From Interval Arithmetic to Interval Constraints*

M.H. van Emden
Department of Computer Science
, University of Victoria, Canada
vanemden@csr.uvic.ca

Abstract

Two definitions of division by an interval containing zero are compared: a functional one and a relational one. We show that the relational definition also provides interval inverses for other functions that do not have point inverses, such as *max* and the absolute-value function. Applying the same approach to the $\leq$ relation introduces the "companion functions" of relations. By regarding the arithmetic operations $+$, $-$, $*$, and $/$ as ternary relations, we obtain the interval versions of the operations. This opens the way for regarding arithmetic problems such as evaluating expressions and solving equations as Constraint Satisfaction Problems (CSPs). These have a useful computational theory, which is, however, influenced by their predominantly discrete applications. We generalize the conventional formulation to better accommodate real-valued variables, and state the main results. When these results are applied to numerical CSPs we relate the interval evaluation of an arithmetic expression to the family of solving algorithms of CSPs. The key to our method of bringing interval arithmetic and interval constraints under a common denominator are *companion functions*. These functions form an alternative characterization of n-ary relations and appear to be a new contribution to the mathematical theory of relations.

Keywords: Constraint satisfaction problems, companion functions, interval arithmetic, interval constraints.
AMS subject classifications: 65G30

1 Introduction

Analysis is primarily concerned with real-valued functions and only secondarily with relations among the reals. The functions are defined by expressions that almost always include the basic arithmetic operations. Interval analysis allows one to accommodate to the limitations of computers by extending the operations on the reals to the analogous ones on sets of reals. For practical reasons these sets are restricted to intervals. These are computed in such a way as to ensure that all reals are included that *may* have been the result of the infinite-precision version of the operation.

*Submitted: January 23, 2009; Accepted: January 22, 2010.

Another way of stating the "*may* have been" is to say that interval arithmetic yields intervals that exclude with *certainty* reals that are *not* results of the infinite-precision version of the operation. This manner of *ex*cluding what is *not* part of a solution is the essence of constraint processing, which is computation with Constraint Satisfaction Problems (CSPs). This is a general paradigm, of which the branch dealing with real-valued variables is somewhat neglected. This branch only became feasible with interval arithmetic and IEEE-standard floating-point arithmetic.

This paper aims to show the advantages of regarding interval arithmetic as a special case of constraint processing. It does this by starting within interval arithmetic and, as it were, discovering constraint processing in response to difficulties within interval arithmetic.

2 Interval division

Moore [12] defined exact-interval arithmetic as

$$[a,b] \circ [c,d] \stackrel{\text{def}}{=} \{x \circ y \mid a \leq x \leq b \wedge c \leq y \leq d\} \tag{1}$$

with $\circ$ being one of the symbols $+$, $-$, $*$, or $/$. He added: "except that we do not define $[a,b]/[c,d]$ if $0 \in [c,d]$".

In the practice of computation with reals the fact that, for reals x and y, x/y is only defined for $y \neq 0$ is a problem, but it is a manageable problem. Restricting $[a,b]/[c,d]$ to $0 \notin [c,d]$ may be acceptable when one is only intervalizing conventional algorithms. But interval arithmetic is important for the new algorithms that it makes possible, such as Interval Newton and Moore-Skelboe. Such use is severely handicapped by requiring that divisor intervals exclude 0. Kahan [11] proposed an "extended interval arithmetic" that allows division by an interval containing zero. This can be defined in several ways.

One approach is:

Definition 1

$$[a,b]/[c,d] \stackrel{\text{def}}{=} \square\{x/y \mid x \in [a,b] \wedge y \in [c,d]\}$$

for all *intervals* $[a,b]$ *and* $[c,d]$.

