~ cc(-1,1),
Z ~ cc(-1,1).
```

which corresponds to

$x = 0$,
$y \in [-1, 1]$,
$z \in [-1, 1]$,

One can note that the domains for variables y and z were further reduced by the interval solver with the additional information on x. Finally, one could design a cooperation scheme where more information is transmitted from the linear solver to the interval solver. In this example, it can be checked that the projection of the solution space defined by S_1 on the three coordinates constrains the solutions to be in the quarter-plane defined by $x = 0, y \leq 0, z \geq 0$. If we provide, manually, this information to the interval solver we obtain:

```
>> s1(X,Y,Z), s2(X,Y,Z), lelin(Y,0), gelin(Z,0).

Z = 1,
Y = -1,
X = 0.
```

which is the only solution of the initial constraint system. It has to be noticed that the algorithms that can be used to propagate such information from the linear to the interval solvers are quite slow and that the same result can be computed, for example with Prolog IV, thanks to enumeration-based built-in predicates. The goal of the presentation of these different examples is not to compare different methods, but to show that for the same constraints, many different approaches can be envisaged. Other possibilities include global non linear constraint processing based on box-consistency, as proposed in the Newton system [30, 29], higher levels of consistencies [18], etc.
This variety in the constraint solving processing lead us to study the possibilities of abstracting this process to generalize these, apparently very different, methods.

3 Abstract Constraint Solving

3.1 Preliminaries

Let Σ be a structure $< \mathcal{D}, \mathcal{O}, \mathcal{R} >$ where $\mathcal{D}$ is a set called *universe*, $\mathcal{O}$ is a set of function symbols and $\mathcal{R}$ a set of relation symbols. Let $V = \{v_1, \ldots, v_n\}$ be a set of variables taking their values over $\mathcal{D}$. We assume that this set is large enough to built the constraint systems described here. Terms in the constraint language are syntactic expressions built in the usual way from constants, variables, and operations. In this paper, constraints are considered to be conjunctions of atomic formulae built from Σ. Throughout the paper we always consider the natural n-ary (where n is the cardinality of V) cylindrification of all m-ary relations ($m \leq n$) denoted by constraints. This allows us to consider only relations of fixed arity.

Notations: in the rest of the paper we denote $\mathcal{P}(\mathcal{D})$ the power set of $\mathcal{D}$, X_i the ith projection of the Cartesian product X. When no confusion is possible, we also use the same notation to denote constraints and the relations they represent.

A *constraint system* is a pair $S = (C, X)$, where C is a set of constraints and X is a finite Cartesian product over $\mathcal{P}(\mathcal{D})^n$ denoting the domains of the variables from V. More precisely, for every i in $\{1, \ldots, n\}$, we have $v_i \in X_i$. Note that any constraint system C, in the usual constraint programming sense, can be represented by the pair $S = (C, \mathcal{D}^n)$.

3.2 Approximate domains

Definition 1 Approximate domain. An *approximate domain* A over a $\mathcal{D}$ is a subset of $\mathcal{P}(\mathcal{D})$, closed under (eventually infinite) intersection and such that $\mathcal{D} \in A$.

An immediate remark is that approximate domains contain $\emptyset$ as soon as they contain disjoint elements. Given a relation ρ over D and an approximate domain A over $\mathcal{D}$, its approximation w.r.t. A, denoted $\underline{\mathrm{apx}}_A(\rho)$ is defined as the intersection of all the elements of A containing ρ.

Definition 2 Approximation function. Let A be an approximate domain over a set $\mathcal{D}$. The *approximation function* over A is the function $\underline{apx}_A : \mathcal{P}(\mathcal{D}) \rightarrow \mathcal{A}$ that associates to any relation $\rho \subset \mathcal{D}$:

$$\underline{apx}_A(\rho) = \bigcap\{\mu \in A \mid \rho \subset \mu\}$$

Proposition 3. *Let A be an approximate domain over domain $\mathcal{D}$. Then*

$$\forall \mu \in A, \underline{apx}_A(\mu) = \mu$$

Proposition 4. *Let A be an approximate domain over domain $\mathcal{D}$. The the approximation function $\underline{apx}_A$ is extensive, monotone and idempotent.*

Proof. Extensibility of $\underline{apx}_A$ directly follows from the definition. Idempotence is a direct consequence of Proposition 3.

Monotonicity is proved in two steps. Let $\rho \subset \mathcal{D}$ and $\rho' \subset \mathcal{D}$ be two relations. We suppose that $\rho \subset \rho'$:

- Suppose that there exists $\mu \in A$ such that $\rho \subset \mu \subset \rho'$. Since μ contains ρ, then $\underline{apx}_A(\rho) \subset D$ by definition of $\underline{apx}_A$. Since $\rho' \subset \underline{apx}_A(\rho')$ by extensibility of $\underline{apx}_A$, it follows that $\underline{apx}_A(\rho) \subset \mu \subset \rho' \subset \underline{apx}_A(\rho')$. Then, $\underline{apx}_A(\rho) \subset \underline{apx}_A(\rho')$.
- Suppose that there does not exist $\mu \in A$ such that $\rho \subset \mu \subset \rho'$. Then, $\forall \mu \in A$ such that $\rho \subset \mu$, $\rho' \subset \mu$. Then, $\underline{apx}_A(\rho) \subset \underline{apx}_A(\rho')$ by definition of $\underline{apx}_A$.

Conversely, it can be shown that, given an extensive, monotone and idempotent function ϕ defined from $\mathcal{P}(\mathcal{D})$ to $\mathcal{P}(\mathcal{D})$, its image $A = \phi(\mathcal{P}(\mathcal{D}))$ is the set of its fixed points and constitutes an approximate domain over $\mathcal{D}$.

n-ary relations Definitions for unary relations are naturally extended to n-ary relations as follows: if ρ is an n-ary relation over $\mathcal{D}$, $\underline{\mathrm{apx}}_A(\rho)$ is the Cartesian product made of the approximations of the projections of ρ i.e.

$$\underline{apx}_A(\rho) = (\underline{apx}_A(\pi_1(\rho)), \ldots, \underline{apx}_A(\pi_n(\rho))).$$

It is easy to verify that this definition preserves the properties of extensibility, idempotence and monotonicity of approximation functions. The definition of the projection is simply:

Definition 5 kth projection of a relation. Let $\mathcal{D}$ be a set, $\rho \subset \mathcal{D}^n$ be a relation and k be an integer from $\{1, \ldots, n\}$. The k-th projection of ρ is

$$\pi_k(\rho) = \{d_k \in \mathcal{D} \mid \exists(d_1, \ldots, d_{k-1}, d_{k+1}, \ldots, d_n) \in \mathcal{D}^{n-1}) \ : (d_1, \ldots, d_n) \in \rho\}$$

Note that, given $\rho \subset \mathcal{D}^n$ and $\rho' \subset \mathcal{D}^n$ two relations, if $\rho \subset \rho'$ then for all $k \in \{1, \ldots, n\}$, $\pi_k(\rho) \subset \pi_k(\rho')$.

It can be shown that approximate domains, partially ordered by set inclusion, constitute complete lattices in which the meet operation is defined by set intersection. The join of sets is defined as the smallest set larger than all of them, i.e. the approximation of their union.

Particular approximate domains Finally, we define two classes of approximate domains which are useful in constraint processing.

Definition 6 Acceptable approximate domain. Let $\mathcal{D}$ be a set and A an approximate domain over $\mathcal{D}$.
A is called *acceptable* if and only if set inclusion is a well-founded relation over A (i.e. A does not contain any infinite decreasing sequence of elements).

In practice, the use of acceptable domains will guarantee termination for most algorithms derived from the different local consistency properties presented in this paper.

Definition 7 Unit approximate domain. Let $\mathcal{D}$ be a set and A an approximate domain over $\mathcal{D}$.
A is a *unit* approximate domain if and only if $A = \{\emptyset, \mathcal{D}, \{a_1\}, \{a_2\}, \ldots\}$, where the a_i's describe $\mathcal{D}$.

Unit approximate domains are used to represent complete solvers detecting "fixed variables", that is variables whose domain of possible values is reduced to a singleton.

