Archives of Control Sciences
Volume 16(LII), 2006
No. 2, *pages 161–190*

An introduction to interval-based constraint processing

GERRIT RENKER and HATEM AHRIZ

Constraint programming is often associated with solving problems over finite domains. Many applications in engineering, CAD and design, however, require solving problems over continuous (real-valued) domains. While simple constraint solvers can solve linear constraints with the inaccuracy of floating-point arithmetic, methods based on interval arithmetic allow exact (interval) solutions over a much wider range of problems. Applications of interval-based programming extend the range of solvable problems from non-linear polynomials up to those involving ordinary differential equations.

In this text, we give an introduction to current approaches, methods and implementations of interval-based constraint programming and solving. Special care is taken to provide a uniform and consistent notation, since the literature in this field employs many seemingly different, but yet conceptually related, notations and terminology.

Key words: constraint programming, interval-based computation, interval consistency techniques

1. Motivation and scope of this paper

This text grew out of the need to communicate concepts from constraint satisfaction to people acquainted with linear and non-linear optimization techniques and conversely, to view concepts used in global optimisation from a constraint satisfaction standpoint. The objective of this text is to present the foundations of numeric constraint satisfaction with an emphasis on interval-based consistency techniques in order to permit both perspectives. This survey is short and necessarily biased towards the objective, but we do point to additional literature which allows further exploration, advanced aspects and other perspectives. It can often be observed that the presentation of original ideas is parallelled by the introduction of original notation, which impairs comparative presentation and can lead to confusion. Therefore special care has been taken to provide a uniform and consistent notation. Conceptually, we present elements from three different disciplines: global optimisation, interval-based computation and constraint programming.

Optimisation is the general term for describing problems over continuous domains in which the objective is to optimise an objective subject to given constraints. *Local optimisation* techniques are able to efficiently obtain some local optimum for often large-scale

The Authors are with School of Computing, The Robert Gordon University, Aberdeen, UK. E-mail: ha@comp.rgu.ac.uk

Thanks are due to the reviewers who helped a lot to improve this document.

problems, but do not necessarily find the globally optimal solution. *Global optimisation*, on the other hand, deals with finding the absolutely best set of admissible conditions, where both the objective and constraints are formulated in mathematical terms. It thus subsumes the ideas of linear programming, non-linear programming, mixed integer (non-) linear programming and related disciplines. For a comprehensive and detailed survey on global optimisation, see [46].

Interval-based computation is gaining momentum as a paradigm for computing with real numbers. Its strengths are (i) rigour – guarantees for results can be given if a 'safe' representation mode is used, (ii) transparency – no loss of information due to the existence and accumulation of rounding errors and (iii) technical feasibility – in contrast to infinite-precision arithmetic, the space demands of the representation (floating-point numbers to represent bounds) are very modest. Fast implementations of interval arithmetic now exist (see section 4) and even come as a standard feature in some modern compilers [51]. The benefits of this technology can thus be reaped without having to spend time on low-level issues.

Constraint programming is, like global optimisation, a general term and it describes a computational approach to solving combinatorial problems which has greatly matured during the past three decades. One of its key strengths is the use of domain-specific inference techniques known as constraint propagation, which exploit information inherent in the problem to reduce a potentially vast search space. This approach is generic and can in principle be instantiated on any domain. Propagation can fruitfully be viewed either as a set of functions that are applied until reaching a common fixed-point (after which no further inferences are triggered) or equivalently as rules that rewrite a system of constraints until it is closed under rule application. A detailed and richly illustrated treatment of these concepts can be found in [2]. Constraint processing has traditionally been more strongly associated with finite domains, whose representation on a computer is less of an issue. There is however no reason to restrict constraint technology to these domains. In fact, professional constraint solvers like Ilog Solver, Prolog IV or ECLiPSe already come equipped with interval constraint propagation facilities. Furthermore, due to the generic nature of the propagation process, the underlying procedures are not constrained to be interval-based. This allows fruitful complementation of and combination with the large corpus of techniques that are already available in (non-) linear programming, global optimisation and numerical analysis. Techniques that have been developed in the constraints community, e.g. redundant modelling or adding redundant constraints to intensify propagation activity [53], can likewise be reused in the real-valued context.

This text is structured as follows. After a brief survey of the main concepts used in constraint-based problem solving in section 2, we describe interval-consistency approaches in section 3. This is followed by a presentation of selected available implementations in section 4. We conclude with bibliographical remarks in section 5.

2. Constraint-based problem solving of numerical problems

2.1. General approach

A *constraint satisfaction problem* (CSP) is traditionally denoted as a triple $\langle X, \mathcal{D}, \mathcal{C} \rangle$ of variables X, set of domains $\mathcal{D}$ and constraints $\mathcal{C}$. Each variable $x \in X$ can only take on values from its domain $D_x \in \mathcal{D}$. A constraint $c \in \mathcal{C}$ is a relation associated with a subset $S \subseteq X$ of variables called *scope*, such that the Cartesian product of all variable domains for S forms a superset of c. The solution set $\mathsf{Sol}(P)$ of a CSP P is a subset of the Cartesian product of all domains in $\mathcal{D}$ in such a way that the projection of $\mathsf{Sol}(P)$ onto the scope of each constraint $c \in \mathcal{C}$ yields a subset of c, thereby 'satisfying' this constraint. A *constrained optimisation problem* (CSOP) is a CSP in combination with a function which maps each element of $\mathsf{Sol}(P)$ into a numerical value which is to be optimised as the objective of the problem.

These problems can be solved by a variety of techniques, the choice of which depends on the problem instance format. If all domains in $\mathcal{D}$ are finite, one of the many available finite-domain constraint solvers can be used. Problems that involve only linear constraints (possibly with integrality conditions) can be solved with (Mixed Integer) Linear Programming techniques such as the Simplex solver [12], alternatively a floating-point solver (e.g. the CLP($\mathbb{R}$) language [26, 38]) can be chosen. Since in principle any algorithm capable of solving a constraint problem could be considered as a constraint solver, we specify as defining characteristic of constraint processing the interleaved combination of constraint propagation with (exhaustive) search.

Example 2.1 (Propagation) Consider the integer CSP $\langle \{x, y\}, D_x = D_y = [1,5], x+1 < y \rangle$. There is no support for $y = 1$, $y = 2$ in D_x and so D_x can be pruned to $[3,5]$. Conversely, the propagation of the changed domain D_x via the constraint reduces the domain D_y to $[1,3]$.

This is an example of establishing arc-consistency (section 3.3.1), more sophisticated propagation techniques (also called local consistency techniques) do exist [2]. Typically the effects of propagation on one variable domain trigger propagations on another variable domain until a stable (fixed-point) state is achieved.

2.2. Solving non-linear numerical problems

A problem common to solvers based on floating-point arithmetic (hence discussed further below) is that the presence and accumulation of rounding errors introduce a source of inaccuracy and possibly even incompleteness, as solutions may be missed. In [47], a problem is reported whose solution was missed by six out of seven state-of-the-art MILP solvers (among these CPLEX 8.00) due to rounding errors. Solving (non-) linear numerical constraint problems as surveyed in this text can be classified into three strategies:

1. incomplete, local or naive strategies

2. complete and standalone solvers
3. constraint processing based on interval arithmetic

2.3. Incomplete, local or naive strategies

Local methods to solving non-linear optimisation problems are in some cases attractive due to their computational efficiency. This comes at the expense of not being able to (i) prove existence and uniqueness of solutions, (ii) isolate global optima in general, and (iii) compute the entire solution set for a system of constraints [29, sec. 1.2]. For a survey of nonlinear local optimisation see [10, chap. 2].

Naive strategies denote constraint solvers that can handle non-linear constraints without actually solving them. The simple idea underlying these approaches is that if a constraint is currently in a non-linear form (e.g. multiplication of two uninstantiated variables), its evaluation is suspended until results of the ongoing solving process permit rewriting it into a linear constraint (e.g. if one of the variables becomes instantiated). This

```
cmult([R1, I1], [R2, I2], [R, I]) :-
     R = R1 * R2 - I1 * I2,
     I = R1 * I2 + I1 * R2.
```

Figure 1. Multiplication of complex numbers in CLP($\mathbb{R}$)

approach is realized in the CLP($\mathbb{R}$) language [26, 38]. CLP($\mathbb{R}$) does not have a solver for non-linear constraints, nor does it provide special interval methods. Non-linear constraints are simply delayed until sufficiently many variables become ground in order to allow an evaluation as a linear constraint [38]. As an example, consider the complex number multiplication program of figure 1. In the following goal

```
?- cmult([R1,2], [R2,4], [-5,10]).

R1 = -0.5 * R2 + 2.5
3 = R1 * R2
```

the system answers `Maybe`, since the nonlinear constraint `R1 * R2 = 3` could not be resolved. A similar principle underlies the naive solving algorithm discussed in [16], which is integrated into Prolog III [15]. The algorithm maintains two sets of constraints, one for linear constraints and one for non-linear constraints. Elementary steps of the algorithm consist of (a) solving the set of linear constraints and (b) using the computed results to instantiate corresponding variables in the set of non-linear constraints. If as a consequence of (b) non-linear constraints can be reduced to a linear format, these results are transferred to the set of linear constraints, whereupon the algorithm enters its next iteration. This process continues until either no further change is possible or an inconsistency is detected.

The disadvantage of the approaches in this class is that the inherent incompleteness leads to either missing out on solutions or non-termination, as demonstrated for CLP($\mathbb{R}$) in [35].

2.4. Complete and standalone solvers

Research in numerical analysis and global optimisation has resulted in techniques which make it possible to obtain all solutions of a given non-linear problem. A discussion of this body of work is beyond scope of this text, for a comprehensive survey see [46]. These techniques are none the less important from a constraint-oriented viewpoint, since ideas developed in these areas (e.g. automatic differentiation [10, chap. 3] or the use of special forms for interval-based consistency methods [29]) have influenced constraint processing techniques. For the further development of constraint-based solvers it is therefore interesting and profitable to incorporate and exploit knowledge from these areas into the solving process. Since we restrict ourselves to presenting constraint-based approaches, we look at two symbolic computer algebra techniques that have gained significance in constraint solving, the computation of Gröbner bases and the cylindrical algebraic decomposition method.