Here $\square$ is the function that maps any set of reals to the least interval containing it. This is found, for example, in [8]. The reasoning behind it is: x/y being undefined (when $y = 0$) means that it does not exist; if it does not exist, then it does give rise to any contribution to the set being defined; therefore, the set is well-defined. The validity of this approach depends on undefinedness implying non-existence. This is not always accepted. For example, undefined could give rise to a special value outside the reals, as is done in the IEEE floating-point standard, where NaN is such an extraneous value.

Ratz proposed a definition of division by an interval regardless of whether and how it contains a zero. He pointed out that

$$\{y \in \mathbf{R} \mid \exists z \in [a,b], x \in [c,d] \,.\, x * y = z\} \tag{2}$$

coincides with (1) when $\circ$ is $/$ and $0 \notin [c,d]$. But (2) is also defined when $0 \in [c,d]$, although the set may in that case not be an interval. So Ratz defined [13]

Definition 2 $[a,b]/[c,d] \stackrel{\text{def}}{=} \square\{y \in \mathbf{R} \mid \exists z \in [a,b], x \in [c,d] \,.\, x * y = z\}$

The definitions differ for the values they give for $[0,0]/[0,0]$: $\emptyset$ and $\mathbf{R}$ for definitions 1 and 2, respectively. The reason behind the correctness properties of interval arithmetic is that only those reals are removed from intervals that have been shown not to be possible as values of the unknown. Accordingly, adopting the relational Definition 2 ensures that no interval arithmetic applications are excluded. Moreover, the relational approach opens up interesting extensions of interval arithmetic, which we will examine further on in this paper.

Note that $\square$ can mean "smallest real interval containing the argument set" or "smallest floating-point interval containing the argument set". In the latter case, Definition 2 defines all there is to define about interval division, as it takes outward rounding into account as well. How to compute the bounds of the result interval on the basis of the bounds of the argument intervals requires a detailed case analysis, such as found in [8].

3 Other applications of relational form

All four interval arithmetic operations could have been expressed in relational form. We start by defining the ternary relations:

$$\begin{aligned} sum &\stackrel{\text{def}}{=} \{\langle x,y,z\rangle \in \mathbf{R}^3 \mid x + y = z\} \\ prod &\stackrel{\text{def}}{=} \{\langle x,y,z\rangle \in \mathbf{R}^3 \mid x * y = z\} \end{aligned}$$

We have by analogy with Definition 2 for all four interval operations:

$$\begin{aligned} [a,b] + [c,d] &\stackrel{\text{def}}{=} \square\{z \in \mathbf{R} \mid \exists x \in [a,b], y \in [c,d] \,.\, \langle x,y,z\rangle \in sum\} \\ [a,b] * [c,d] &\stackrel{\text{def}}{=} \square\{z \in \mathbf{R} \mid \exists x \in [a,b], y \in [c,d] \,.\, \langle x,y,z\rangle \in prod\} \\ [a,b] - [c,d] &\stackrel{\text{def}}{=} \square\{y \in \mathbf{R} \mid \exists z \in [a,b], x \in [c,d] \,.\, \langle x,y,z\rangle \in sum\} \\ [a,b]/[c,d] &\stackrel{\text{def}}{=} \square\{y \in \mathbf{R} \mid \exists z \in [a,b], x \in [c,d] \,.\, \langle x,y,z\rangle \in prod\} \end{aligned} \tag{3}$$

The function $\square$ is added to all definitions for the sake of uniformity when it is interpreted as the least real interval containing its argument; it is of course only necessary for division, and then only for the case $0 \in [c,d]$. When $\square$ is interpreted as the least floating-point interval containing its argument it is necessary for all four definitions, and it implies outward rounding.