3.3 Examples of Approximation domains

If we come back to the introductory examples, let us examine the different cases that were encountered:

1. Approximation of a set of linear constraints where only consistency is checked: the approximation domain is $A_1 = \{Q, \emptyset\}$, where Q is the set of rational numbers
2. Approximation of a set of linear constraints where consistency is checked and fixed variables are detected: the approximation domain is the unit approximate domain of Q, $A_2 = \{Q, \emptyset\, \{q_1\}, \{q_2\}, \ldots\}$, where the q_is describe Q,
3. Interval approximation of a set of eventually non linear constraints over the reals: the corresponding approximate domain is the set A_3 of intervals of the form[3] $[f, f']$, where f and f' are taken from a fixed subset of the rational numbers, generally the set defined by the IEEE standard for floating point numbers. This approximation domain is also the domain that is used for the global constraints over the reals,
4. Approximation of a set of linear constraints where the actual Cartesian product of the projections of the polyhedron is computed: the first idea is to use the set of intervals of the form $[q, q']$, where q and q' are rational numbers. This set is unfortunately not closed under infinite intersection. We then use the approximate domain A_3 defined above. In practice, intervals of rational numbers can be used if there is no possibility of infinite domain reductions.

[3] For the sake of simplicity we only consider here closed intervals. Open intervals or even unions of arbitrary intervals can be safely used.

5. approximations for sets of finite domain constraints and global constraints over finite domains are defined thanks to the approximate domain $A_4 = \mathcal{P}(\mathbb{N})$;
6. Approximate domains can also be defined for delayed non-linear constraints, constraints over finite and rational trees, set constraints, etc.

It is easy to show that domains A_1, A_2, A_3 and A_4 are closed under intersection.

3.4 Constraint narrowing operators

Approximate domains were introduced in the previous section to represent the approximations computed by various constraint solving algorithms with respect to the solution space defined by a given set of constraints. What remains to be studied is how the approximation is computed and this leads to the description of two different aspects. In effect, basic algorithms to compute the approximation of a given solution space are generally limited to a certain form of constraint (e.g a set of linear constraints for Simplex-like algorithm, *atomic* constraints for finite domain and interval constraints and certain precise constraint combinations for global interval or finite domain constraints. The first step is to describe the operators that compute the approximation in this limited cases, and then investigate how these approximation are combined, with the help of local consistency algorithms to compute the final approximation of the initial constraint set.

The operators we introduce to provide an abstract representation of the algorithms used to compute specific approximation are called *constraint narrowing operators.* Given an approximate domain A, and an n-ary relation ρ over $\mathcal{D}$, a *constraint narrowing operator* (thereafter abbreviated CN operator) for the relation ρ is a correct, contractant and monotone function $N : A^n \longrightarrow A^n$. More formally,

Definition 8. Let Σ be a structure and A be an approximate domain over a set $\mathcal{D}$. Let ρ be an n-ary relation over $\mathcal{D}$. The function $N : A^n \longrightarrow A^n$ is a *constraint narrowing operator* for the relation ρ iff for every $D, D' \in A^n$, the three following properties hold:

(1)	$N(D) \subset D,$	(Contractance)
(2)	$D \cap \rho \subset N(D),$	(Correctness)
(3)	$D \subset D'$ implies $N(D) \subset N(D'),$	(Monotonicity)

The solving of heterogeneous constraint systems is essentially a sequence of computations over the lattice of fixed points of such operators, as will be developed in the rest of the paper. Let us also note that most constraint narrowing operators actually used to implement solvers in CLP languages share the additional property of being idempotent, that is verify D, $N(N(D)) = N(D)$

Particular constraint narrowing operators

Optimal constraint narrowing operators Many constraint solvers implement a particular class of CN operators. As for example described in [9] for interval constraints, these operators compute the best approximation of the constraint system with respect to some approximate domain. We first prove that these operators are CN operators.

Proposition 9. *Let A be an approximate domain over a set $\mathcal{D}$, $\rho \subset \mathcal{D}^n$ be a relation and k be an integer in $\{1, \ldots, n\}$. Let $N : A^n \to A^n$ such that for all D, $N(D) = \underline{apx}_A(\rho \cap D)$. Then N is a constraint narrowing operator for ρ and is idempotent.*

Proof.
- Contractance: since $\rho \cap D \subset D$, then $N(D) = \underline{apx}_A(\rho \cap D) \subset \underline{apx}_A(D) = D$ by monotonicity of $\underline{apx}_A$.
- Correctness: $\rho \cap D \subset \underline{apx}_A(\rho \cap D) = N(D)$ by extensibility of $\underline{apx}_A$.
- Monotonicity: let $D \subset D'$ then, $\rho \cap D \subset \rho \cap D'$. Then, $N(D) = \underline{apx}_A(\rho \cap D) \subset \underline{apx}_A(\rho \cap D') = N(D')$ by monotonicity of $\underline{apx}_A$.
- Idempotence: $\rho \cap D \subset N(D)$ by correctness of N and $N(D) \subset D$ by contractance of N. Then, $\rho \cap D \subset N(D) \subset D$. By intersection with ρ, it follows $\rho \cap D \subset \rho \cap N(D) \subset \rho \cap D$. Then, $\rho \cap N(D) = \rho \cap D$. Then, $N(N(D)) = \underline{apx}_A(\rho \cap N(D)) = \underline{apx}_A(\rho \cap D) = N(D)$.

This class of operators is called here *optimal* and is defined as follows:

Definition 10. Let A be an approximate domain over a set $\mathcal{D}$. Let N be a CN operator over A. N is said to be *optimal* if and only if, for every relation ρ and every $D \in A^n$, $N(D) = \underline{\mathrm{apx}}_A(\rho \cap D)$.

Complete constraint narrowing operators A number of constraint solving algorithms are so-called *complete* in the sense that they are guaranteed to detect inconsistent input constraint systems. These algorithms implement a class of CN operators called *complete CN operators*:

Definition 11. A CN operator N over an approximate domain A ($\emptyset \in A$) for a relation ρ is *complete* iff $\forall D \in A^n, N(D) = \emptyset$ iff $\rho \cap D = \emptyset$

Perfect constraint narrowing operators Finally, we call *perfect* any CN operator which is optimal and complete. From what precedes, a CN operator defined over an approximate domain A is perfect iff it is optimal and $\emptyset \in A$.

These operators are important for constraint programming languages. For example, perfect CN operators defined over unit approximate domains are the main elements used to describe complete constraint solving algorithms for which the detection of so-called *fixed* variables is guaranteed.

3.5 Examples of Constraint narrowing operators

All solvers used in CLP systems we are aware of, including the complex solvers used in `Newton` [7, 29], can be described with CN operators. We consider in this section some examples of solvers used in constraint languages and we discuss their correspondence with CN operators. Since CN operators are defined for one constraint (as opposed to a constraint set), the key idea here is to consider constraint sets as being one constraint made of the conjunction of the initial constraints. By doing this, we show that solving algorithms on trees or the Simplex method are implementations of families of constraint narrowing operators. We first explicit the links between solvers and constraint narrowing operators.

Solvers Let Σ be a structure and A be an approximate domain over a set $\mathcal{D}$. We call solver associated to a family $\mathcal{F}_A$ of narrowing operators over A indexed by $\mathcal{P}(D^n)$ any algorithm which takes as input a pair (C, X), where C is a constraint and X is a Cartesian product of elements of A and outputs in a finite amount of time a pair (C', X') such that:

1. C and C' are equivalent (i.e. the relations they represent are equal)
2. $N_C(X) = X'$, where N_C is the CN operator indexed by $\mathcal{F}_A$ for the relation corresponding to C.

Informally, a solver associates to every n-ary relation over D a constraint narrowing operator. Since, for a given solver, there is a (different) CN operator for every relation, the solver is best represented by a mapping from $\mathcal{P}(D^n)$ onto the set of constraint narrowing operators, that is as a family of constraint narrowing operators indexed by $\mathcal{P}(D^n)$.

Constraints over rational trees The most obvious solver implemented in constraint logic programming systems is the equation and inequation solving algorithm over the set of rational trees (unification). Let $< T, \mathcal{O}, \mathcal{R} >$ be the usual structure where T is the set of rational trees [11] and where constraints are either of the form $t_1 = t_2$ or $t_1 \neq t_2$, where t_1 and t_2 are terms. The algorithm proposed initially in [11] computes, from a set of constraints S, either the empty set if S is unsatisfiable, or an equivalent set of constraint S' where all variables whose domain is reduced to a singleton are explicitly represented by appropriate constraints. Let A be the unit approximate domain over T, $A = \{\emptyset, T, \{t_1\}, \{t_2\}, ...\}$, where the t_i's describe T. Then clearly, the algorithm mentioned above implements the family $\mathcal{F}_T$ of perfect CN operators over A.