The *Gröbner basis* of a given set of polynomial equations is essentially a simplified set of equations enabling the test for satisfiability. Gröbner bases are canonical finite sets of multivariate polynomials which define the same algebraic structure as the initial polynomial system. The Buchberger algorithm can be used to compute Gröbner bases and to solve a system of non-linear polynomial equations, it operates on complex numbers and may be viewed as a generalization of Gaussian elimination [35, p. 141]. The variant of the Buchberger algorithm used in the `CAL`[1] system [50] performs these computations using rewrite rules and is also capable of handling polynomial inequations. The close proximity of the Buchberger algorithm and a standard completion procedure for rewrite systems is emphasized in [42], exploiting the fact that polynomials can be seen as rewrite rules for an equational theory.

Cylindrical algebraic decomposition is a method for solving systems of non-linear inequalities which performs quantifier elimination [52]. These techniques are reviewed with regard to improvements and implementation details in [34]. The `RISC-CLP(Real)` system [35] implements a combination using improved variants of both algorithms. Apart from determining the decidability of non-linear constraints using exact real arithmetic, the `RISC-CLP` system is capable of removing non-query variables in answer constraints using quantifier elimination.

2.5. Constraint processing based on interval arithmetic

This class of methods is discussed at length in the subsequent sections, the common underlying principle is to encapsulate real numbers within intervals and to replace floating-point arithmetic with interval-based computation. The representation is accurate in that numbers are contained within 'safely' (outward) rounded bounds. Furthermore, constraint propagation methods can be used that contract the initial intervals either until an empty interval is obtained (no solution) or a minimal one is found; where 'minimal' depends either on the granularity of the floating-point hardware or on a precision

[1] *Contrainte avec Logique*

bound set by the user. Existence proofs can be given for these methods such that one is assured that a non-empty interval obtained after propagation does indeed contain a solution. In combination with an exhaustive form of search, interval-based propagation techniques constitute a complete, reliable and rigorous solving technique; under reasonable assumptions, all solutions to a given problem can be obtained. Unlike some local methods, there is no dependence on using a good starting point and additional constraints can easily be integrated and exploited for increased propagation activity. Moreover, since the representation is finite ('safely' rounded floating point numbers as interval bounds), the convergence of these approaches can easily be verified. In summary, the advantages of this approach are rigour of computation, completeness and declarativity of knowledge representation (to an extent):- constraint programming aspires to the *'holy grail'* of declarative solving [3] which ideally means that the user merely has to state the problem without having to devise a special algorithm. In practice this means an increase of freedom for the user – available propagators allow narrowing for a wide range of constraints. Constraint processing can already go as far as dealing with ordinary differential equations, as documented in the sections to come.

3. Interval-based consistency techniques for problems over the Reals

As real numbers on computers are approximated by floating-point numbers (called "*machine numbers*" in [33, 44]) and real arithmetic by floating-point arithmetic, the presence and accumulation of rounding errors become inevitable.

The problems of floating-point arithmetic are well-known [19], to name a few:- certain real (and even rational decimal numbers) can at best be approximated by the IEEE 754/854 standard representation; operations involving numbers with large differences in exponents absorb the effect of the smaller numbers (e.g. $10^{30} + 1.03^{-6}$ still gives 10^{30}); catastrophic cancellation describes the situation where rounding errors eliminate the difference between nearly equivalent numbers. The semantics of real arithmetic do in general not carry over to floating-point arithmetic, changing the format of parenthesised expressions into a theoretically equivalent expression can increase or decrease the overall error [19, sec. 3.2.1]. This entails that essential properties of elementary arithmetic operators, such as the associativity of addition, do not necessarily always hold.

Example 3.1 Compare for instance $(a+b)+c$ and $a+(b+c)$ when $a = 10^{30}, b = -a, c = 1.0$:

```
main() { printf("%F %F", (10.0E30 - 10.0E30) + 1.0, 10.0E30 +
(-10.0E30 + 1.0)); }
```

The first result correctly evaluates to 1.0, whereas rounding in the second case wrongly yields 0.0.

With careful reformulation the results of elementary floating-point expressions can be kept within provable error bounds [19], this practice could also be extended to estimate

accumulated rounding errors (figure 2). Such analysis becomes, however, very difficult when many function evaluations are applied in an iterative or recursive fashion and the number of iterations is a priori unknown – as often in constraint processing. As a result, the overall rounding error can only be vaguely guessed. Another problem is that the IEEE standards are normative only with regard to elementary arithmetic and conversion operations, for transcendental functions such as tan, exp or log no recommendations were made due to the Table Maker's Dilemma [19, sec. 2.1.4], i.e. there is no universally applicable way of defining a rounding mode and hence the precision differs from implementation to implementation. The consequence for constraint solving is that none of

```
| ?- X is float(1)/10, Y is X+X+X + X+X+X + X+X+X + X.

X = 0.10000000000000001
Y = 0.99999999999999989
```

Figure 2. Accumulation of rounding errors

the local consistency notions known from finite domain problem solving directly carries over to real-valued CSPs. Interval-based consistency techniques avoid the problems of floating-point computation by performing equivalence-preserving approximations of local consistency.

3.1. Basic definitions regarding CSPs

In the following, we will first consider the definition of numerical CSPs in a format convenient for further discussion, then review the approximations of real numbers by floating-point intervals and finally introduce common local consistency notions in solving CSPs over the Reals.

3.1.1. Numerical constraint problems

A real-valued constraint $c(x_1,\ldots,x_n) \subseteq \mathbb{R}^n$ is an n-ary relation over the Reals, where each x_i ($1 \leqslant i \leqslant n$) takes on values from a domain $D_i \subseteq \mathbb{R}$. In the following, such n-ary relations will be abbreviated by the special symbol ρ. The relationship between an n-ary constraint and the domains of its associated variables is established by the notion of projection constraints.

Definition 3.1 (Projection) *The i-th projection ($i \in 1..n$) of an n-ary relation $\rho \subseteq \mathbb{R}^n$ is given as* $\pi_i(\rho) = \left\{x_i \in D_i \middle| \left(\exists(x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n) \in \mathbb{R}^{n-1}\right)(x_1,\ldots,x_n) \in \rho\right\}$.

A real-valued constraint problem P is a pair $\langle \mathcal{C}; \mathcal{D} \rangle$ where $\mathcal{C}$ is a set of constraints involving at most n variables and $\mathcal{D}$ is a set of domain expressions taking the form $x_i \in D_i$ ($1 \leqslant i \leqslant n$) where $D_i \subseteq \mathbb{R}$ is the domain of x_i. Solutions to numerical constraint problems are defined as in the general case.

3.1.2. General solving scheme

It is fortunate for the presentation of constraint-solving techniques that the underlying search schema for this domain is very simple and based on a dichotomic, exhaustive *branch-and-prune* principle.[2] We can not present all possible search schemes here and refer to [2] for further reference. In fact, many existing solvers for constraint problems over the Reals are based on a branch-and-prune algorithm, e.g. Numerica [29], RealPaver [23], Newton [28] or CLIP [32]; an implementation is e.g. described in [27]. Branch-and-prune is based on the recursive invocation of two steps:

1. PRUNING the search space by applying interval consistency techniques
2. BRANCHING: generating sub-problems by bisecting the domains of a variable

The *pruning* step ensures, through reduction of variable domains, that certain interval consistency notions hold (see section 3.3); *branching* on non-empty variable domains creates sub-problems built from the disjoint parts of the bisected variable domain. Thus the branching step creates a search tree 'on the fly', where the leaf nodes represent solved states, i.e. intervals that where either found to be empty or which can not be narrowed any further. The recursive invocation of pruning and branching continues until no further narrowing of variable domains is possible or a user-defined termination criterion was met. The problem is proven to have no solution once an empty variable domain is encountered. A variation of this scheme is *branch-and-bound* search where subtrees can be pruned away as soon as it becomes clear that the estimated gain for the objective function is not going to improve on the current bound.

3.2. Approximating Reals by floating-point intervals

The set $\mathbb{R}$ of real numbers is infinite and each of its elements may have arbitrarily many digits after the decimal point. In contrast, the floating-point numbers that are available on a computer are a finite subset $\mathcal{F}$ of $\mathbb{R}$, where each element has a fixed, implementation-dependent precision. Additionally, the two special symbols $-\infty$ and ∞ are used to denote real numbers which are strictly smaller (larger) than the smallest (largest) floating-point number available in an implementation, respectively. Figure 3 illustrates these concepts.

3.2.1. Notational conventions

We use $\mathbb{R}^\infty = \mathbb{R} \cup \{-\infty, \infty\}$ to denote $\mathbb{R}$ augmented by the two symbolic constants $-\infty$ and ∞.[3] The ordering relation $<$ is extended such that for all $r \in \mathbb{R}$, $-\infty < r < \infty$. The symbol $\mathcal{F}$ refers to the set of all floating-point numbers in a given implementation. We have that $\mathcal{F} \subset \mathbb{R}^\infty$, so $\mathcal{F}$ is also ordered under $<$. For any floating-point constant a, a^+ corresponds to the smallest number in $\mathcal{F}$ strictly greater than a and a^- denotes

[2] According to [46], this branching principle underlies almost all global optimization algorithms, so it is not exclusive to constraint methods.

[3] The set $\mathbb{R}^\infty$ is sometimes also called the set of *extended Reals* [30, sec. 4].

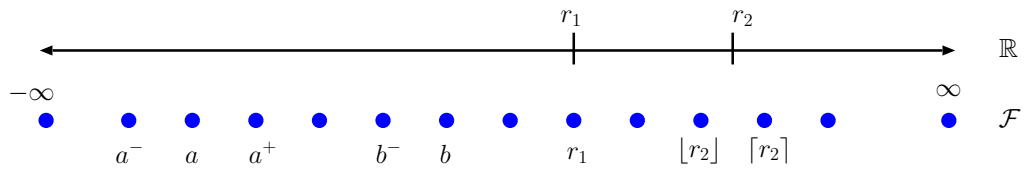

Figure 3. Real numbers $\mathbb{R}$ and floating-point numbers $\mathcal{F}$

the largest number in $\mathcal{F}$ strictly smaller than a (cf. figure 3). We will further use two functions to describe the effects of rounding.