The relational method of Ratz was originally used because of an inverse not being everywhere defined. It can also be used when the inverse is multivalued. Consider the function $\max : \mathbf{R}^2 \to \mathbf{R}$ that has as value the greater of its two arguments or the value of both arguments when they are equal. This function fails to have an inverse because it would be undefined at some points (e.g. for what x do we have $\max(x,1) = 0$?) and multivalued at other points (e.g. for what x do we have $\max(x,1) = 1$?). By analogy with Definition 2 we get for the inverse of max among intervals:

Definition 3

$$max^{-1}([a,b],[c,d]) = \{y \in \mathbf{R} \mid \exists z \in [a,b], x \in [c,d].\ \max(x,y) = z\}$$

To compute this function we can use

Theorem 1

$$max^{-1}([a,b],[c,d]) = \begin{cases} \text{if } d < a\text{:} & \emptyset \\ \text{if } d \geq a \text{ and } b \geq c\text{:} & [-\infty, d] \\ \text{if } b < c\text{:} & [c,d] \end{cases}$$

Let us next consider the absolute-value function such that abs(x) is x if x is positive and $-x$ otherwise. The inverse is undefined or multivalued. With Ratz's relational definition we have a well-defined interval inverse:

Definition 4

$$\text{abs}^{-1}([a,b]) = \square\{x \in \mathbf{R} \mid \exists y \in [a,b].\ \text{abs}(x) = y\} \tag{4}$$

Here $\square$ is needed to ensure an interval result when $0 < a$.

To compute this inverse we can use:

Theorem 2

$$\begin{aligned} \text{abs}^{-1}([a,b]) &= [-b,b] \text{ if } b \geq 0 \\ &= \emptyset \text{ if } b < 0 \end{aligned}$$

So far we have shown that the relational method is successful for obtaining interval inverses in cases where the real-valued function does not have an inverse. The method depends on translating the function to a relation. Let's see what happens if we apply the same method in a case where we do not start out with a function at all: the not-greater-than relation among reals. Below we define an interval function analogously to (4), replacing abs(x) = y by $x \leq y$:

$$f([c,d]) = \{x \in \mathbf{R} \mid \exists y \in [c,d].\ x \leq y\} = [-\infty, d]$$

Analogously to the usual notion of abs($[a,b]$) we have

$$g([a,b]) = \{y \in \mathbf{R} \mid \exists x \in [a,b].\ x \leq y\} = [a, \infty]$$

Due to the nature of the relation involved, the $\square$ operation would not have an effect in either case.

The functions f and g represent two ways of associating an interval function with an arbitrary binary relation, one that does not need to be a function. These functions are examples of the *companion functions* for a binary relation. A general definition will be given in Section 5.2.

The companion functions for this particular binary relation cast light on the question that sometimes asked by newcomers to interval arithmetic: What is the truth value of $[a,b] \leq [c,d]$? If $b < c$ or if $d < a$ the answer is clear. The lack of obviousness in the remaining case indicates that it is not helpful to think of intervals as a kind of generalized number, as in [12].

What is a better way to think about intervals? Let's retreat to familiar ground and consider the addition operator. Regarding an interval as a generalized number suggests defining $[a,b] + [c,d]$ by asking

What is the set of all $x + y$ when $x \in [a,b]$ and $y \in [c,d]$?

But we can ask instead:

> Is $x + y = z$ possible when $x \in [a, b]$ and $y \in [c, d]$? And if so, what are the possible values of z?

The first part of this question is not interesting. But we can ask, analogously:

> Is $x \leq y$ possible when $x \in [a, b]$ and $y \in [c, d]$? And if so, what are the possible values for x and y?

Here we regard $x \leq y$ as a *constraint* to be imposed on the values of x and y that are known to be *a priori* in $[a, b]$ and $[c, d]$. The constraint reduces these possible values to $x \in ([a, b] \cap f([c, d]))$ and $y \in ([c, d] \cap g([a, b]))$. Or, as in the following theorem, where the companion functions f and g have had their definitions substituted:

Theorem 3

$$\begin{aligned} leq \cap ([a, b] \times [c, d]) &= \emptyset \text{ if } d < a \\ &= [a, d] \times [a, d] \text{ if } c \leq a \leq d \leq b \\ &= [a, b] \times [a, d] \text{ if } b \leq d \wedge c \leq a \\ &= [c, b] \times [c, b] \text{ if } a \leq c \leq b \leq d \\ &= [a, d] \times [c, d] \text{ if } a \leq c \wedge d \leq b \\ &= [a, b] \times [c, d] \text{ if } b \leq c \end{aligned}$$

where $leq = \{\langle x, y\rangle \in \mathbf{R}^2 \mid x \leq y\}$.