Let us note that the classic version of the unification algorithm over the Herbrand universe does not implement a family of CN operators if the occur-check is omitted (termination).

Linear constraints over the reals Linear solvers over the real numbers, as implemented in Prolog III and Prolog IV with infinite precision arithmetic can be viewed in a very similar fashion and implement the perfect constraint narrowing operator over the unit approximate domain of Q, the set of rational numbers. It is to be mentioned that the introduction of floating point arithmetic violates the correctness property.

Another interesting family of CN operators used in these systems (see also section 7) applies to delayed non-linear constraints. The suspension of these constraints can be viewed as the application of the (unique) CN operator defined over the approximate domain $A = \{Q\}$.

4 Extended constraint systems

The framework provided so far is not sufficient to model propagation methods. This is the object of this section. We will first consider the case where different CN operators are defined over the same approximate domain (e.g. interval constraints, finite domain constraints) and then generalize to heterogeneous constraint systems, that is constraint systems for which different CN operators are defined over different constraint domains. We first introduce formally extended constraint systems.

Definition 12 Extended constraint systems. Let Σ be a structure and A be an approximate domain over a set $\mathcal{D}$. An *extended constraint system* is a pair (S, X), where $S = \{(C_1, N_1), (C_2, N_2), \ldots, (C_m, N_m)\}$ is a set of pairs made of a constraint C_i and of a CN operator N_i for the relation associated to C_i, and X is an n-ary Cartesian product of elements of A. The set of these extended constraint systems is denoted $\mathcal{E}_{(\Sigma,A)}$.

For a given structure Σ and an approximate domain A over a set $\mathcal{D}$, we introduce the relation $\preceq$ over $\mathcal{E}_{(\Sigma,A)}$ defined as follows:

$$\text{For every } E, E' \in \mathcal{E}_{(\Sigma,A)},\ E = (S, X), E' = (S', X')$$

$$E \preceq E' \text{ iff } S' \subseteq S \text{ and } X \subseteq X'.$$

$\mathcal{E}_{(\Sigma,A)}$ forms a complete lattice w.r.t. $\preceq$ for which meet and join operations, respectively denoted $\sqcap$ and $\sqcup$ are defined as follows:

$$\text{For every } E, E' \in \mathcal{E}_{(\Sigma,A)},$$

$$E \sqcap E' = (S \cup S', X \cap X'), E \sqcup E' = (S \cap S', \underline{\mathrm{apx}}_A(X \cup X')).$$

4.1 Declarative and approximate semantics

The declarative semantics of an extended constraint system E, denoted E^*, is the set of n-ary tuples $(x_1, \ldots, x_n) \in X$ satisfying the conjunction of the constraints $C_1, C_2, \ldots, C_m$.

Definition 13 Declarative semantics of extended constraint systems. Let $E = (\{(C_1, N_1), (C_2, N_2), \ldots, (C_m, N_m)\}, X)$ be an extended constraint systems. The *declarative semantics* of E is defined as

$$E^* = \bigcap_{i=1}^{m} C_i \cap X$$

This set is generally either computationally intractable or even uncomputable. This leads to define the *approximate semantics* of E, denoted $\overline{E}$, as being the greatest common fixed point of the N_i's included in X. We recall that the Ni's are monotone operators defined over a complete lattice.

Definition 14 Approximate semantics of extended constraint systems.

Let $E = (\{(C_1, N_1), (C_2, N_2), \ldots, (C_m, N_m)\}, X)$ be an extended constraint system. If $fp(N_i)$ denotes the set of fixed points of N_i, then the *approximate semantics* of E is defined as

$$\overline{E} = max(\{u \in \bigcap_{i=1}^{m} fp(N_i) \mid u \subseteq X\})$$

Due to the correctness property of narrowing operators with respect to their associated constraints, we have the following, expected, result concerning the two semantics, which guarantees the correctness of the approximation:

Proposition 15. *Let E be an extended constraint system. Then, $E^* \subseteq \overline{E}$.*

We describe in the next section the local consistency notions related to the approximate semantics of ECSPs.

4.2 Weak arc-consistency

Approximate constraint solving is based on local consistency techniques, that is propagation. The local consistency we propose here is an extension of arc-consistency, called *weak arc-consistency.* This extension takes into account the approximation introduced by the CN operators.

Definition 16 Weak arc-consistency. Let $E = (\{(C_1, N_1), (C_2, N_2), \ldots, (C_m, N_m)\}, X)$ be an extended constraint system. E is *weakly arc-consistent* iff $X = \overline{E}$.

The definition for *maximal weak arc-consistency* is expressed as follows:

Definition 17 Maximal weak arc-consistency. Let $E = (S, X)$ and $F = (S, X'), X \subseteq X'$ be two extended constraint systems. Then, E is *maximally weakly arc-consistent* with respect to X' iff E is weakly arc-consistent and $X = \overline{F}$.

We will develop in the next sections the links between weak arc consistency and other local consistency notions like arc consistency [19], arc B-consistency [18], hull, interval and box consistency [7] abd show that they are instances of weak arc consistency for particular constraints, approximate domains and narrowing operators.

4.3 Filtering algorithm

Given a structure Σ and an admissible approximate domain A of its domain D, the algorithm computes, from an extended constraint system E and a Cartesian product of domains X, an equivalent extended constraint system maximally weak arc-consistent with respect to X. It is essentially an immediate adaptation of AC-3 [19] and of the filtering algorithms used in interval constraint-based systems like BNR-Prolog [25], CLP(BNR) [9] or Newton [7, 29].

function `Nar(` **inout** $E = (\{(C_1, N_1), \ldots, (C_m, N_m)\}, X)$)
begin
    `S :=` $\{C_1, \ldots, C_m\}$;
    while $S \neq \emptyset$ **and** $X \neq \emptyset$ **do**
        C `:= choose one` C_i `in` S;
        X' `:=` $N_i(X)$;
        if $X' \neq X$ **then**
            S `:=` $S \cup \{C_j \mid \exists v_k \in var(C_j) \wedge X'_k \neq X_k\}$;
            X `:=` X'
        endif
        S `:=` $S \setminus \{C\}$
    endwhile
end;

We state without proof the main properties of the algorithm:

1. if the computation of the CN operators terminates, the algorithm terminates due to the fact that A is admissible. Note that in the interval constraint case, the termination was relying on the finiteness of the approximate domain.
2. the algorithm is correct in the sense that the solution space defined by the declarative semantics of the initial system is a subset of the computed Cartesian product. In particular, if this Cartesian product is empty, the system is unsatisfiable.
3. the algorithm is confluent in the sense that the output is independent of the order in which the constraints are chosen.
4. given an extended constraint system $E = (S, X)$ the output is $E' = (S, \overline{E})$.
5. the function Nar defined by the algorithm is a contractant, monotone operator defined over $\mathcal{E}_{(\Sigma, A)}$.

Termination is due to the fact that set inclusion is a well-founded ordering for approximate domains. Correctness comes from correctness of the constraint narrowing operators.

Proof. (Confluency) The confluency is addressed by proving the existence of the greatest common fixed point of the N_i's and that it is effectively computed by the algorithm.
The operator $\cap_{i=1}^{m} N_i : A^n \to A^n$ is monotone and contractant by monotonicity and contractance of the N_i's. By double inclusion, it can be easily shown that the fixed points of $\cap_{i=1}^{m} N_i$ are exactly the common fixed points of the N_i's. The well-known Tarski's theorem says that the set of fixed points of a monotone operator over a complete lattice forms a complete lattice in itself. Then every subset of the fixed points of $\cap_{i=1}^{m} N_i$ admits a sup; in particular those included in the initial box D as input of the algorithm.
Let U be a common fixed point of the N_i's and let D' be the output of the algorithm. First, D' is a common fixed point of the N_i's which is given by the exit condition of the while loop, that is $S = \emptyset$. It remains to show that $U \subset D'$. In fact, we prove that U is included in every computed box during the algorithm. Let V be such a box containing U (it is true for the first domain D) and let N the constraint narrowing operator to be applied. By monotonicity of N, $N(U) \subset N(V)$. Then, $U \subset N(V)$ by definition of U, which ends the proof.