Definition 3.2 (Rounding) *The functions* $\lfloor\ \rfloor : \mathbb{R} \to \mathcal{F}$, $\lceil\ \rceil : \mathbb{R} \to \mathcal{F}$ *are defined for all* $r \in \mathbb{R}$ *as*

$$\lfloor r \rfloor = \max\{s \in \mathcal{F} \mid s \leqslant r\}$$
$$\lceil r \rceil = \min\{s \in \mathcal{F} \mid s \geqslant r\}$$

3.2.2. Approximation in general

Informally speaking, the type of approximation embodied by the above rounding functions incurs a loss of information. In order to describe approximation which does not lead to such loss in a formal and generic way, *approximation functions* were studied in [4, 9]. We review the most important properties of these functions before turning to concrete instances.

Definition 3.3 (Approximation function [4]) *Let D be a set and $\wp(D)$ be the power set of D Let further $\underline{apx}$ be a function defined from $\wp(D)$ into $\wp(D)$. The function $\underline{apx}$ is an approximation over D if the following properties hold for all $\rho, \rho' \in \wp(D)$:*

1. $\underline{apx}(\emptyset) = \emptyset$
2. $\rho \subseteq \underline{apx}(\rho)$
3. $\rho \subseteq \rho' \Rightarrow \underline{apx}(\rho) \subseteq \underline{apx}(\rho')$
4. $\underline{apx}(\underline{apx}(\rho)) = \underline{apx}(\rho)$

In general, an n-ary relation ρ is approximated by the Cartesian product of the approximations of its projections.

Definition 3.4 (Approximations of n-ary relations [4]) *Let $\underline{apx}$ be an approximation function over a set D and let ρ be an n-ary relation over D. Then*

$$\underline{apx}(\rho) = \left(\underline{apx}(\pi_1(\rho)) \times \cdots \times \underline{apx}(\pi_n(\rho))\right)$$

The following important properties have been shown for relations $\rho \subseteq \mathbb{R}^n, \rho' \subseteq \mathbb{R}^n$ in [9, sec. 2].

$$\begin{array}{lcl} \underline{apx}(\rho \cup \rho') & = & \underline{apx}(\underline{apx}(\rho) \cup \underline{apx}(\rho')) \\ \underline{apx}(\rho \cap \rho') & \subset & \underline{apx}(\rho) \cap \underline{apx}(\rho') \end{array} \tag{1}$$

3.2.3. Real and floating-point intervals

An important result with respect to using floating-point intervals shall be summarized first. The definition of closed connected sets [30] (called *convex sets* in [39, sec. 2.3]) is of particular relevance here.

Definition 3.5 (Closed sets, connected sets) *A basic open set of Reals is a set of the form* $\{x \in \mathbb{R} | a < x < b\}$ *where* a, b *are real numbers. A set S of Reals is open if, for every point* $x \in S$*, there is a basic open set* U_x *such that* $x \in U_x \subseteq S$*. A set S of Reals is closed if its complement is open. A set S of Reals is connected if there do not exist disjoint non-empty open sets* U_1 *and* U_2 *that each intersect S and for which* $S \subseteq U_1 \cup U_2$*.*

As in [2, 30, 33], we use this definition as the basis for the definition of real intervals.

Definition 3.6 (Real interval) *A real interval is a closed connected set* $S \subseteq \mathbb{R}$*.*

The important result here is that the only closed connected sets of Reals are these [30, Thm. 1]:

$$\begin{array}{ll} \{x \in \mathbb{R} \,|\, a \leqslant x\}, & \{x \in \mathbb{R} \,|\, x \leqslant b\} \\ \{x \in \mathbb{R} \,|\, a \leqslant x \leqslant b\}, & \text{and } \mathbb{R} \end{array}$$

where $a, b \in \mathbb{R}$. This entails that $\emptyset$ is a closed connected set. These four closed connected sets are grouped into the second column of table 1 and are the basis for the definition of floating-point intervals, which also establishes the relationship with the set $\mathbb{R}^{\infty}$ (similar to [30, Def. 7]).

$[\underline{x}, \overline{x}) = \{x \in \mathbb{R} \,|\, \underline{x} \leqslant x < \overline{x}\}$	$[\underline{x}, \overline{x}] = \{x \in \mathbb{R} \,|\, \underline{x} \leqslant x \leqslant \overline{x}\}$
$(\underline{x}, \overline{x}] = \{x \in \mathbb{R} \,|\, \underline{x} < x \leqslant \overline{x}\}$	
$(\underline{x}, \infty) = \{x \in \mathbb{R} \,|\, \underline{x} < x < \infty\}$	$[\underline{x}, \infty) = \{x \in \mathbb{R} \,|\, \underline{x} \leqslant x < \infty\}$
$(-\infty, \overline{x}) = \{x \in \mathbb{R} \,|\, -\infty < x < \overline{x}\}$	$(-\infty, \overline{x}] = \{x \in \mathbb{R} \,|\, -\infty < x \leqslant \overline{x}\}$
$(\underline{x}, \overline{x}) = \{x \in \mathbb{R} \,|\, \underline{x} < x < \overline{x}\}$	$(-\infty, \infty) = \mathbb{R}$

Table 1. Real interval notation

Definition 3.7 (Floating-point interval) *Let* $\underline{x}, \overline{x} \in \mathcal{F}$*. A floating-point interval* $\langle \underline{x}, \overline{x} \rangle$ *is one of the four real intervals* $[\underline{x}, \overline{x}]$*,* $(-\infty, \overline{x}]$*,* $[\underline{x}, \infty)$ *or* $(-\infty, \infty)$*, using the notation in table 1. The set of all floating-point intervals is denoted by* $I(\mathcal{F})$*.*

3.2.4. Rounding and representing single Reals

The rounding functions of definition 3.2 preserve equivalence with $r \in \mathbb{R}$ only in the special case where $r = \lfloor r \rfloor = \lceil r \rceil$. Therefore (sets of) real numbers are usually represented by floating-point intervals [4, 8, 9, 27, 29]. As illustrated in figure 3, there are only two possibilities to represent a real number r by a floating-point interval [39]. In the first case, when r is equal to a floating-point number r_1, no rounding is required and the interval $[r_1, r_1]$ can be used. In the other case, a real number r_2 can be represented by the floating-point interval $\langle \lfloor r_2 \rfloor, \lceil r_2 \rceil \rangle$. Please note that the latter case constitutes the slight imprecision of including the interval bounds (if different from $-\infty$ or ∞). This is conform with the literature (e.g. [28, 29, 36]) and has the advantage of a uniform notation. Such "*outward rounding*" shall be used in the same manner when enclosing sets of Reals by intervals as it is used for single numbers.

It is important to emphasize that this text refers to properly *rounded interval arithmetic* [44, sec. 3.2], not *exact interval arithmetic*; we suppose that rounding is always done in such a manner that the exact result (of the corresponding infinite-precision arithmetic) is contained. We further assume that, in all operations described hereafter, the largest possible computing error is always smaller than the minimal value for $|a - a^+|$ where $a \in \mathcal{F}$, cf. [14, sec. 2.2]. This simplifies the exposition but does not necessarily reflect the practical reality, the careful reader should consider [19] and critically check which guarantees (perfect rounding) are provided by the software at hand.

Definition 3.8 (Approximation of real numbers) *Let $r \in \mathbb{R}$. The approximation of r, denoted $appx(r)$, is defined as $appx(r) = \langle \lfloor r \rfloor, \lceil r \rceil \rangle$.*

For simplicity, $appx(r)$ is abbreviated as $\overleftrightarrow{r}$. Since the arguments are singletons, conditions (3) and (4) of definition 3.3 do not apply to this special case; the generalisation follows below. It is important to note that condition (2) holds, since for all $r \in \mathbb{R}$, $\{r\} \subseteq \langle \lfloor r \rfloor, \lceil r \rceil \rangle$ using the given definitions and assumptions, and condition (1) vacuously holds. Concluding, for any real number r, its floating-point interval approximation is either of the form $\langle a, a \rangle$ or $\langle a, a^+ \rangle$. These special intervals are called *canonical intervals* in the literature [28, 29, 49].

3.2.5. Approximating sets of Reals

Under certain conditions, a set $S \subset \mathbb{R}$ may be representable by a real interval. For instance, the set $\{-30.0, 1.3, 3.0\}$ can at best be enclosed in e.g. $[-30, 3]$, but it is not representable as an interval. Further, since bounded non-empty real intervals are only closed for the interval operations of addition, subtraction and multiplication [30, Thm. 2], several interval operations require to use *unions* of intervals for the representation of results (e.g. interval division [30, sec. 4.7] or interval exponentiation [33, sec. 5.2]). This motivates the definition of a special set $U(\mathcal{F})$ whose elements are unions of floating point intervals [4, 8]:

$$U(\mathcal{F}) \quad = \quad \{D \subseteq \mathbb{R} \,|\, (\exists \langle I_1, \ldots, I_n \rangle \in I(\mathcal{F})^n)\, D = \textstyle\bigcup_{i=1}^{n} I_i\}$$

As pointed out in [4], both $I(\mathcal{F})$ and $U(\mathcal{F})$ are closed under intersection. The union approximation and hull approximation of a set can now be defined.

Definition 3.9 (Union and hull approximation of a set) *Let S be a subset of $\mathbb{R}$. The union approximation of S, denoted by $appx(S)$,[4] is the least element $\underline{U} \in U(\mathcal{F})$ under the ordering of set inclusion such that $\underline{U} \supseteq S$. The F-interval hull of S, denoted $hull(S)$, is the smallest (wrt. inclusion ordering) element $\underline{I} \in I(\mathcal{F})$ such that $\underline{I} \supseteq S$.*

The notion of "smallest element" in the inclusion ordering is understood with regard to the use of outward rounding in order to preserve the computational correctness. It is pointed out in [4, sec. 4.4] that both functions satisfy definition 3.3 and that additionally the following properties hold:

1. $appx(S) \subseteq hull(S)$
2. $hull(appx(S)) \subseteq appx(hull(S)) = hull(S)$
3. $hull(\{r\}) = appx(\{r\}) = \overleftrightarrow{r}$

These approximations further have orthogonal strengths. Representing sets of Reals by unions of intervals can lead to combinatorial explosion, [39, sec. 3] shows a simple example where representing an arc-consistent domain requires the union of 10^4 intervals. On the other hand, approximating a set which is not closed and connected by an interval hull reduces the precision of returned results. Some tools, e.g. RealPaver [23], allow users to switch between union and hull output mode.