4 Arithmetic constraints

In (3) we saw a set-theoretic expression for interval division in relational form. For better understanding it helps to convert it to algebraic form:

$$\begin{aligned} & \{y \in \mathbf{R} \mid \exists x \in [a, b], z \in [c, d] \,.\, \langle x, y, z\rangle \in prod\} \\ = \quad & \pi_y(\{\langle x, y, z\rangle \in \mathbf{R}^3 \mid x \in [a, b], z \in [c, d] \,.\, \langle x, y, z\rangle \in prod\}) \end{aligned}$$

where $prod = \{\langle x, y, z\rangle \in \mathbf{R}^3 \mid x * y = z\}$.

This is an example of a companion function — this time for the ternary *prod* relation. A general definition will be given in Section 5.2.

This is but one of three symmetrical formulas for the projection. Putting all three together we have:

$$\begin{aligned} Z/Y &= \pi_x(\Box(prod \cap (\mathbf{R} \times Y \times Z))) \\ Z/X &= \pi_y(\Box(prod \cap (X \times \mathbf{R} \times Z))) \\ X * Y &= \pi_z(\Box(prod \cap (X \times Y \times \mathbf{R}))) \end{aligned} \tag{5}$$

Suppose that all candidates for solutions are in $X \times Y \times Z$, but that the constraint $prod \subseteq \mathbf{R}^3$ has not yet been taken into account. Then we can restrict the candidates for solutions to $prod \cap (X \times Y \times Z)$. Because this intersection is typically difficult to

represent we store $\square(prod \cap (X \times Y \times Z))$. The function $(X \times Y \times Z) \mapsto \square(prod \cap (X \times Y \times Z))$ we call the *domain restriction operator* (DRO) of *prod*, denoted γ_{prod}.

The equalities in (5) show that the DRO can be computed by means of interval arithmetic. In fact, if we already have *a priori* intervals X, Y, and Z, then we can modify (5) so that we get the projections of $\square(prod \cap (X \times Y \times Z))$:

$$\begin{aligned} X \cap Z/Y &= \pi_x(\square(prod \cap (X \times Y \times Z))) \\ Y \cap Z/X &= \pi_y(\square(prod \cap (X \times Y \times Z))) \\ Z \cap X * Y &= \pi_z(\square(prod \cap (X \times Y \times Z))) \end{aligned} \tag{6}$$

The interval arithmetic operations are special cases of applying the DRO. For example, we have $\pi_y(\gamma_{prod}(X \times \mathbf{R} \times Z)) = Z/X$. In fact, for interval Y sufficiently wide compared to intervals X and Z, we have $\pi_y(\square(\gamma_{prod}(X \times Y \times Z))) = Z/X$.

In [9] similar characterizations were introduced for the relational versions of sum, prod, integral power, $\leq$, and $=$.

Benhamou and Older [2] introduced ternary versions of interval operations: $X \cap Z/Y$ instead of Z/Y, $Y \cap Z/X$ instead of Z/X, and $Z \cap X * Y$ instead of $X * Y$.

Example Let $X = [-0.5, +0.5]$, $Y = [-1, +1]$, and $Z = [+1, +1]$. Note that

$$\pi_x(\square(\gamma_{prod}(X \times Y \times Z))) = X \cap Z/Y = \emptyset.$$

This is to be compared with Definitions 1 and 2, which give

$$Z/Y = \square([-\infty, -1] \cup [1, +\infty]) = [-\infty, +\infty].$$

Thus the DRO gives more information than interval division.