4.4 Decomposition of CN operators

In most cases, the application of a given CN operator to a Cartesian product of domains is achieved by the resolution of a sub-problem which is itself an extended constraint system where univariate CN operators are defined over projection constraints. To keep a certain level of generality, let us assume that, given an approximate set A over a set $\mathcal{D}$ and an n-ary relation ρ over $\mathcal{D}$, a projection constraint for ρ of rank k denote an approximation of $\pi_k(\rho)$, that is a relation μ from A, such that $\pi_k(\rho) \subset \mu$. We first define the notion of *univariate CN operator*. A univariate CN operator of rank k is a CN operator such that its application on a Cartesian product X let all its projections unchanged except for its kth projection.

Definition 18 univariate CN operator. Let A be an approximate domain over a set $\mathcal{D}$. Let C be a constraint denoting a relation $\rho \in \mathcal{D}^n$. A CN operator $N_[\rho, k]$ is an univariate CN operator of rank k for the relation ρ iff:
$\forall X \in A^n, \forall j \in \{1, \ldots, n\}, j \neq k, N[\rho, k](X) = X_i$

Then we define decomposition sets for complex CN operators.

Definition 19 Decomposition sets. Let N be a CN operator for a constraint C. A set of univariate CN operators $\{N_1, \ldots, N_n\}$ over dimensions $\{1, \ldots, n\}$ is said to be a decomposition set for N iff, for every Cartesian product X defined over A^n:

1. $\forall kin\{1, \ldots, \mathrm{n}\}, \pi_k(N(X)) \subseteq N_k(X)$
2. if E is the extended constraint system $E = (\{(C, N_1), \ldots, (C, N_m)\}, X)$, then $\overline{E} = N(X)$.

Based on the main properties of the CN operators, in this case it is not necessary to compute the application of this sort of CN operator completely before propagation. It is often better to compute only some projections and then propagate to the rest of the system. This can be done thanks to the following property:

Proposition 20. *Let $E = (\{(C_1, N_1), \ldots, (C_m, N_m)\}, X)$ be an extended constraint system. For any given $k \in \{1, \ldots, n\}$, if $\{N'_1, \ldots, N'_n\}$ is a decomposition set for N_k, let us denote E' the following extended constraint system:*
$E' = (\{(C_1, N_1), \ldots, (C_k, N'_1), \ldots, (C_k, N'_n), \ldots, (C_m, N_m)$
Then, $\overline{E} = \overline{E'}$

5 Generalizing local consistency techniques

In this second part of the document, we show that the framework presented in the first part generalize most local consistency notions used in Constraint Logic Programming languages and systems. We introduce first Arc consistency and then develop the links between weak arc consistency and the different notions introduced for interval constraints. In our knowledge, this is the first study of the various extensions used for box-consistency that proves the confluency of the algorithm.

5.1 Arc-consistency

Arc-consistency is a well-known notion in artificial intelligence [19] which states a simple local condition on a constraint c and the set of possible values for each of its variables, say $D_1, \ldots, D_n$. Informally speaking, a constraint c is arc-consistent if none of the D_k can be reduced by using projections of c.

Definition 21 Arc-consistency. Let $c(x_1, \ldots, x_n)$ be a real constraint, k be an integer in $\{1, \ldots, n\}$ and $D = (D_1, \ldots, D_n)$ represent the domains for $x_1, \ldots, x_n$. c is *arc-consistent* wrt k and D iff

$$D_k \subset \pi_k(\tilde{c} \cap D)$$

c is *arc-consistent* wrt D if for all $k \in \{1, \ldots, n\}$, c is *arc-consistent* wrt k and D. A set of constraints $\mathcal{S}$ is arc-consistent wrt D if each constraint in $\mathcal{S}$ is arc-consistent wrt D.

5.2 Hull, Interval, 2B and box-consistency

This section is devoted to the study of different local consistency notions used in Interval constraints. For self-containment purposes, we begin with some basics of interval arithmetic.

Basics of interval arithmetic

5.3 Interval arithmetic

Interval computations restore correctness of numerical computations, enclosing solutions with *intervals* being bounded compact sets of reals. Termination is obtained by taking a finite number of possible values for the bounds of intervals (in practice the floating point numbers), together with some convergence properties. Correctness is addressed providing a mechanism of *outward rounding* of computations at bounds of intervals. We recall in this section the essentials which can be found in [21, 2].

Definition 22 Interval. Let R be a finite subset of $\mathbb{R}$ and l, r be two reals such that $l \in R \cup \{-\infty\}$ and $r \in R \cup \{+\infty\}$. An *interval* $I = [l, r]$ is the compact set of reals $\{x \in \mathbb{R} \mid l \leq x \leq r\}$. In particular $[-\infty, +\infty]$ corresponds to $\mathbb{R}$, $[-\infty, r]$ represents $\{x \in \mathbb{R} \mid x \leq r\}$ and $[l, +\infty]$ denotes $\{x \in \mathbb{R} \mid l \leq x\}$. All the intervals $[l, r]$ such that $l > r$ are mapped to the empty interval $\emptyset$.
The set of all the intervals over R is denoted $\mathbb{I}(R)$.
An interval in $\mathbb{I}(R)$ is said to be *canonical* if it contains at most two elements of R (then $\emptyset$ is canonical).

Note that $\mathbb{I}(R)$ is an approximate domain with respect to the set inclusion ordering. A real x is approximated by the smallest interval containing x. More generally, a relation ρ over $\mathbb{R}$ is approximated by the smallest interval containing all the elements in ρ, using the approximation function over $\mathbb{I}(R)$. For simplifying notations, $\mathbb{I}(R)$ is denoted $\mathbb{I}$ considering the implicit existence of a set R. When presenting an example, the set R is the set of floating point numbers in double precision, in which case an element of $\mathbb{I}$ is said to be a floating point interval. The set of unions of intervals is denoted by $\mathbb{I}_\cup$.

Interval arithmetic operations are set extensions of real operations. Given $\cdot$ a real operation, two intervals I, J, and $\odot$ the interval operation corresponding to $\cdot$, $I \odot J$ is $\underline{\mathrm{apx}}_{\mathbb{I}}(\{x = y \cdot z \mid y \in I, z \in J\})$. An n-ary *interval function* is a mapping $\mathbb{I}^n \to \mathbb{I}$ involving interval constants, interval operations and interval variables. When evaluating over intervals the range of real functions, this definition remains insufficient because it does not explain how to obtain an interval function from a real one. Van Hentenryck et als. [30] introduced various forms of *interval extension* of a real function, which are defined as interval functions approximating the real one, obtained from its by a syntactic operation. In particular, two different expressions of a same real function may lead to different interval extensions. Our definition of interval extension corresponds to the *natural interval extension* of [30].

Proposition 23 Subdistributivity. *Let $I, J, K \in \mathbb{I}$. Then*

$$I \otimes (J \oplus K) \subseteq (I \otimes J) \oplus (I \otimes K).$$

Proposition 24. $\mathbb{I}$ *and* $\mathbb{I}_\cup$ *are approximate domains.*

To $\mathbb{I}$ corresponds the approximate function $\underline{apx}_{\mathbb{I}}$. Given a relation $\rho \subset \mathbb{R}$, $\underline{apx}_{\mathbb{I}}(\rho)$ is the smallest interval containing ρ and is generally called the hull of ρ. In the following, the hull of a real number r will be denoted $\overline{r}$.

Real functions and real constraints are evaluated over $\mathbb{I}$ using the concept of *interval extension*.

Definition 25 Interval extensions. $F : \mathbb{I}^n \to \mathbb{I}$ is an interval extension of $f : \mathbb{R}^n \to \mathbb{R}$ iff

$$\forall D_1, \ldots, D_n \in \mathbb{I} \ : \ d_1 \in D_1, \ldots, d_n \in D_n \Rightarrow f(d_1, \ldots, d_n) \in F(D_1, \ldots, D_n)$$

An interval relation $C \subset \mathbb{I}^n$ is an interval extension of $c \subset \mathbb{R}^n$ iff

$$\begin{gathered}\forall D_1, \ldots, D_n \in \mathbb{I} \ : \ \exists d_1 \in D_1, \ldots, \exists d_n \in D_n, \\ (d_1, \ldots, d_n) \in c \Rightarrow (D_1, \ldots, D_n) \in C\end{gathered}$$

In practice, two additive considerations for interval extensions are needed, that is monotonicity and taking of domains into account. This leads on the next definition.