3.2.6. Approximating n-ary relations over the Reals

An n-ary relation $\rho \subseteq \mathbb{R}^n$ is a subset of a Cartesian product $D_1 \times \cdots \times D_n$ of sets $D_i \subseteq \mathbb{R}$, $i \in [1,n]$. Such relations can be approximated by Cartesian products of floating-point intervals, called F-boxes.

Definition 3.10 ($\mathcal{F}$-box) *An n-ary real box is a Cartesian product $R_1 \times \cdots \times R_n$ where for all $i \in [1,n]$ R_i is a real interval. An n-ary F-box is a Cartesian product $F_1 \times \cdots \times F_n$ where $F_i \in I(\mathcal{F})$ for all $i \in [1,n]$.*

The inclusion ordering is generalized from $I(\mathcal{F})$ to its n-fold Cartesian product $I(\mathcal{F})^n$. Lemma 8 in [33] guarantees the existence of a unique least F-box for every n-ary relation $\rho \subseteq \mathbb{R}^n$, such that applying definition 3.4 to the hull approximation yields $hull(\rho)$ as the smallest F-box containing ρ [33, Def. 9].[5] This is the approach taken in [4, 9], where n-ary relations ρ are represented by the smallest F-box containing ρ and the approximation functions of definition 3.9 are extended by applying definition 3.4. These concepts can furthermore be formally described in terms of (monotone) interval extensions [8, 29].

[4] called $\underline{union}(r)$ in [4, Def. 10].

[5] instead of F-box, the term "*machine box*" and instead of $hull(\rho)$ the expression $bx(\rho)$ is used in [33].

Definition 3.11 (Interval extension) *An interval function $F : I(\mathcal{F})^n \to I(\mathcal{F})$ is an interval extension of a function $f : \mathbb{R}^n \to \mathbb{R}$ if $f(a_1,\ldots,a_n) \in F(\overleftrightarrow{a_1},\ldots,\overleftrightarrow{a_n})$ for every $(a_1,\ldots,a_n)$ in the domain of f. An interval relation $R \subseteq I(\mathcal{F})^n$ is an interval extension of a relation $r \subseteq \mathbb{R}^n$ if $(\forall (a_1,\ldots,a_n) \in r)\ (\overleftrightarrow{a_1},\ldots,\overleftrightarrow{a_n}) \in R$.*

Table 2 shows some interval extensions of arithmetic operations. The general case of interval division is complex and we refer to [33] for an in-depth treatment. Many inter-

$$
\begin{array}{lcl}
[a_1,b_1] \oplus [a_2,b_2] & = & [\lfloor a_1+a_2 \rfloor, \lceil b_1+b_2 \rceil] \\
[a_1,b_1] \ominus [a_2,b_2] & = & [\lfloor a_1-b_2 \rfloor, \lceil b_1-a_2 \rceil] \\
[a_1,b_1] \otimes [a_2,b_2] & = & [\min(\lfloor a_1a_2 \rfloor, \lfloor a_1b_2 \rfloor, \lfloor b_1a_2 \rfloor, \lfloor b_1b_2 \rfloor), \\
 & & \max(\lceil a_1a_2 \rceil, \lceil a_1b_2 \rceil, \lceil b_1a_2 \rceil, \lceil b_1b_2 \rceil)] \\
[a_1,b_1] \oslash [a_2,b_2] & = & [\min(\lfloor \frac{a_1}{a_2} \rfloor, \lfloor \frac{a_1}{b_2} \rfloor, \lfloor \frac{b_1}{a_2} \rfloor, \lfloor \frac{b_1}{b_2} \rfloor), \\
 & & \max(\lceil \frac{a_1}{a_2} \rceil, \lceil \frac{a_1}{b_2} \rceil, \lceil \frac{b_1}{a_2} \rceil, \lceil \frac{b_1}{b_2} \rceil)];\ \text{if } 0 \notin [a_2,b_2]
\end{array}
$$

Table 2. Interval extensions for arithmetic operations

val functions and relations satisfy definition 3.11, e.g. an interval function that always returns $(-\infty,\infty)$. With regard to approximation, the interest is to find such interval extensions that are *optimal* in the sense that there is no 'smaller' interval extension.

Definition 3.12 (Monotone extensions [8]) *An interval function $F : I(\mathcal{F})^n \to I(\mathcal{F})$ is monotone if $(\forall i \in [1,n])\, I_i \subseteq I_i' \Rightarrow F(I_1,\ldots,I_n) \subseteq F(I_1',\ldots,I_n')$. An interval relation $R \subseteq I(\mathcal{F})^n$ is monotone if $(\forall i \in [1,n])\, I_i \subseteq I_i' \Rightarrow [(I_1,\ldots,I_n) \in R \Rightarrow (I_1',\ldots,I_n') \in R]$.*

Note how this definition relates closely to definition 3.3(3). The two preceding definitions lead to a very important result, taken from [8, Thm. 5]:

Theorem 3.1 (Fundamental theorem of interval arithmetic) *Let $F : I(\mathcal{F})^n \to I(\mathcal{F})$ be a monotone interval extension of $f : \mathbb{R}^n \to \mathbb{R}$. Then $a_1 \in I_1,\ldots,a_n \in I_n$ implies $f(a_1,\ldots,a_n) \in F(I_1,\ldots,I_n)$. Similarly, let $R \subseteq I(\mathcal{F})^n$ be a monotone interval extension of $r \subseteq \mathbb{R}^n$. Then $a_1 \in I_1,\ldots,a_n \in I_n$ implies $(a_1,\ldots,a_n) \in r \Rightarrow (I_1,\ldots,I_n) \in R$.*

An important and simple interval extension of a function $f : \mathbb{R}^n \to \mathbb{R}$ or a relation $r \subseteq \mathbb{R}^n$ is the *natural interval extension* [14, 27, 28], which is obtained as follows. Each occurrence of a constant $k \in \mathbb{R}$ is replaced by $\langle \lfloor k \rfloor, \lceil k \rceil \rangle$, each variable x_i by an interval variable X_i and each operation is replaced by its optimal interval extension. In the remainder of the paper, natural interval extensions will be denoted by $\hat{f}$ and $\hat{r}$, respectively, and we will use the interval extensions of $+,-,*,/$ as shown in table 2. More sophisticated variants, such as the distributed interval extension or the Taylor interval extension [27, 28], exist (see also [10, chap. 5]).

3.3. Local consistency forms

The previous section has considered the approximations of (sets of) Reals, relations and functions by (Cartesian products of) intervals. This section extends these principles to the approximation of local consistency in constraint problems over the Reals.

3.3.1. Approximations of arc-consistency

Most of the local consistency forms discussed below derive from (hyper-) arc consistency[6], which is defined as follows.

Definition 3.13 (Arc-consistency) *The i-th projection* $\pi_i(\rho)$ *of an n-ary constraint* $\rho \subseteq \mathbb{R}^n$ *is arc-consistent with respect to the sequence of domains* $D_1, \ldots, D_n$ *if for all* $i \in [1,n]$

$$D_i = D_i \cap \left\{ r_i \,|\, (\forall j \in [1,n] \setminus \{i\})\, (\exists r_j \in D_j)\, (r_1, \ldots, r_n) \in \rho \right\}$$

An n-ary constraint $\rho \subseteq \mathbb{R}^n$ *is arc-consistent wrt. the sequence of domains* $D_1, \ldots, D_n$ *if each of its projections* $\pi_i(\rho)$ $(1 \leqslant i \leqslant n)$ *is arc-consistent. A constraint problem* $\langle \mathcal{C}; \mathcal{D} \rangle$ *is arc-consistent if each constraint* $c \in \mathcal{C}$ *is arc-consistent with regard to the sequence of domains in* $\mathcal{D}$.

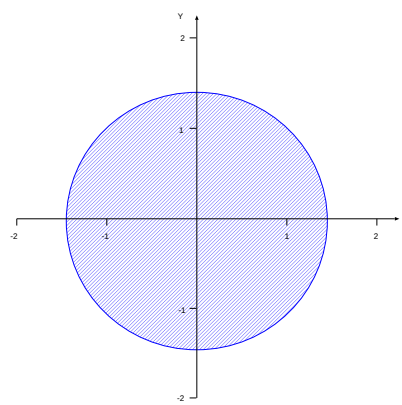

Figure 4. Arc-consistent area for $x^2 + y^2 \leqslant 2$

The definition of local consistency conditions leads to the algorithmic process of *constraint propagation*, which exploits knowledge inherent in constraints to reduce the variable domains of the problem. Constraint propagation can generally be regarded as the iterative application of so-called domain reduction functions known as *chaotic iteration* [1]. Specific domain reduction functions (also called narrowing operators in [4, 5, 8]),

[6] this distinction refers to the fact that the original notion of arc-consistency [40] is limited to binary constraints. Definition 3.13 generalises arc-consistency to the *n*-ary case and will be used when referring to arc-consistency in this section.

which distinguish the local consistency approximations, will be considered below. The iterative process terminates by reaching a stable fixed-point called *closure*, which is characterized by the property that any further application of the reduction functions will not result in any change of the domains [1]. In the following, we will not consider the individual details of closures and the algorithms to achieve these and point to the respective literature instead [4, 8, 9, 14, 39]. Establishing arc-consistency on the i-th projection $\pi_i(\rho)$ of a constraint $\rho \subseteq \mathbb{R}^n$ as per definition 3.13 leads to a new variable domain D_i' which is a subset of the original domain $D_i \subseteq \mathbb{R}$.

Example 3.2 (Arc-consistent domains) Consider the constraint problem $\langle x^2 + y^2 \leqslant 2; x \in \mathbb{R}, y \in \mathbb{R} \rangle$. The arc-consistent domains for x, y form a disk in figure 4. The circular area contains infinitely many points (x, y) that satisfy the problem constraint.

Arc-consistency on real-valued problems thus depends on the exact representation of sets of Reals, which is generally made impossible on computers by inherently finite precision limits. Several approximations of arc-consistency have therefore evolved, all of which use floating-point intervals to describe variable domains over the Reals. The following definitions will only describe the consistency condition for single projection constraints, the extension of this condition to whole constraints ρ and entire constraint problems $\langle \mathcal{C}; \mathcal{D} \rangle$ is established as per definition 3.13. The first approximation of arc-consistency is called *interval consistency*, it is described in [8] and uses the union approximation of definition 3.9.