5 Constraints in a general setting

So far we have only seen how the individual interval arithmetic operations generalize to arithmetic constraints. The evaluation of complex expressions in interval arithmetic also generalizes to interval constraints. To see this we need to move from binary and ternary constraints to the n-ary case. While we generalize this, we might as well lift the restriction from reals and intervals to allow other types of values.

5.1 Types, domains, and domain systems

We are going to consider relations over n variables. A common numerical example of such a constraint takes the form $f(x_0, \ldots, x_{n-1}) = 0$. With f a given function, this defines an n-ary relation constraining the values of $x_0, \ldots, x_{n-1}$. In the general case, not necessarily numerical, we allow for the possibility that each of the variables takes its value from a set independently of the others. Thus we have sets $T_0, \ldots, T_{n-1}$, which we call *types*, not necessarily different, from which the variables take their values.

We will use constraints to obtain information about the variables. We do this by finding as small as possible subsets of the types to which the variables can be restricted. Computational realities often force us to limit the variety of sets of values under consideration. When the type is $\mathbf{R}$, these subsets are restricted to intervals. For

other types it may also be necessary to similarly restrict the subsets of the type under consideration. We refer to such allowed subsets of a type as *domains*.

We call a *domain system* for a set S a set of subsets of S that has S itself as element and is closed under intersection. A trivial example is $\mathcal{P}(S)$; this is typically used when S is finite, and small. To model real-valued variables in a computer we need a non-trivial domain system, and this is provided by the set of floating-point intervals. Note that the inclusion of the infinities among the floating-point numbers supports the requirement that $\mathbf{R}$ itself belong to the domain system.

5.2 The companion functions of an n-ary relation

Let $r \subseteq T_0 \times \cdots \times T_{n-1}$ be an n-ary relation. Let $\mathcal{D}(T_i)$ be a domain system for T_i, for $i = 0, \ldots, n-1$. The *companion functions* $r_0, \ldots, r_{n-1}$ of r have the type $r_i : \mathcal{D}(T_0) \times \cdots \times \mathcal{D}(T_{n-1}) \to \mathcal{D}(T_i)$. The DRO of r has the type

$$\gamma_r : \mathcal{D}(T_0) \times \cdots \times \mathcal{D}(T_{n-1}) \to \mathcal{D}(T_0) \times \cdots \times \mathcal{D}(T_{n-1}).$$

Both are defined as:

Definition 5

$$\begin{array}{ll} & r_i(D_0, \ldots, D_{n-1}) \\ \stackrel{\text{def}}{=} & \pi_i(r \cap (D_0 \times \cdots \times D_{i-1} \times T_i \times D_{i+1} \times \cdots \times D_{n-1})) \\ = & \{x_i \in T_i \mid \exists_{j \in (n \ominus i)} x_j \in D_j \,.\, r(x_0, \ldots, x_{n-1})\}, \end{array}$$

where $n \ominus i = \{0, \ldots, n-1\} \backslash \{i\}$.

The DRO *of* r *is defined by its projections; for* $i = 0, \ldots, n-1$*:*

$$\pi_i(\gamma_r(D_0, \ldots, D_{n-1})) = \square(D_i \cap r_i(D_0, \ldots, D_{n-1}))$$

where $\square$ *is the smallest domain product containing its argument.*

Theorem 4 γ_r *is* non-increasing*:*

$$\pi_i(\gamma_r(D_0, \ldots, D_{n-1})) \subseteq D_i$$

for $i = 0, \ldots, n-1$*, and it is* idempotent*:*

$$\pi_i(\gamma_r(\gamma_r(D_0, \ldots, D_{n-1})) = \pi_i(\gamma_r(D_0, \ldots, D_{n-1}))$$

for $i = 0, \ldots, n-1$.

Example Let $n = 3$, T_i all equal to $\mathbf{R}$, $\mathcal{D}(T_i)$ all equal to the set of intervals, and $r = \{\langle x, y, z\rangle \in \mathbf{R}^3 \mid x * y = z\}$. The three projections of $\gamma_r(D_0, D_1, D_2)$ are given by (6) if D_0, D_1, D_2 are X, Y, Z, respectively.