Definition 26 Monotone interval extensions. Let c be a real constraint and D be a box. C is a *monotone interval extension* of $\tilde{c}$ wrt D, denoted $C[\tilde{c}, D]$, iff C is an interval extension of $\tilde{c}$ and is monotone, i.e. for all D, D' such that $D \subset D'$, $C[\tilde{c}, D] \subset C[\tilde{c}, D']$.

5.4 Interval,hull,2B,box-consistency

The key idea of local consistency techniques is that domains of variables are reduced using projection of relations.

Definition 27 Projections. We introduce the projections of real and interval relations.

1. Let $c \subset \mathbb{R}^n$, k be an integer in $\{1, \ldots, n\}$ and D be a box. The k-th projection of c wrt D is

$$\begin{aligned}\pi_k(c \cap D) = \{d_k \in \mathbb{R} \mid \exists (d_1 \in D_1, \ldots, d_{k-1} \in D_{k-1}, d_{k+1} \in D_{k+1}, \ldots, \\ d_n \in D_n) \ : \ (d_1, \ldots, d_n) \in c\}\end{aligned}$$

2. Let $C \subset \mathbb{I}^n$, k be an integer in $\{1, \ldots, n\}$ and D be a box. The k-th projection of C wrt D is

$$\pi_k(C \cap D) = \{J \in \mathbb{I} \mid (D_1, \ldots, D_{k-1}, J, D_{k+1}, \ldots, D_n) \in C\}$$

Definition 28 Hull-consistency. Let c be an n-ary real constraint, k be an integer in $\{1, \ldots, n\}$ and $D \subset \mathbb{I}^n$. c is *hull-consistent* wrt k and D iff

$$D_k \subset \underline{apx}_{\mathbb{I}}(\pi_k(\tilde{c} \cap D))$$

The definition of *interval-consistency* is obtained from the precedent by replacing $\mathbb{I}$ by $\mathbb{I}_\cup$.

Definition 29 2B-consistency. Let c be an n-ary real constraint, k be an integer in $\{1,\ldots,n\}$ and $D \subset \mathbb{I}^n$. c is *2B-consistent* wrt k and D, $D_k = [d_k, d'_k]$, iff

$$d_k \in \pi_k(\tilde{c} \cap D) \quad and \quad d'_k \in \pi_k(\tilde{c} \cap D)$$

Definition 30 Box-consistency. Let c be an n-ary real constraint, k be an integer in $\{1,\ldots,n\}$, $D \subset \mathbb{I}^n$ and $C[\tilde{c}, D]$ be a monotone interval extension of $\tilde{c}$ wrt D. c is *box-consistent* wrt k and D iff

$$D_k \subset \underline{apx}_{\mathbb{I}}(\{r \in \mathbb{R} \mid \overline{r} \in \pi_k(C[\tilde{c}, D] \cap D)\})$$

We then define an important CN operator for box-consistency:

Proposition 31. *Let $\rho \subset \mathbb{R}^n$ be a relation, k be an integer in $\{1,\ldots,n\}$, $D \subset \mathbb{I}^n$ be a box and $C[\rho, D]$ be a monotone interval extension of ρ wrt D. Let $N : \mathbb{I}^n \to \mathbb{I}^n$ such that $\pi_k(N(D)) = \underline{apx}_{\mathbb{I}}(\{r \in \mathbb{R} \mid \overline{r} \in \pi_k(C[\rho, D] \cap D)\})$ and for all $j \neq k$, $\pi_j(N(D)) = D_j$. Then N is an univariate constraint narrowing operator of rank k for ρ.*

Proof.

- Contractance: is $\underline{apx}_{\mathbb{I}}(\{r \in \mathbb{R} \mid \overline{r} \in \pi_k(C[\rho, D] \cap D)\}) \subset D_k$. Since $C[\rho, D] \cap D \subset D$ then, $\pi_k(C[\rho, D] \cap D) \subset D_k$. Let $d \in \mathbb{R}$ such that $\overline{d} \subset \pi_k(C[\rho, D] \cap D)$ then, $\overline{d} \subset D_k$ and $d \in D_k$ by extensibility of $\underline{apx}_{\mathbb{I}}$. Then, $\{r \in \mathbb{R} \mid \overline{r} \in \pi_k(C[\rho, D] \cap D)\} \subset D_k$. Then, $\underline{apx}_{\mathbb{I}}(\{r \in \mathbb{R} \mid \overline{r} \in \pi_k(C[\rho, D] \cap D)\}) \subset \underline{apx}_{\mathbb{I}}(D_k) = D_k$ by monotonicity of $\underline{apx}_{\mathbb{I}}$.
- Correctness: is $\pi_k(c \cap D) \subset \underline{apx}_{\mathbb{I}}(r \in \mathbb{R} \mid \overline{r} \in \pi_k(C[\rho, D] \cap D)\})$. Let $d \in \pi_k(c \cap D)$. Then, $\overline{d} \subset \pi_k(C[\rho, D] \cap D)$ by correction of C. Then, $d \in \{r \in \mathbb{R} \mid \overline{r} \in \pi_k(C[\rho, D] \cap D)\}$. Then, $d \in \underline{apx}_{\mathbb{I}}(\{r \in \mathbb{R} \mid \overline{r} \in \pi_k(C[\rho, D] \cap D)\})$ by extensibility of $\underline{apx}_{\mathbb{I}}$.
- Monotonicity: let $D \subset D'$ and show that $\underline{apx}_{\mathbb{I}}(\{r \in \mathbb{R} \mid \overline{r} \in \pi_k(C[\rho, D] \cap D)\}) \subset \underline{apx}_{\mathbb{I}}(\{r \in \mathbb{R} \mid \overline{r} \in \pi_k(C[\rho, D'] \cap D')\})$. It trivially follows from the monotonicity of C which states that $C[\rho, D] \subset C[\rho, D']$.

5.5 Interval extensions for box-consistency

Box-consistency strongly depends on the interval extensions chosen for the constraints and different interval extensions can produce very different (often incomparable) tradeoffs between pruning and computational complexity. In this section, we consider three extensions used in `Newton`: the natural interval extension, the distributed interval extension, and the Taylor interval extension.

Natural Interval Extension The simplest extension of a function (resp. of a constraint) is its natural interval extension. Informally speaking, it consists in replacing each number by the smallest interval enclosing it, each real variable by an interval variable, each real operation by its fixed interval extension and each real relation by its fixed interval extension. In the following, if f (resp. c) is a real function (resp. constraint), we denote by $Nat(f)$ (resp. $Nat(c)$) its natural extension.

Example 1 (Natural Interval Extension) *The natural interval extension of the function $x_1(x_2 + x_3)$ is the interval function $X_1(X_2 + X_3)$. The natural interval extension of the constraint $x_1(x_2 + x_3) = 0$ is the interval constraint $X_1(X_2 + X_3) \doteq \overline{0}$.*

Proposition 32. *Let c be a real constraint. Then $Nat(c)$ is monotone.*

Proof. The proof directly follows from monotonicity of interval extensions of real operations, considering that the combination of monotone operations is monotone.

The advantage of this extension is that it preserves the way constraints are written and hence users of the system can choose constraint representations particularly appropriate for the problem at hand. A very nice application where this extension is fundamental is the Moré-Cosnard discretization of a nonlinear integral equation described in the experimental results. Using the natural extension allows users to minimize the problem of dependency of interval arithmetic and hence to increase precision.

Distributed Interval Extension The second interval extension used by `Newton` does not preserve the way constraints are written but uses a distributed form of the constraints. The key advantage of this extension is that it allows the algorithm to enforce box-consistency by applying the interval Newton method on univariate real functions. The real functions are derived from univariate interval constraints obtained by replacing all but one variable by their intervals. As a consequence, applying box-consistency will be particularly efficient, although the pruning may be weaker than for the natural extension due to the dependency problem of interval arithmetics.[4] Intuitively, the distributed interval extension should be viewed as a way to speed up the computation of box-consistency on the natural extension. However, it may happen that it gives more precision than the natural extension if users are not careful in stating their constraints.

Definition 33 Distributed Form. A constraint c in (simplified) sum of products form

$$m_1 + \cdots + m_k = 0,$$

where each monomial m_i is of the form $qx_1^{e_1} \cdots x_n^{e_n}$ with $q \in \mathcal{Q}$ and $e_i \in \mathcal{N}$, is said to be in *distributed form*.[5]

Definition 34 Distributed Interval Extension. The *distributed interval extension* of a function f (resp. constraint c) is the natural extension of its distributed form. The distributed interval extension of a function f (resp. of a constraint c) is denoted by $Dis(f)$ (resp. $Dis(c)$).