Definition 3.14 (Interval consistency) *The i-th projection $\pi_i(\rho)$ of an n-ary constraint $\rho \subseteq \mathbb{R}^n$ is interval-consistent with respect to the sequence of domains $D_1, \ldots, D_n$ if for all $i \in [1, n]$*

$$D_i = appx\left(D_i \cap \left\{ r_i \,|\, (\forall j \in [1,n] \setminus \{i\})\, (\exists r_j \in D_j)\, (r_1, \ldots, r_n) \in \rho \right\}\right)$$

In this case, sets of real numbers are simply represented by a union of corresponding floating-point intervals. Such an approach is feasible only for a few problem types, e.g. finite domain problems over integral numbers. For other types of problems establishing interval consistency may lead to either combinatorial explosion or prove infeasible due to infinite variable domains. Thus, most systems compute an approximation of arc-consistency called *2B-consistency* [39, Def. 9] or *hull-consistency* [4, 7, 9], which enforces the arc-consistency property only at the bounds of floating-point intervals. As in [14], we will use the terms 2B-consistency and hull-consistency interchangeably.

Definition 3.15 (Hull consistency) *The i-th projection $\pi_i(\rho)$ of an n-ary constraint $\rho \subseteq \mathbb{R}^n$ is hull-consistent with respect to the sequence of domains $D_1, \ldots, D_n$ if for all $i \in [1, n]$*

$$D_i = hull\left(D_i \cap \left\{ r_i \,|\, (\forall j \in [1,n] \setminus \{i\})\, (\exists r_j \in D_j)\, (r_1, \ldots, r_n) \in \rho \right\}\right)$$

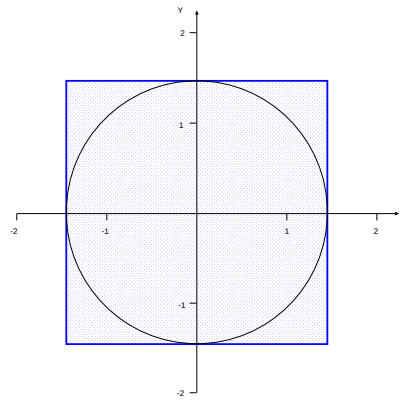

Figure 5. Hull-consistent area for $x^2 + y^2 \leqslant 2$

Notice that this is a weaker condition than arc-consistency in that hull-consistent domains may still contain inconsistent values.

Example 3.3 (Hull-consistency) Consider again the constraint problem $\langle x^2 + y^2 \leqslant 2; x \in \mathbb{R}, y \in \mathbb{R} \rangle$. The square area in figure 5 describes the Cartesian product of the variable domains after enforcing hull-consistency. In contrast (cf. figure 4), the inner disk describes the arc-consistent area.

A slightly stronger variant of hull consistency is used in the *tolerance propagation* scheme of [36], which requires the arc-consistency property to hold for each value within the interval bounds. The tolerance propagation scheme is thus a combination of interval and hull consistency notions, which entails that it may not be possible to enforce it on certain problems. To illustrate this, consider figure 6. The arc-consistency property holds for all points of the shaded square area, but this comes at the expense of excluding consistent values – those which lie outside the square area and within the disk. Hull-consistency can be enforced in a straightforward manner on primitive constraints, as detailed in section 3.3.2. On primitive constraints, hull-consistency is almost indistinguishable from box-consistency (definition 3.16 below). To better compare these two consistency notions, we use a property that has been shown in [14, Prop. 3] to hold for the natural interval extension (cf. page 173) of n-ary constraints in which no variable occurs more than once.[7]

Proposition 3.1 (Hull-consistency [14]) *Let $\rho \subseteq \mathbb{R}^n$ be an n-ary constraint taking the form $\rho(x_1, \ldots, x_n)$ such that each variable x_i occurs exactly once in ρ and is associated with a domain $D_i \subseteq \mathbb{R}$, $1 \leqslant i \leqslant n$. Let further $\hat{\rho}(I_1, \ldots, I_n)$ be the natural interval extension of $\rho(x_1, \ldots, x_n)$ where the i-th projection of $\hat{\rho}$ is the floating-point interval $I_i = \langle \underline{x_i}, \overline{x_i} \rangle$.*

[7] again, we consider 2B-consistency and hull-consistency as the same concepts.

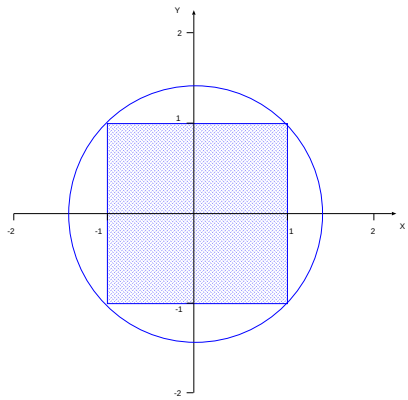

Figure 6. Consistent tolerance propagation area for $x^2 + y^2 \leqslant 2$

Then the i-th projection of $\hat{\rho}(I_1,\ldots,I_n)$ *is hull-consistent iff the following relations hold with regard to* ρ*:*

$$(1) \quad \hat{\rho}\left(I_1,\ldots,I_{i-1},[\underline{x_i},\underline{x_i}^{+}),I_{i+1},\ldots,I_n\right) \quad \textit{and}$$
$$(2) \quad \hat{\rho}\left(I_1,\ldots,I_{i-1},(\overline{x_i}^{-},\overline{x_i}],I_{i+1},\ldots,I_n\right)$$

The constraint $\hat{\rho}(I_1,\ldots,I_n)$ *is hull-consistent if all its projections are hull-consistent.*

We now turn to the definition of *box-consistency* [7, 8, 27, 28, 29], which is an even coarser approximation of arc-consistency than hull-consistency. It can be obtained by replacing all existentially quantified variables in definition 3.13 with the corresponding intervals. Thus, a system of n-ary constraints is turned into a system of univariate interval functions, which is a very convenient format for numerical methods such as the Newton interval method (see below). As pointed out in [14], box consistency can be defined as follows.[8]

Definition 3.16 (Box-consistency) *Let* $\rho(x_1,\ldots,x_n) \subseteq \mathbb{R}^n$ *be an n-ary constraint, each variable* x_i *associated with a domain* $D_i \subseteq \mathbb{R}$, $1 \leqslant i \leqslant n$. *Let further* $\hat{\rho}(I_1,\ldots,I_n)$ *be the natural interval extension of* $\rho(x_1,\ldots,x_n)$ *where the i-th projection of* $\hat{\rho}$ *is the floating-point interval* $I_i = \langle \underline{x_i}, \overline{x_i} \rangle$. *Then the i-th projection of* $\hat{\rho}(I_1,\ldots,I_n)$ *is box-consistent iff the following relations hold wrt.* ρ*:*

$$(1) \quad \hat{\rho}\left(I_1,\ldots,I_{i-1},[\underline{x_i},\underline{x_i}^{+}),I_{i+1},\ldots,I_n\right) \quad \textit{and}$$
$$(2) \quad \hat{\rho}\left(I_1,\ldots,I_{i-1},(\overline{x_i}^{-},\overline{x_i}],I_{i+1},\ldots,I_n\right)$$

[8] Box consistency was originally defined in [8], this definition differs slightly from the original definition. A third variant appears in [7, sec. 4.1]. We chose only one definition to avoid confusion and to enable a better comparison of the underlying principles.

The constraint $\hat{\rho}(I_1,\ldots,I_n)$ is box-consistent if all its projections are box-consistent.

The original definition of box consistency is parameterised by the type of interval extension to be used and thus can be instantiated to produce various narrowing operators, as illustrated in [27, 28].

3.3.2. Comparing approximations of arc-consistency

If we denote the Cartesian product of variable domains after enforcing arc-consistency, interval-consistency, hull-consistency and box-consistency by AC, IC, HC and BC, respectively, [8, Prop. 4.1] presents the following inclusion hierarchy: $AC \subseteq IC \subseteq HC \subseteq BC$.

To compare the strengths of hull- (2B) and box-consistency more closely, we now shortly review implementation features. Establishing hull-consistency is based on the principles of interval arithmetic, which generally restricts the selection of domain reduction functions to *interval-convex relations* [4, 9].

Definition 3.17 (Interval-convexity [9]) *An n-ary relation $\rho \subseteq \mathbb{R}^n$ is interval-convex if for every real box u and for all $i \in [1,n]$, $\pi_i(\rho \cap u)$ is a real interval. ρ is F-interval-convex if for every F-box u and for all $i \in [1,n]$, $\pi_i(\rho \cap u)$ is an F-interval.*

Only few arithmetic operations are interval-convex, even the multiplication relation needs to be represented as the disjoint union of two subsidiary interval-convex relations [9, p. 9]. Thus the principle of establishing hull-consistency is to use only a restricted set of basic domain reduction functions,[9] other, more complex (non interval-convex) constraints need to be decomposed into unions and intersections of these primitives. Moreover, the tight computation of interval hulls demands operations to be evaluated with a precision of 1 ulp,[10] which can not always be guaranteed for every elementary function [10, sec. 5.2.2].

Example 3.4 (Narrowing functions) Given the constraint $x+y=z$ with corresponding floating-point interval domains X, Y and Z. The domain reduction functions for X, Y and Z are (primed intervals denote reduced ones):

$$\begin{aligned} X' &= X \cap (Z \ominus Y) \\ Y' &= Y \cap (Z \ominus X) \\ Z' &= Z \cap (X \oplus Y) \end{aligned}$$

A drawback of the decomposition into constraint narrowing primitives is the *dependency problem* [14], i.e. narrowing based on the composition of narrowing operators is equal to the theoretically achievable narrowing only if each variable occurs at most once in a constraint [4, Thm. 2]. The decomposition process generates primitive constraints by introducing auxiliary variables, which is illustrated by the following example.

[9] the definition of 2B consistency in [36, sec. 2.3] also supposes that constraints are basic.

[10] *Units in the Last Place*, the absolute error of floating-point computations wrt. the last digits [19].