Example This will be familiar to Sudoku puzzlers. Let $n = 3$, T_i all equal to $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, $\mathcal{D}(T_i)$ all equal to $\mathcal{P}(T_i)$. The rules of the puzzle require certain combinations of variables to have different values. Suppose we have *a priori* $x \in \{3, 5, 9\}$ and y and z both in $\{3, 9\}$. Then the DRO for the constraint that requires the values of these variables to be all different reduces the domain for x to $\{5\}$ without causing any change in the domains for y and z.

6 Constraint Satisfaction Problems

We introduced the notion of constraint with the $\leq$ relation, then extended it to other binary relations and to ternary relations. Most recently we considered n-ary relations as constraints. Now we turn to multiple constraints to be satisfied simultaneously.

Waltz [14] introduced a new computational paradigm: he generalized the familiar systems of equations between numerical expressions to include other relations that constrain the values that variables are allowed to assume. His motivation for the generalization was that he needed to solve a certain type of combinatorial problem expressed by means of non-numeric constraints. The characteristics of these problems are: a large number of variables, a large number of constraints, and small finite sets of values over which the variables can range. The constraints involve only small subsets of the set of all variables, yet the constraints typically share at least one variable with one or more other constraints. In case such a constraint system would consist of numerical equations or inequalities, it would be called "sparse".

The solution method adopted by Waltz was to associate with each variable a set of values (the variable's "domain"). These values include by default all conceivable ones. By examining each constraint separately, some values in the domain of one or more of its variables may be found not to satisfy the constraint. Such values are removed.

This process terminates because of the finiteness of all entities involved. As only values have been removed that cannot be part of a solution, at every stage the set of solutions is contained in the Cartesian product of the domains. At termination the following possibilities exist:

1. A domain is empty; this implies that there is no solution.
2. The terminal Cartesian product contains only a single tuple. There is a unique solution, and this is the one.
3. The terminal Cartesian product contains more than one tuple. None of these may be a solution. But possibly existing solutions have to be among these. Whether this is the case has to be established by enumeration.

Since Waltz's work much research has been done in CSPs. As this was mostly concerned with finite domains, the conventional formalization [1] is suitable for this situation. The application of constraint solving to numerical problems [7] has not followed this convention. The conventional formulation has the great advantage that useful results have been obtained for it. To be able to exploit these, we generalize the conventional formalization to accommodate reals as type, without excluding the other kinds of CSP.

Definition 6 *A Constraint Satisfaction Problem (*CSP*) consists of a finite set* $\mathcal{X} = \{x_0, \ldots, x_{n-1}\}$ *of variables, a set* $\boldsymbol{C} = \{C_0, \ldots, C_{m-1}\}$ *of constraints, each of which is a relation over a sequence of elements of* $\mathcal{X}$ *and has an associated* DRO*. With each variable* x_i *is associated a type* T_i*, which is the set of values that* x_i *can assume. For each type there is a domain system. A* solution *of a* CSP *is an assignment to each variable* x_i *of an element of* T_i *such that each constraint in* $\boldsymbol{C}$ *is satisfied.*

Theorem 5 *The solution set equals* $C_0 \bowtie \cdots \bowtie C_{m-1}$ *where* $\bowtie$ *is the natural join in the sense of relational database theory.*

Proof $\{C_0, \ldots, C_{m-1}\}$ can be translated to an existentially quantified conjunction of atomic formulas of first-order predicate logic. According to [6] an atomic formula

denotes a cylinder in the space of all variables, while the conjunction denotes the intersection of these cylinders. The "natural join" in the sense of database theory can be characterized as the intersection of such cylinders□