[4] Note that it is not always necessary to go through the distributed form to obtain the above property but `Newton` adopts it for simplicity.

[5] The distributed version can easily be turned into a canonical representation for constraints.

Example 2 (Distributed Interval Extension) *The distributed interval extension of the function* $x_1(x_2+x_3)$ *is the interval function* $X_1X_2+X_1X_3$. *The distributed interval extension of the constraint* $x_1(x_2+x_3)=0$ *is the interval constraint* $X_1X_2+X_1X_3=\overline{0}$.

Proposition 35. *Let* c *be a real constraint. Then* $Dis(c)$ *is monotone.*

Proof. idem as 32.

Taylor Interval Extension The last interval extension we introduce is based on the Taylor expansion around a point. This extension is an example of centered forms which are interval extensions introduced by Moore [21] and studied by many authors, since they have important properties. The Taylor interval extension of a constraint is parametrized by the intervals for the variables in the constraint. It also assumes that the constraint which it is applied to is of the form $f=0$ where f denotes a function which has continuous partial derivatives. Given these assumptions, the key idea behind the extension is to apply a Taylor expansion of the function around the center of the box and to bound the rest of the series using the box.

Definition 36 Taylor Interval Extension. Let c be a constraint $f=0$, f be a function with continuous partial derivatives, D be a box, and m_i be the center of D_i. The *Taylor interval extension* of c wrt D, denoted by $Tayl(c,D)$, is the interval constraint

$$Nat(f)(\overline{m_1},\ldots,\overline{m_n})+\sum_{i=1}^{n} Nat(\frac{\partial f}{\partial x_i})(D_1,\ldots,D_n)\ (X_i-\overline{m_i}) \ = \ \overline{0}.$$

Proposition 37. *Let* c *be a real constraint and* $D\subset\mathbb{I}^n$ *be a box. Then* $Tayl(c,D)$ *is monotone.*

Proof. Caprani et als [23] proved that the mean-value form is inclusion isotone, which is exactly the Taylor interval extension.

We show that arc N-consistency generalizes interval-consistency [7], hull-consistency [7], box-consistency [7], and 2B-consistency [18] in the case of *interval-convex* relations (ρ is interval convex if for all i, $\pi_i(\rho\cap D)$ is a compact subset of $\mathbb{R}$).

Proposition 38. *Let* $E=(\{N_1[c_1,k_1],\ldots,N_m[c_m,k_m]\},D)$ *be an ECSP over an approximate domain* A.

1. *If* $A=\mathbb{I}$ *and if for all* i *in* $\{1,\ldots,m\}$, *and for all* $U\in A^n$, $\pi_{k_i}(N_i[c_i,k_i](U))$ *is* $\pi_{k_i}(\underline{apx}_A(c_i\cap U))$, *then* E *is arc N-consistent iff* $csp(E)$ *is hull-consistent.*
2. *If* $A=\mathbb{I}$ *and if for all* i *in* $\{1,\ldots,m\}$, *and for all* $U\in A^n$, $\pi_{k_i}(N_i[c_i,k_i](U))$ *is* $\pi_{k_i}(\underline{apx}_A(c_i\cap U))$ *and* c_i *is interval convex, then* E *is arc N-consistent iff it is 2B-consistent.*

3. *if $A = \mathbb{I}_\cup$ and the N_i's are defined as above, then E is arc N-consistent iff $csp(E)$ is interval-consistent.*
4. *If $A = \mathbb{I}$, $C[\tilde{c}_i, U]$ denotes a monotone interval extension of $\tilde{c}_i$ wrt U, and if for all i in $\{1,\ldots,m\}$,, for all $U \in A^n$, $\pi_{k_i}(N_i[c_i,k_i](U))$ is $\underline{apx}_{\mathbb{I}}(\{r \in \mathbb{R} \mid \overline{r} \in \pi_{k_i}(C[c_i,U] \cap U)\})$, then E is arc N-consistent iff $csp(\overline{E})$ is box-consistent.*

Proof. 1. First, N_j is an univariate constraint narrowing operator of rank k for c_j (see proposition 9).
Suppose that $csp(E)$ is hull-consistent then, $D \subset \underline{apx}_{\mathbb{I}}(c_j \cap D)$. Then $D_{k_j} \subset \pi_{k_j}(\underline{apx}_{\mathbb{I}}(c_j \cap D))$. $\pi_{k_j}(\underline{apx}_{\mathbb{I}}(c_j \cap D)) \subset D_{k_j}$ by contractance of N_j. Then, $D_{k_j} = \pi_{k_j}(\underline{apx}_{\mathbb{I}}(c_j \cap D)) = \pi_{k_j}(N_j)(D))$. Then, for all j, $N_j(D) = D$. It follows that D is a common fixed point of the N_j's. Obviously, it is the greatest one included in D. Then, $D = \overline{E}$.
Suppose that E is arc N-consistent then, $D = N_j(D)$ for all j. Then, $D_{k_j} = \pi_{k_j}(N_j(D)) = \pi_{k_j}(\underline{apx}_{\mathbb{I}}(c_j \cap D))$. If for all constraint $c_j(x_{j_1},\ldots,x_{j_p})$ in $csp(E)$ and for all $h \in \{1,\ldots,p\}$ it exists in E an univariate constraint narrowing operator of rank k_{jh} for c_j then, $\forall i, \forall j, D_i = \pi_i(\underline{apx}_{\mathbb{I}}(c_j \cap D))$.
2. We prove the equivalence between hull-consistency and 2B-consistency for interval convex relations.
Suppose that $csp(E)$ is hull-consistent then, $\forall i \in \{1,\ldots,n\}, \forall j \in \{1,\ldots,m\}, D_i \subset \pi_i(\underline{apx}_{\mathbb{I}}(c_j \cap D))$. Since $\pi_i(\underline{apx}_{\mathbb{I}}(c_j \cap D)) = \underline{apx}_{\mathbb{I}}(\pi_i(c_j \cap D))$ then, $D_i \subset \underline{apx}_{\mathbb{I}}(\pi_i(c_j \cap D))$. Note $D_i = [d_i, d'_i]$ and suppose that $d_i \notin \pi_i(c_j \cap D)$. Then, $d_i \notin \bigcap\{Y \in A \mid \pi_i(c_j \cap D) \subset Y\} = \underline{apx}_{\mathbb{I}}(\pi_i(c_j \cap D))$ which is a contradiction. Then, $d_i \in \pi_i(c_j \cap D)$. The proof of $d'_i \in \pi_i(c_j \cap D)$ is similar.
Suppose that $csp(E)$ is 2B-consistent then, $\forall i \in \{1,\ldots,n\}, \forall j \in \{1,\ldots,m\}, D_i = [d_i, d'_i], d_i \in \pi_i(c_j \cap D)$ and $d'_i \in \pi_i(c_j \cap D)$. Since c_j is interval-convex then, $\forall d \in D_i$, $d \in \pi_i(c_j \cap D)$. Then, $D_i \subset \pi_i(c_j \cap D)$. Then, $D_i = \underline{apx}_{\mathbb{I}}(D_i) \subset \underline{apx}_{\mathbb{I}}(\pi_i(c_j \cap D))$. Since $\underline{apx}_{\mathbb{I}}(\pi_i(c_j \cap D)) = \pi_i(\underline{apx}_{\mathbb{I}}(c_j \cap D))$ then, E is hull-consistent.
3. The proof is analogous of item (i) by taking $\mathbb{I}_\cup$ as approximate domain.
4. The proof is analogous of item (i). The proposition 31 ensures that N is a constraint narrowing operator.

6 Heterogeneous Constraint Solving

Having represented various constraint solvers with the same object, combining them is easier. The main difficulty comes from the fact that they may be defined over different approximate domains. To simplify the presentation, we impose the restriction that constraints are defined over the same structure but processed with different algorithms using different approximate domains, as it is the case when several numerical solvers cooperate, for example. This can be generalized to the case where different structures are involved.

Let Σ be a structure of domain D. Let A and A' be two approximate domains of D. Let $E = (\{(C_1, N_1), \ldots, ((C_m, N_m)\}, X)$ be an extended constraint system over A and $E' = (\{(C'_1, N'_1), \ldots, ((C'_p, N'_p)\}, X')$ be an extended constraint system over A'. Let us suppose that $A \cup A'$ is closed under intersection. then we define operators $M_1, \ldots, M_m, M_{m+1}, \ldots, M_{m+p}$ over $A \cup A'$ for constraints $C_1, \ldots, C_m, C'_1, \ldots, C'_p$ as follows.