Example 3.5 (Multiple occurrences) Let $x_1 + x_2 = x_1$ be a constraint with the floating-point interval domains $X_1 = [-1,1]$ and $X_2 = [0,1]$. Without decomposition, establishing 2B consistency reduces X_2 to $[0,0]$, since there is no value for x_1 with $x_2 = 1$ such that the constraint holds. Decomposition of this constraint yields the system $\langle x_1 + x_2 = x_3,\ x_1 = x_3 \rangle$ with $X_3 = [-1,1]$. This decomposition now makes the consideration of x_1 independent from x_2 and so domain reduction can not be applied as effectively. In fact, it is straightforward to verify that application of narrowing operators (using e.g. those from example 3.4) leaves the domains entirely unaffected, as each interval variable bound in turn has a supporting value among the bounds of the other interval variables.

Box consistency, in contrast, does not suffer from the same problem, since its domain narrowing is not based on interval arithmetic. Rather, an n-ary constraint of the form $f(x_1,\ldots,x_n)$ is first transformed into a univariate interval function of the variable x_i ($1 \leqslant i \leqslant n$). An interval extension of the Newton method then reduces the interval domain of x_i by computing the leftmost and rightmost interval zeros[11] [14, 27]. However, as pointed out in [7, 14, 21, 22, 32], the computation of interval zeros is a time-consuming process and therefore [7, sec. 4.1] recommends not to use box-consistency when a constraint involves many different variables.

Improvements on the basic algorithms The basic algorithms to enforce hull and box consistency are called HC3 and BC3 (respectively) in [7], where improvements to these algorithms are made. The new HC4 algorithm [7, sec. 3.2] does no longer require decomposition when establishing hull consistency, this is automated by a traversal of an attribute tree representing the decomposed constraints. The traversal consists of a *forward evaluation* and a *backward propagation* phase during which the interval domains of the involved variables are continuously reduced. Based on these ideas, a new hybrid algorithm called BC4 is then introduced [7, sec. 4.2] which interleaves the computation of hull and box-consistency. The experimental results confirm that BC4 combines the strengths of the two consistency notions in that it outperforms both BC3 and HC3 on nearly all of the benchmarks with a faster convergence.

This line of research into combining the strengths of different consistency techniques is continued by Granvilliers, who proposes a branch-and-prune algorithm using a combination of hull-consistency, box consistency and the interval Newton method in [22]. The combined strategy outperforms Numerica and a state-of-the-art numerical continuation method on benchmarks, and it proves to be faster on average than the pruning based on a single type of consistency. In particular, the cooperative strategy helps to avoid slow convergence tendencies. Combined consistency strategies are also implemented in RealPaver (cf. section 4.3), a branch-and-prune solver developed by the same author.

3.3.3. Higher-order consistencies

Higher order consistencies derive from the notion of k-consistency [18]. The approximations of arc-consistency can be described in terms of (strong) 2-consistency; the

[11] with regard to finite precision of computation these are often more appropriately called "*quasi-zeros*".

following will present two notions of strong 3-consistency. These can in theory be generalized to even higher levels [39]. The principle immanent in the following definitions of 3B and bound consistency is that a variable is instantiated to one of its floating-point interval bounds and the remaining constraint problem is made 2B or box consistent, respectively.

Definition 3.18 (3B-consistency [39]) *Let $\mathcal{P} = \langle \mathcal{C}; \mathcal{D} \rangle$ be a CSP and x be any of the variables in $\mathcal{D}$. Let further*

- *$\mathcal{P}'$ be the problem derived from $\mathcal{P}$ by substituting $[\underline{x}, \underline{x}^{+})$ for x*
- *$\mathcal{P}''$ be the problem derived from $\mathcal{P}$ by substituting $(\overline{x}^{-}, \overline{x}]$ for x*

The domain of x is 3B-consistent if establishing 2B-consistency on both $\mathcal{P}'$ and $\mathcal{P}''$ does not lead to an empty variable domain. $\mathcal{P}$ is 3B-consistent if all the domains in $\mathcal{D}$ are 3B-consistent.

This definition entails that a 3B-consistent problem is automatically also 2B-consistent.

Example 3.6 (3B-consistency [39]) Consider the following binary constraint problem $\mathcal{P}$:

$$\langle x + y = 2 \wedge y \leqslant x + 1 \wedge y \geqslant x; x \in [0.5, 1], y \in [1, 1.5] \rangle$$

Table 3 illustrates the proof that the problem is 3B-consistent. For each instantiation of x, a 2B-consistent (unary) sub-problem can be found. The converse case of instantiating y to its interval bounds and determining whether the remaining sub-problem is 2B-consistent follows analogously.

$\mathcal{P}'$	$y = 1$	$y = 1.5$	$\mathcal{P}''$	$y = 1$	$y = 1.5$
$x + y = 2 \wedge x = 0.5$	*fail*	*true*	$x + y = 2 \wedge x = 1$	*true*	*fail*
$y \leqslant x + 1 \wedge x = 0.5$	*true*	*true*	$y \leqslant x + 1 \wedge x = 1$	*true*	*true*
$y \geqslant x \wedge x = 0.5$	*true*	*true*	$y \geqslant x \wedge x = 1$	*true*	*true*

Table 3. Establishing 3B consistency

Bound-consistency [29, 49] relies on the same principle as 3B-consistency but establishes box-consistency on the remaining sub-problem.

Definition 3.19 (Bound-consistency [29]) *Let $\mathcal{P} = \langle \mathcal{C}; \mathcal{D} \rangle$ be a CSP and x be any of the variables in $\mathcal{D}$. Let further*

- *$\mathcal{P}'$ be the problem derived from $\mathcal{P}$ by substituting $[\underline{x}, \underline{x}^{+})$ for x*
- *$\mathcal{P}''$ be the problem derived from $\mathcal{P}$ by substituting $(\overline{x}^{-}, \overline{x}]$ for x*

The domain of x is bound-consistent if establishing box-consistency on both $\mathcal{P}'$ and $\mathcal{P}''$ does not lead to an empty variable domain. $\mathcal{P}$ is bound-consistent if all the domains in $\mathcal{D}$ are bound-consistent.

A spectacular application of box and bound consistency techniques in a branch-and-prune algorithm is documented in [49], where this approach reduced the runtime of solving a numerical transistor modelling problem from 14 months (on 30 Sun Sparc-1 stations) to less than one hour (on a single Sun Ultra-2 station).

3.3.4. Variations to allow faster convergence

By default, interval-based consistency techniques aspire to approximate real values during domain narrowing as close as possible, computing e.g. canonical intervals or canonical boxes (section 3.2.4). This can be computationally very expensive and lead to lengthy computations that in some cases may require manual termination [39, sec. 3.1]. Therefore, several relaxations of known consistency techniques have evolved, which all permit an imprecision tolerance when computing new interval bounds. Hull-consistency is relaxed to arc-B(w) consistency in [39, Def. 11], where w is the tolerance parameter for the computation of new interval bounds. This means that computed results need to lie within a tolerance interval which contains at most w floating-point numbers in between of its bounds, which generalises the use of canonical intervals to enclose real-valued results. Thus, 2B-consistency is equivalent to arc-B(0) consistency. In the same manner, 3B-consistency has been relaxed to 3B(w_1, w_2) consistency [39, Def. 13], now using two width dimensions. Again, 3B-consistency is equivalent to 3B$(0, 0)$ consistency.

The analogue of arc-B(w) consistency for box-consistency is box$_\varphi$-consistency [21, Def. 4], where φ is the parameter limiting the width of the computed interval bounds. Box$_\varphi$-consistency significantly reduced the computation times with regard to computing exact box-consistency in all of the benchmarks reported in [21, sec. 5.1]. See figure 9 for an example of using box$_\varphi$-consistency.

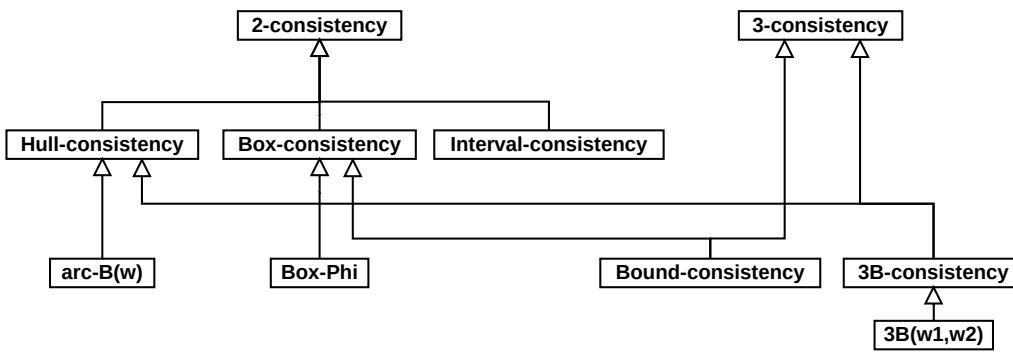

Figure 7. Interval consistency techniques

3.4. The big picture

Figure 7 summarizes the different consistency notions where arrows denote conceptual relations (not inheritance). Concepts that are known under different names, such as hull-, 2B- and arc-B consistency appear under a single name only.

4. Implementations

This section presents some implementations of solvers for non-linear constraint problems that are based on interval consistency techniques. Table 4 surveys the availability of consistency techniques in a selection of solvers.

Language	Ref	Basis	Hull Cons.	Box. Cons.	Remarks
Prolog IV	[45]	Prolog III	•		
Newton	[28]	Prolog		•	
DecLIC	[21]	clp(FD)	•	•	box_φ consistency
clp(BNR)	[9]	BNR Prolog	•		
Numerica	[29]	Ilog		•	bound consistency
RealPaver	[23]	C	•	•	HC3..4, BC3..5, 3B

Table 4. Features of implementations

4.1. Jail

JAIL ("*Just Another Interval Library*") [20] is an interval arithmetic library based on clever exploitation of IEEE 754 features. The availability of JAIL is unclear, but the same author has released an open-source library version called Gaol ("*Not Just Another Interval Library*")[12] whose outstanding feature among comparable C++ libraries is the implementation of relational interval arithmetic operators which can be used to implement constraint narrowing functions.