Definition 7 *A* computation state *of a* CSP *is* $D_0 \times \cdots \times D_{n-1}$ *where* $D_i \subseteq T_i$ *is a domain and is associated with* x_i*, for* $i = 0, \ldots, n-1$.
A computation *of a* CSP *is a sequence of computation states in which each (after the initial one) is obtained from the previous one by the application of the* DRO *of one of the constraints of the* CSP.
The limit *of a computation is the intersection of its states.*
A fair computation *of a* CSP *is a computation in which each of the constraints is represented by its* DRO *infinitely many times.*

Fair computations have infinite length. However, for all practical CSPs no change occurs from a certain point onward. By the idempotence of the DROs, this is detectable by algorithms that generate fair computations, so that they can terminate accordingly.

Theorem 6 *The limit of a fair computation of a* CSP *is equal to the intersection of the initial state of the computation with the greatest fixpoint common to all* DROs.

Proof A tuple that belongs to all states of the computation has been subjected to the DROs of all constraints. Therefore the intersection of all states is a fixpoint of all DROs. We next consider whether it is the greatest.

Let t be a tuple be a tuple not belonging to the intersection of all computation states, yet belonging to a fixpoint common to all DROs. As t is not in the intersection, it has been removed by the DRO of a constraint. Therefore t does not satisfy that constraint and does not belong to any common fixpoint. This contradiction shows that no such t can exist, and that the intersection of all computation states is the greatest common fixpoint of all DROs□

For a given CSP the intersection of the states of any fair computation only depends on the initial state. It is therefore independent of the computation itself. Apparently the CSP maps the set of Cartesian products to itself. It is a non-increasing, idempotent mapping.

Theorem 7 *Let* D *be the initial state of a fair computation of a* CSP. *Then the limit of the fair computation contains the intersection of* D *with the solution set.*

The limit is not, in general, the smallest box containing the intersection of D with the solution set.

6.1 Numerical CSPs

We are interested in CSPs with the following characteristics. The variables range over the reals; that is, all types $T_0, T_1, \ldots$ are equal to $\mathbf{R}$. The domain system is that of the floating-point intervals. The constraints include *sum*, *prod*, $\leq$, *max*, and *abs*. The reason is that these have DROs that are efficiently computable, as shown in this paper. DROs are also easily obtainable for $=$ and for rational powers. DROs for the constraints corresponding to the transcendental functions are not easy to obtain, as their definition requires them to be the least floating-point box containing the intersection of the relation with the argument box. But reduction operators closely approximating this ideal are used in some systems [7].

We call such instances *numerical* CSPs. They are interesting because many numerical problems take the form of mixtures of equalities and inequalities between arithmetical expressions that may contain transcendental functions as terms. Such problems can be translated to numerical CSPs.

Example Consider the equation $x(x-2)+1=y$, where x and y are real variables. It represents a constraint on the values of $x \in X$ and $y \in Y$. Constraint processing is not directly applicable here, as we only have available the constraints *sum* and *prod*. A logically equivalent form of $x(x-2)+1=y$ is

$$\exists z, u \in \mathbf{R}.\ sum(2, z, x) \wedge prod(x, z, u) \wedge sum(u, 1, y).$$

The equation is therefore translated to a CSP with the following set of constraints:

$$\boldsymbol{C} = \{sum(2, z, x), prod(x, z, u), sum(u, 1, y)\}$$

The initial computation state is specified by $x \in X, y \in Y, z \in [-\infty, +\infty], u \in [-\infty, +\infty]$

The end result depends on the initial intervals X and Y. They can be such that the limit of the CSP is empty. In such a case it has been proved that no solution has an x value in the initial X, and similarly for y. In other cases termination may occur with the intervals for x and y narrowed to a small box around a solution.

Let us now consider some special cases of this example. Suppose the initial interval for y is $[0, 0]$. In that case the CSP is used to approximate the solution set of $x(x-2)+1=0$. If the initial interval of x is large enough to contain both solutions, it will narrow to an interval containing both solutions. A sufficiently small interval around one of the solutions can be expected to narrow to an interval small enough to represent an accurate approximation.