For every $i \in \{1, \ldots, m\}$, every $j \in \{1, \ldots, p\}$ and every n-ary Cartesian product u over $(A \cup A')^n$:

$M_i(u) = N_i(\underline{\mathrm{apx}}_A(u)) \cap u.$

$M_{j+m}(u) = N'_j(\underline{\mathrm{apx}}_{A'}(u)) \cap u.$

One can show that the M_i's ($i \in \{1, \ldots, m+p\}$) are correct, monotone contracting operators over $(A \cup A')^n$ and therefore CN operators. We conclude that, under these conditions, the system
$E'' = (\{(C_1, M_1), \ldots, (C_m, M_m), (C'_1, M_{m+1}), \ldots, (C'_m, M_{m+p})\}, X \cap X')$ is therefore an extended constraint system over $(A \cup A')^n$. Finally, we have the following properties:

Proposition 39.
Let Σ be a structure of domain D. Let A and A' be two approximate domains of D. Let $E = (\{(C_1, N_1), \ldots, ((C_m, N_m)\}, X)$ be an extended constraint system over A and $E' = (\{(C'_1, N'_1), \ldots, ((C'_p, N'_p)\}, X')$ be an extended constraint system over A'. If $A \cup A'$ is closed under intersection, then:

1. *$E'' = (\{(C_1, M_1), \ldots, (C_m, M_m), (C'_1, M_{m+1}), \ldots, (C'_m, M_{m+p})\}, X \cap X')$ is an extended constraint system over $(A \cup A')^n$,*
2. *$E''^* = E^* \cap E'^*$,*
3. *$\overline{E''} \subseteq \overline{E} \cap \overline{E'}$*

From this property, it is easy to design an algorithm for heterogeneous constraint solvers such as those presented in the examples above. To give a concrete example, let us assume that we have a CLP in which a infinite precision Simplex-like algorithm and a floating point interval-based propagation algorithm cooperate. If a variable is fixed by the Simplex to a value which is not representable in the floating point representation used, the processing will consist in approximating the infinite precision rational with a floating point interval, apply the propagation algorithm (possibly modifying the domains of possible values for other variables) and then intersect back the floating point interval representing the set of possible values for this variable with the initial rational number, preserving monotonicity and computing the best approximation.

Unfortunately, this algorithm does not take into account two additional notions frequently encountered in CLP systems, namely delayed constraints and incremental constraint solving. This is the object of the following section.

7 Constraint rewriting operators

Delayed constraints To take an example of delayed constraints, let us focus on delayed multiplications as performed in CLP($\Re$) or Prolog III. The basic principle is as follows. Given a set of constraints $S = S_1 \cup S_2$, where $S_1 = \{c_1, \ldots, c_m\}$ is made of linear equalities and inequalities and $S_2 = \{c'_1, \ldots, c'_p\}$ is made of constraints of the form $x = yz$, apply the following algorithm:

Step 1: check the satisfiability of the set S_1. If S_1 is unsatisfiable return *false* else for every variable x occurring in S_1 such that any constraint of the type $x = a, a \in Q$ is entailed by S_1 add the (redundant) constraint $x = a$ to S_1. Let us call all variables occurring in a constraint of the form $x = a, a \in Q$ *fixed variables.*

Step 2: for every constraint $c'_i = (x = yz), i \in \{1, \ldots, p\}$ from S_2 such that either y or z (or both) are fixed variables, add c'_i, in which the appropriate variables are replaced with their values, to S_1.

Step 3: if S_1 was unchanged by step 2 then return $S_1 \cup S_2$ else goto step 1.

This algorithm is described in details in [13, 16]. We are interested here in expressing it as an instance of our framework. Representing:

- S_1 by the pair $(c_1 \wedge \ldots \wedge c_m, N)$, where N is the CN operator corresponding to the application of a Simplex-like algorithm detecting fixed variables (like the one described in the previous examples),
- S_2 by the pairs $(c'_1, N'), \ldots, (c'_p, N')$ where N' is the narrowing operator over $\{Q\}$ (see also previous examples),

and applying the filtering algorithm (note that both N and N' are defined over the same domain) to the extended constraint system $(c_1 \wedge \ldots \wedge c_m, N), (c'_1, N'), \ldots, (c'_p, N')$ corresponds exactly to the application of step 1. No further filtering can be done. Therefore the usual algorithm cannot be described with the notions introduced previously.

The main problem, which also concerns many other constraint solving techniques comes from the fact that the solving process is actually made of the iteration of two different processes, namely filtering and *symbolic system rewriting.*

Dynamic constraint solving Another similar problem comes from incrementality. As it is well-known, running a program written in a constraint logic programming language generates sequences of constraint sets to be solved incrementally.

Generalizing the systems to allow infinite Cartesian products, a simple way to deal with incrementality is as follows. Let E be an extended constraint system called *store* and E' be an extended constraint system. The goal is to add the system E'. The resulting store can then be expressed as $\mathrm{Nar}(E \sqcap E')$.

Unfortunately various incremental combinations of solvers (e.g. Prolog IV) cannot be represented in this manner. For example, assume a combination of linear constraint solving using Simplex-like methods and non-linear constraint solving processed with interval constraints. Given two extended constraint systems

$E = (\{(C_l, N_l), (C_1, N_1), \ldots, (C_m, N_m)\}, X)$ and

$E' = (\{(C'_l, N'_l), (C'_1, N'_1), \ldots, (C'_p, N'_p)\}, X)$, where

1. C_l and C'_l are sets of linear inequalities over the reals,
2. C_i's and C'_i's are primitive numerical constraints,
3. N_l and N'_l represent applications of Simplex to C_l and C'_l,
4. the N_i's and N'_i's are the perfect CN operators over the set of floating point intervals for relations C_i and C'_i

Then, $E \sqcap E'$ is of the form:
$(\{(C_l, N_l), (C'_l, N'_l), (C_1, N_1), \ldots, (C_m, N_m), (C'_1, N'_1), \ldots, (C'_p, N'_p)\}, X \cap X')$

Computing this set and applying the filtering algorithm is a possible combination of Simplex-like methods and interval constraint techniques but it is not the only one. One may want to compute the set of linear constraints $C_l \cup C'_l$ instead of processing them separately and having only the fixed variables communicating their domain modifications. The extended constraint system computed in this last case should be the extended constraint system:

$$E'' = \{(C_l \cup C'_l, N''_l), (C_1, N_1), \ldots, (C_m, N_m), (C'_1, N'_1), \ldots, (C'_p, N'_p)\}, X \cap X')$$

where N''_l represent the application of Simplex to $C_l \cup C'_l$. Again, we encounter the need for a rewriting step applied before filtering. To address this problem, we extend our framework by introducing *constraint rewriting operators.*

Definition 40 Constraint rewriting operators. Let Σ be a structure and A be an approximate domain over its domain D.
The mapping Rew, $\mathcal{E}_{(\Sigma,A)} \longrightarrow \mathcal{E}_{(\Sigma,A)}$ such that for every extended constraint system $E = (\{(C_1, N_1), (C_2, N_2), \ldots, (C_m, N_m)\}, X)$,
$\mathrm{Rew}(E) = (\{(C'_1, N'_1), (C'_2, N'_2), \ldots, (C'_p, N'_p)\}, X)$ is a constraint rewriting operator iff

1. Rew is contractant, monotone and idempotent and
2. $E^* = \mathrm{Rew}(E)^*$.

Operationally, the main idea for solving extended constraint systems when both filtering (propagation) and rewriting are involved is as follows. Given a structure of domain D, an extended constraint system E over an approximate domain A over D and a constraint rewriting operator Rew , apply iteratively Rew and Nar to the system until a stable state is reached. This algorithm has the following properties:

Proposition 41. *The process above terminates if A is admissible and computes the greatest fixed point E' of $(Nar \circ Rew)$ verifying $E' \preceq E$.*

Coming back to the delayed multiplication example, let us define the constraint rewriting operator Rew as the output of the following algorithm:

input A system $E = (E_l \cup E_n, X)$, defined over the unit approximate domain of the set of rational numbers, where

1. $E_l = \{C, N\}$ and C is a conjunction of linear inequalities over the reals,
2. $E_n = \{(C_1, N'), (C_2, N'), \ldots, (C_m, N')\}$ and $\{C_1, \ldots, C_m\}$ is a set of delayed non-linear equations of the form $(x_i = x_j \times x_k)$.