4.2. Numerica

Numerica [29] is a modelling language and constraint solver for (non-) linear constraint problems over the Reals. The constraint solver uses a branch-and-prune algorithm developed earlier [27] by the authors for the Newton [28] constraint language. The solver requires a square system[13] of non-linear (in-) equations [29, p. 96]. The available consistency techniques are box consistency and bound consistency, using the natural or Taylor interval extensions. For optimisation problems, a branch-and-bound algorithm is used. Previous experiences in developing the Helios modelling language as an interface for Newton had contributed to a research prototype with features quite similar to those available in Numerica [41]. The standalone version of Ilog Numerica is no longer available, but these and related concepts have been merged into the Solver library (section 4.7). As an example of the Numerica style, figure 8 shows the code for the Broyden banded function problem [29, p. 89], which amounts to finding the zeros for the system

[12] `http://sourceforge.net/projects/gaol/`

[13] in a square system, the number of equations equals the number of unknowns.

of n equations:

$$f(x_1,\ldots,x_n) = x_i * (2+5*x_i^2) + 1 - \Sigma_{j\in J_i} x_j * (1+x_j)$$

where for $1 \leqslant i \leqslant n$, $J_i = \{j | \max(1,i-5) \leqslant j \leqslant \min(n,i+1) \wedge j \neq i\}$.

```
Input:
    int n : "Number of variables: ";
Constant:
    range idx = [1..n];
Set:
    J[i in idx] = { j in [max(1,i-5)..min(n,i+1)] | j <> i };
Variable:
    x : array[idx] in [-10e8..10e8];

Body:
    solve system all
      [i in idx]:
          0 = x[i] * (2+5*x[i]^2) + 1 - Sum(j in J[i]) x[j] *
(1+x[j]);
```

Figure 8. The Broyden banded function problem in Numerica

4.3. RealPaver

RealPaver[14] is a constraint solver and language for problems over the Reals [23]. It can solve systems of (non-) linear equations and (in-) equations. The solver uses a branch and prune strategy and provides the hull- (HC3, HC4), box- (BC3, BC4, BC5) consistency techniques, also combined with interval Newton, and two variants of Lhomme's 3B consistency [39].

4.4. DecLIC

DecLIC [21] is an interval solver extension of clp(FD) [13] which provides hull, box and box$_\varphi$ consistency as domain narrowing techniques. In DecLIC, constraints over heterogeneous domains (Reals, Integers and Booleans) may be used in the same program, as well as combinations of hull and box consistency operators. The choice of which consistency technique to use is left to the programmer. Figure 9 shows a simple DecLIC program to compute the Broyden banded function. The statement `box_precision(0.1)` sets the parameter for box$_\varphi$-consistency, exact box-consistency is achieved when $\varphi = 0$ [6, p. 154]. (To use hull-consistency instead of box-consistency, this statement needs to be removed and the two instances of `$$=` be replaced by `$=` [21].)

[14] http://www.sciences.univ-nantes.fr/info/perso/permanents/granvil/realpaver/

```
broyden2 :- box_precision(0.1), %% box-phi
    domain([X1,X2],-1,1),
    X1*(2+5*X1**2) + 1 - (X1*(1+X1) + X2*(1+X2)) $$= 0 and
    X2*(2+5*X2**2) + 1 - (X1*(1+X1)) $$= 0,
    solve([X1,X2]).
```

Figure 9. The Broyden banded function problem with 2 variables in DecLIC

4.5. CLIP

CLIP [32] is an extension of GNU-PROLOG [17] which syntactically integrates interval arithmetic constraints via enclosing these in curly braces. Technically, CLIP uses an external constraint solver which is based on an interval arithmetic library, available as open source.[15] The distinguishing feature of CLIP is the combination of the ideas underlying systems like clp(BNR) [9, 48] and Numerica in such a way that interesting possibilities for solving real-valued problems become possible. These will be shortly presented below. The foundation of the CLIP architecture is a set of domain narrowing functions for primitive constraints such as e.g. arithmetic constraints and (in-) equalities. This is very similar in concept to the (hull-consistency) approach taken in clp(BNR), as pointed out in [32, sec. 1]. The algorithms for the elementary domain narrowing functions were developed in [30, 33] and form the basis of the interval arithmetic library. Within the architecture of elementary domain narrowing functions integrated into Prolog, higher-level constraint contractors are constructed by using Taylor extensions of functions,[16] which is presented in [32, sec. 4] using a univariate Taylor formula of degree one in combination with symbolic and numerical differentiation. These exten-

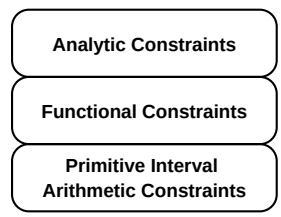

Figure 10. CLP(F) Architecture

sions are taken further in [31], where a three-layer architecture of constraint solving called CLP(F) is introduced, sketched in figure 10. The underlying assumption is that all considered functions over the Reals are infinitely often differentiable on a given interval [31, sec. 3]. *Functional constraints* apply to functions and comprise constraining variables to be of type function, equivalence constraints among functions and constraints on the range of functions. *Analytic constraints* extend functional constraints by three fea-

[15] http://interval.sourceforge.net/interval/

[16] this imposes restrictions such as differentiability on the class of functions that may be used.

tures: an n-fold differentiation operator, an operator to evaluate a given function at any point of an interval, and constraints on vectors of real or function variables. Analytic constraints in combination with the other constraints can be used to construct formulas that specify numerical differentiation, integration as well as constraints on ordinary differential equations (ODEs). Examples of the latter include among others initial value problems and parameter determination problems [31, sec. 5]. For instance, to find out where $\frac{d^2}{dx}e^{\sin(x)}$ has a zero in the interval $[0.5, 1]$, a query as shown in figure 11 can be used. The solving strategy for analytic constraints consists of reducing these to func-

```
| ?- type([F,T], function(0.5,1.0)),
     identity(T), {[F = exp(sin(T)), eval(ddt(F,2),X)=0.0]}.

  X = 6.6623943...e-01 ?
  (5420 ms) yes
```

Figure 11. Analytic constraints in CLIP

tional constraints which can then be solved with interval arithmetic methods as before. Although the analytic constraint solving approach is not always as efficient as other methods and exhibits some limitations as stated in [31, sec. 8], it can nevertheless be seen as a step forward into very interesting applications of constraint technology.

4.6. ECLiPSe IC Library

The IC (Interval Constraint) library of the ECLiPSe system provides a general interval propagation solver which can be used to solve hybrid problems over integer and real variables (both types of variables can be used in the same program). The IC solver performs almost all of its computation using interval arithmetic and provides a wide range of arithmetic, relational, non-linear and reified constraints, coupled with the facility to create user-defined constraints over Reals and Integers [11]. A standard branch-and-prune approach is provided, as well as a stronger propagation using the Squash algorithm [11, sec. 3.2.10].

4.7. Ilog Solver

The more recent variant of Ilog Solver [37] is able to solve non-linear problems over the Reals using interval propagation techniques. In order to avoid errors induced by rounding, the technique of outward rounding is used when working on intervals. Hybrid formulations with both integer and real variables are possible.

4.8. Prolog IV

Prolog IV, being a full extension of the linear constraint-solving capabilities of Prolog III [15], is additionally able to solve non-linear constraints using an interval constraint solver. A particular and remarkable feature is the built-in possibility to compute

with unions of intervals instead of simple intervals. For a detailed description of the differences between versions III and IV, see [45].

5. Conclusion and bibliographical remarks

This text has presented techniques for solving non-linear constraints over the Reals using methods that are based on interval arithmetic. We pointed out the advantages of this paradigm in comparison to existing approaches, demonstrated consistency techniques and presented a selection of implementations.

We would like to close with a few references for further reading. In particular, we recommend the recent, detailed and comprehensive COCONUT survey [10] on the state of the art approaches for solving nonlinear constrained and optimisation problems, covering a wide range of local and complete techniques. If the interest is mainly in complete techniques (from a global optimisation point of view), the extensive survey [46] should be consulted. There is furthermore a comprehensive online archive for global optimisation, which provides a helpful and large collection of literature, references and many further pointers.[17] General implementation aspects of interval arithmetic using floating-point intervals with regard to mathematical properties and computational correctness results can be found in [30].

An interesting concept is the use of *solver cooperation* on non-linear problems. Benhamou has developed an abstract framework for describing the cooperation of constraint solvers in [5], using domain reduction functions and constraint rewriting operators. Based on this framework, Granvilliers and Monfroy introduce a parameterized propagation algorithm in [24], which can be instantiated to known and parallel algorithm instances. Monfroy presents the CoSAc architecture for solving non-linear polynomial constraints in [43]. CoSAc is based upon the cooperation of three independent solvers: a linear solver in CHR, a non-linear solver computing Gröbner bases and the Maple computer algebra system for symbolic manipulations. The system is controlled by a constraint and communication manager implemented in ECLiPSe. For recent surveys on solver cooperation involving symbolic and interval-based approaches, see [25] and [10, chap. 6].

References

[1] K. R. APT: From Chaotic Iteration to Constraint Propagation. *Proc. 24th Int. Colloquium on Automata, Languages and Programming (ICALP'97)*, P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, (Eds), **1256** LNCS, Springer-Verlag, (1997), 36-55.

[17] `http://www.mat.univie.ac.at/ neum/glopt.html`

[2] K. R. APT: Principles of Constraint Programming. Cambridge Univ. Press, 2003.

[3] R. BARTÁK: Constraint Programming: In Pursuit of the Holy Grail. *Proc. WDS99 (invited lecture),* Prague, (1999).

[4] F. BENHAMOU: Interval Constraint Logic Programming. *Constraint Programming: Basics and Trends, Châtillon Spring School, Châtillon-sur-Seine*, France, 1994, Selected Papers, A. Podelski, (Ed), **910** LNCS, Springer, (1995), 1-21.

[5] F. BENHAMOU Heterogeneous Constraint Solving. *Proc. ALP'96, Algebraic and Logic Programming, 5th Int. Conf. Aachen, Germany,* M. Hanus and M. Rodr"guez-Artalejo, (Eds), **1139** LNCS, Springer, (1996), 62-76.

[6] F. BENHAMOU, P. CODOGNET, D. DIAZ, F. GOUALARD, and L. GRANVILLIERS: DecLIC 1.1b: User's Guide and Reference Manual. INRIA-Rocquencourt / LIFO-Orléans / IRIN-Nantes, 2003.