3. N, N' are defined as already discussed above,

step 1: $C' \longleftarrow C$

step 2: for every constraint $(x_i = x_j \times x_k \in \{C_1, \ldots, C_m\}, i, j, k \in \{1, \ldots, n\}$, if $X_j = a, a \in Q$ then

$C' \longleftarrow C' \wedge x_i = ax_k$

else if $X_k = a, a \in Q$ then

$C' \longleftarrow C' \wedge x_i = ax_j$

output $(E_l \cup E_n \cup \{C', N''\}, X)$, where N'' is the constraint narrowing operator corresponding to the application of Simplex to C'.

Clearly, the operator defined by this algorithm verifies the necessary properties to be a constraint rewriting operator. Then by applying repeatedly this algorithm and the filtering algorithm to the initial system, one eventually reaches a stable point corresponding exactly to what would be computed by Prolog IV, for example. Let us terminate with a few comments:

1. Adding a step to the rewriting algorithm would allow to implement easily the incrementality behaviour discussed previously,
2. In the actual delayed multiplication algorithm the non-linear constraints are *replaced* with a linear constraint when necessary, while in our algorithm we keep the non-linear constraint and the previous linear system. This is required to verify the contractance property of constraint rewriting operators. Of course, in practice, these redundant systems are not computed.
3. Different communications between the solvers can be easily described. In this example, the modification of values for the Simplex would only be transmitted, say, to the interval-based algorithm, when a variable becomes fixed. If the expected behaviour is to compute and transmit bound modifications for example, the corresponding family of narrowing operators must be defined over the approximate domain $\{\emptyset, Q, \{q_1\}, \ldots, f_1, \ldots\}$, where the f_i's are floating point intervals.

4. Proving termination, correctness and confluence of complex cooperating algorithms is reduced to showing that the involved solvers are constraint narrowing operators, that the symbolic rewriting achieved by the solver defines a constraint rewriting operator and that the different approximate domains involved in the process are admissible and that their union is closed under intersection.

8 Conclusion

We have designed in this paper a framework describing the operational and fixed point semantics of an abstract constraint solver generalizing most standalone and cooperating constraint solvers encountered in Constraint Logic Programming languages. We have shown that the different solvers either computing exact or approximate solutions can be represented by extending the "domain" approach, initially designed for discrete constraints and previously extended to the interval constraint paradigm. This has led us to extend the notion of arc-consistency to heterogeneous constraint networks.

The framework described here is general enough to describe systems in which different sets of constraints, eventually over different domains are processed by different algorithms. We have proposed a general propagation algorithm and we have given conditions on solving algorithms which guarantee correctness, termination and confluence of the algorithm. The output of the algorithm has been characterized, giving a fixed point semantics to the cooperation. Finally, we have shown on several examples that various instances of solver cooperation in actual CLP systems can be described within our framework.

Acknowledgements We are indebted to John Lloyd for numerous discussions on this subject, especially during his recent visit to the University of Orléans. An important part of this work was financed by the LTR ESPRIT project 22532 Discipl.

References

1. A. Aggoun and N. Beldiceanu. Overview of the CHIP Compiler System. In F. Benhamou and A. Colmerauer, editors, *Constraint Logic Programming: Selected Research*, pages 421–435. MIT Press, 1993.
2. G. Alefeld and J. Herzberger. *Introduction to Interval Computations*. Academic Press, 1983.
3. F. Baader and K. U. Schulz. On the combination of symbolic constraints, solution domains, and constraint solvers. In *Proceedings of the First International Conference on Principles and Practice of Constraint Programming, Cassis (France)*, volume 976 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Sept. 1995.
4. F. Benhamou. Interval Constraint Logic Programming. In A. Podelski, editor, *Constraint Programming: Basics and Trends*, volume 910 of *LNCS*, pages 1–21. Springer-Verlag, 1995.

5. F. Benhamou. Heterogeneous Constraint Solving. In *Proceedings of the fifth International Conference on Algebraic and Logic Programming (ALP'96)*, volume 1139 of *LNCS*, pages 62–76, Aachen, Germany, 1996. Springer-Verlag.
6. F. Benhamou, F. Goualard, and L. Granvilliers. Programming with the declic language. Technical report, University of Orléans, 1997.
7. F. Benhamou, D. McAllester, and P. Van Hentenryck. CLP(Intervals) Revisited. In *Proceedings of ILPS'94*, pages 1–21, Ithaca, NY, USA, 1994. MIT Press.
8. F. Benhamou and W. J. Older. Applying Interval Arithmetic to Real, Integer and Boolean Constraints. Technical report, Bell Northern Research, 1992.
9. F. Benhamou and W. J. Older. Applying Interval Arithmetic to Real, Integer and Boolean Constraints. *Journal of Logic Programming*, 32(1):1–24, 1997.
10. J. G. Cleary. Logical Arithmetic. *Future Computing Systems*, 2(2):125–149, 1987.
11. A. Colmerauer. Equations and inequations on finite and infinite trees. In *FGCS-84: Proceedings International Conference on Fifth Generation Computer Systems*, pages 85–99, Tokyo, 1984. ICOT.
12. A. Colmerauer. An Introduction to Prolog III. *Communications of the ACM*, 33(7):69–90, July 1990.
13. A. Colmerauer. Naive solving of non-linear constraints. In F. Benhamou and A. Colmerauer, editors, *Constraint Logic Programming: Selected Research*, pages 89–112. MIT Press, 1993.
14. A. Colmerauer. Specifications of Prolog IV. Draft, 1996.
15. H. Hong. Confluency of Cooperative Constraint Solving. Technical Report 94-08, RISC-Linz, Johannes Kepler University, Linz, Austria, 1994.
16. J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. The CLP($\Re$) language and system. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, July 1992.
17. H. Kirchner and C. Ringeissen. Combining symbolic constraint solvers on algebraic domains. *Journal of Symbolic Computation*, 18(2):113–155, 1994.
18. O. Lhomme. Consistency techniques for numeric CSPs. In R. Bajcsy, editor, *Proceedings of the 13th IJCAI*, pages 232–238, Chambéry, France, 1993. IEEE Computer Society Press.
19. A. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1):99–118, 1977.
20. P. Marti and M. Rueher. A Distributed Cooperating Constraints Solving System. *International Journal on Artificial Intelligence Tools*, 4(1):93–113, 1995.
21. R. E. Moore. *Interval Analysis.* Prentice-Hall, Englewood Cliffs, NJ, 1966.
22. A. Narin'yani. Subdefinite models and operations with subdefinite values. *Preprint, USSR acad. of sciences, Siberian Division*, 400, 1982. (in Russian).
23. A. Neumaier. *Interval Methods for Systems of Equations.* Cambridge University Press, 1990.
24. W. Older and A. Vellino. Extending Prolog with Constraint Arithmetic on Real Intervals.
25. W. Older and A. Vellino. Constraint Arithmetic on Real Intervals. In F. Benhamou and A. Colmerauer, editors, *Constraint Logic Programming: Selected Research.* MIT Press, 1993.
26. V. Saraswat, M. Rinard, and P. Panangaden. Semantic foundations of concurrent constraint programming. In *POPL'91: Proceedings 18th ACM Symposium on Principles of Programming Languages*, pages 333–352, 1991.

27. M. van Emden. Value constraints in the CLP scheme. In F. Benhamou, W. Older, M. van Emden, and P. van Hentenryck, editors, *ILPS'95 post-conference workshop on Interval Constraints*, Portland, Oregon, USA, 1995.
28. P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. Logic Programming Series. MIT Press, Cambridge, MA, 1989.
29. P. Van Hentenryck, D. McAllester, and D. Kapur. Solving Polynomial Systems Using a Branch and Prune Approach. *SIAM Journal on Numerical Analysis*, 1996. (Forthcoming).
30. P. Van Hentenryck, L. Michel, and F. Benhamou. `Newton`: Constraint Programming over Non-linear Constraints. *Scientific Programming*, 1997. (Forthcoming).
31. D. L. Waltz. Generating Semantic Descriptions from Drawings of Scenes with Shadows. In P. H. Winston, editor, *The Psychology of Computer Vision*. McGraw Hill, 1975.