[7] F. BENHAMOU, F. GOUALARD, L. GRANVILLIERS, and J.-F. PUGET: Revising Hull and Box Consistency. *Proc. ICLP'99*, D. D. Schreye, (Ed), MIT Press, (1999), 230-244.

[8] F. BENHAMOU, D. MCALLESTER and P. V. HENTENRYCK: CLP(Intervals) Revisited. *Proc. 1994 Int. Symp. on Logic Programming (ILPS-94),* M. Bruynooghe, (Ed), MIT Press, (1994), 124-138.

[9] F. BENHAMOU and W. J. OLDER, Applying Interval Arithmetic to Real, Integer, and Boolean Constraints. *J. Logic Programming*, **32**(1) (1997), 1-24.

[10] C. BLIEK, P. SPELLUCCI, L. N. VICENTE, A. NEUMAIER, L. GRANVILLIERS, E. MONFROY, F. BENHAMOU, E. HUENS, P. V. HENTENRYCK, D. SAMHAROUD and B. FALTINGS: Algorithms for Solving Nonlinear Constrained and Optimization Problems: The State of the Art. Project Report D1, COCONUT Project, 2001.

[11] P. BRISSET, H. E. SAKKOUT, T. FRÜHWIRTH, C. GERVET, W. HARVEY, M. MEIER, S. NOVELLO, T. L. PROVOST, J. SCHIMPF, K. SHEN, and M. WALLACE: ECLiPSe Constraint Library Manual, release 5.8 ed. International Computers Limited and Imperial College, London, October 2004.

[12] B. D. BUNDAY: Basic Linear Programming. Edward Arnold Publishers Ltd., 1984.

[13] P. CODOGNET and D. DIAZ: Compiling Constraints in clp(FD). *J. Logic Programming*, **27**(3) (1996), 185-226.

[14] H. COLLAVIZZA, F. DELOBEL and M. RUEHER: Comparing Partial Consistencies. *Reliable Computing*, **5**(3) (1999), 213-228.

[15] A. COLMERAUER: An Introduction to Prolog III. *CACM 33*, (1990), 69-90.

[16] A. COLMERAUER: Naive Solving of Non-Linear Constraints. *Constraint Logic Programming: Selected Research*, F. Benhamou and A. Colmerauer, (Eds), MIT Press, (1993), 89-112.

[17] D. DIAZ: GNU Prolog 1.2.16 User Manual, 1.7 ed. Free Software Foundation, (2002).

[18] E. C. FREUDER: Synthesizing Constraint Expressions. *Communications of the ACM*, **21**(11) (1978), 958-966.

[19] D. GOLDBERG: What Every Computer Scientist Should Know About Floating-Point Arithmetic. *ACM Computing Surveys (CSUR)*, **23**(1) (1991), 5-48.

[20] F. GOUALARD: Towards Good C++ Interval Libraries: Tricks and Traits. *Proc. 4th Asian Symposium on Computer Mathematics (ASCM), Thailand,* (2000).

[21] F. GOUALARD, F. BENHAMOU and L. GRANVILLIERS: An Extension of the WAM for Hybrid Interval Solvers. *J. Functional and Logic Programming,* (1999).

[22] L. GRANVILLIERS: On the Combination of Interval Constraint Solvers. *Reliable Computing*, **7**(6) (2001), 467-483.

[23] L. GRANVILLIERS: RealPaver User's Manual: Solving Nonlinear Constraints by Interval Computations, for RealPaver Version 0.3. Institut de Recherche en Informatique de Nantes, France, 2003.

[24] L. GRANVILLIERS and E. MONFROY: Declarative Modelling of Constraint Propagation Strategies. *Proc. First Biennial Int. Conf. on Advances in Information Systems*, Turkey, T. M. Yakhno, (Ed), **1909** LNCS, Springer, (2000), 201-215.

[25] L. GRANVILLIERS, E. MONFROY and F. BENHAMOU: Symbolic-Interval Cooperation in Constraint Programming. *Proc. Int. Sym. Symbolic and Algebraic Computation (ISSAC'2001),* ACM Press, (2001), 150-166.

[26] N. C. HEINTZE, J. JAFFAR, S. MICHAYLOV, P. J. STUCKEY and R.H.C. YAP: The CLP(R) Programmer's Manual, Version 1.2. IBM Thomas J. Watson Research Center (Yorktown Heights, NY, USA), 1992.

[27] P. V. HENTENRYCK, D. MCALLESTER and D. KAPUR: Solving Polynomial Systems Using a Branch and Prune Approach. *SIAM J. Numerical Analysis*, **34**(2) (1997), 797-827.

[28] P. V. HENTENRYCK, L. MICHEL and F. BENHAMOU: Newton - Constraint Programming over Nonlinear Constraints. *Science of Computer Programming*, **30**(1-2) (1998), 83-118.

[29] P. V. HENTENRYCK, L. MICHEL and Y. DEVILLE: A Modeling Language for Global Optimization. MIT Press, 1997.

[30] T. HICKEY, Q. JU and M. H. VAN EMDEN: Interval Arithmetic: From Principles to Implementation. *Journal of the ACM (JACM)*, **48**(5) (2001), 1038-1068.

[31] T. J. HICKEY: Analytic Constraint Solving and Interval Arithmetic. *Proc. 27th ACM SIGPLAN-SIGACT Symp. Principles Of Programming Languages,* (2000), 338-351.

[32] T. J. HICKEY: CLIP: A CLP(Intervals) Dialect for Metalevel Constraint Solving. *Proc. Second Int. Workshop on Practical Aspects of Declarative Languages*, Boston, MA, USA, E. Pontelli and V. S. Costa, (Eds), **1753** LNCS, Springer, (2000), 200–214.

[33] T. J. HICKEY, M. H. VAN EMDEN and H. WU: A Unified Framework for Interval Constraints and Interval Arithmetic. *Proc. Principles and Practice of Constraint Programming - CP98, 4th Int. Conf.*, Pisa, Italy, M. J. Maher and J.-F. Puget, (Eds), **1520** LNCS, Springer, (1998), 250-264.

[34] J. HOLLMAN and L. LANGEMYR: Algorithms for Non-linear Algebraic Constraints. *Constraint Logic Programming: Selected Research*, F. Benhamou and A. Colmerauer, (Eds), MIT Press, (1993), 113-131.

[35] H. HONG: RISC-CLP(Real): Logic Programming with Non-linear Constraints over the Reals. *Constraint Logic Programming: Selected Research*, F. Benhamou and A. Colmerauer, (Eds), MIT Press, (1993), 133-159.

[36] E. HYVÖNEN: Constraint Reasoning Based on Interval Arithmetic. *Proc. 11th Int. Joint Conf. on Artificial Intelligence (IJCAI-89),* (1989), N. S. Sridharan, (Ed), **2**, Morgan Kaufmann, 1193-1198.

[37] ILOG. Ilog Solver 6.0, User's Manual. France, 2003.

[38] J. JAFFAR, S. MICHAYLOV, P. J. STUCKEY and R.H.C. YAP: The CLP(R) Language and System. *ACM Trans. Programming Languages and Systems (TOPLAS)*, **14**(3) (1992), 339-395.

[39] O. LHOMME: Consistency Techniques for Numeric CSPs. *Proc. 13th Int. Joint Conf. on Artificial Intelligence (IJCAI-93)*, Chambéry, France, R. Bajcsy, (Ed), **1**, Morgan Kaufmann, (1993), 232-238.

[40] A. K. MACKWORTH: Consistency in Networks of Relations. *Artificial Intelligence*, **8**(1) (1977), 99-118.

[41] L. MICHEL and P. V. HENTENRYCK: Helios: A modeling language for global optimization and its implementation in Newton. *Theoretical Computer Science*, 173(1) (1997), 3-48.

[42] E. MONFROY: Gröbner Bases: Strategies and Applications. *Proc. First Int. Conf. on Artificial Intelligence and Symbolic Mathematical Computation AISMC-1*, Karlsruhe, Germany, (1992), J. Calmet and J. A. Campbell, (Eds), **737** LNCS, Springer, (1993), 133-151.

[43] E. MONFROY, M. RUSINOWITCH and R. SCHOTT: Implementing Non-Linear Constraints with Cooperative Solvers. *Proc. ACM Symposium on Applied Computing (SAC-96),* , K. M. George, J. H. Carroll, D. Oppenheim, and J. Hightower, (Eds), ACM Press, (1996), 63-72.

[44] R. E. MOORE: Interval Analysis. Series in Automatic Computation, Prentice-Hall, 1966.

[45] G. A. NARBONI: From Prolog III to Prolog IV: The Logic of Constraint Programming Revisited. *CONSTRAINTS*, **4**(4) (1999), 313-335.

[46] A. NEUMAIER: Complete Search In Continuous Global Optimization And Constraint Satisfaction. *Acta Numerica*, **13** (2004), 271-369.

[47] A. NEUMAIER and O. SHCHERBINA: Safe bounds in linear and mixed-integer programming. *Mathematical Programming*, 99(2) (2004), 283-296.

[48] W. OLDER and F. BENHAMOU: Programming in CLP(BNR). *Proc. PPCP 1993: Newport, Rhode Island,* (1993), 228-238.

[49] J.-F. PUGET and P. V. HENTENRYCK: A Constraint Satisfaction Approach to a Circuit Design Problem. *J. Global Optimization*, **13**(1) (1998), 75-93.

[50] K. SAKAI and A. AIBA: CAL: A Theoretical Background of Constraint Logic Programming and its Applications. *J. Symbolic Computation*, **8**(6) (1989), 589-603.

[51] SUN MICROSYSTEMS. Interval Arithmetic Solves Nonlinear Problems While Providing Guaranteed Results. Sun Feature Stories, April 2001. `http://wwws.sun.com/software/sundev/news/features/intervals.html`.

[52] A. TARSKI: A Decision Method for Elementary Algebra and Geometry. second revised ed. University of California Press: Berkeley & Los Angeles, (1951), p. iii. 63ff.

[53] M. H. VAN EMDEN: Algorithmic Power from Declarative Use of Redundant Constraints. *CONSTRAINTS*, **4**(4) (1999), 363-381.