If, on the other hand, the initial interval for y is $[-\infty, +\infty]$, then the CSP represents the task of evaluating $X(X-2)+1$ in interval arithmetic. If the constraints are selected in a suitable order, then none of the DROs needs to be applied more than once□

In general, for any expression E in variables $x_0, \ldots, x_{n-1}$, we can translate the equation $E = y$ to a numerical CSP with variables $x_0, \ldots, x_{n-1}, y$ with as initial domains the intervals $X_0, \ldots, X_{n-1}, Y$. If we set $Y := [-\infty, +\infty]$, it may be proved that the limit of the CSP has as projections $X_0, \ldots, X_{n-1}, E[x_0/X_0, \ldots, x_{n-1}/X_{n-1}]$, where this last expression is evaluated in interval arithmetic. Moreover, there is a computation where the limit is reached in the same number of steps as the number of subexpression evaluations needed for evaluating E.

This special case of $Y = [-\infty, +\infty]$ only serves to illustrate interval arithmetic as special case of constraint processing. More interesting is to start off with Y narrow enough to cause some of the $X_0, \ldots, X_{n-1}$ to narrow. In the extreme case of $Y = [0, 0]$ we have equation solving. In that case the limit is not the smallest box containing the solution set $C_0 \bowtie \cdots \bowtie C_{m-1}$; it is usually far from being the smallest box. See [7] for supplementary techniques that result in tighter enclosures.

7 Related work

We have emphasized a progression from interval arithmetic to interval constraints. Accordingly we took Ratz's relation definition of 1994 as starting point. But Cleary [4] and Davis [5] described DROs for the arithmetic constraints as found here in Section 4.

Ratz's definition is subsumed by the DRO for the product constraint. BNR Prolog [3] included DROs for the *max* and *abs* constraints; so used the equivalent of theorems 1 and 2. Although we are concerned here with earliest sources rather than a survey of the literature, mention should be made of the comprehensive [10], which treats both interval arithmetic and interval constraints.

8 Conclusions

For many observers the motivation for interval arithmetic is control of rounding errors. However, from the beginning [12] the field has been more than the invervalizing of conventional algorithms (which always lead to disappointingly wide intervals). Interval Newton was an early example of an algorithm that is much more powerful than its non-interval counterpart. Moore-Skelboe is an algorithm that does not even seem to have a non-interval counterpart.

It may seem that interval constraints is just another application of interval arithmetic. Conversely, interval arithmetic may seem just a special case of interval constraints. This paper is designed to suggest that there are opportunities for more interplay between the two interval methods; opportunities that are likely to be amply rewarded.

References

[1] K. R. Apt, *Principles of Constraint Programming*, Cambridge University Press, 2003.

[2] F. Benhamou, W. J. Older, Applying interval arithmetic to real, integer, and Boolean constraints, *Journal of Logic Programming*, Vol. 32, pp. 1–24, 1997.

[3] BNR, BNR *Prolog User Guide and Reference Manual*, Version 3.1 for Macintosh, 1988.

[4] J. G. Cleary, Logical arithmetic, *Future Computing Systems*, Vol. 2, pp. 125–149, 1987.

[5] E. Davis, Constraint propagation with interval labels, *Artificial Intelligence*, Vol. 32, pp. 281-331, 1987.

[6] L. Henkin, J. D. Monk, A. Tarski, *Cylindric Algebras, Parts I, II*, Studies in Logic and the Foundations of Mathematics, North-Holland, 1985.

[7] P. Van Hentenryck, L. Michel, Y. Deville, *Numerica: A Modeling Language for Global Optimization*, MIT Press, 1997.

[8] T. Hickey, Q. Ju, M.H. van Emden, Interval arithmetic: from principles to implementation, *Journal of the ACM*, Vol. 48, pp. 1038–1068. 2001.

[9] T. J. Hickey, M.H. van Emden, H. Wu, A unified framework for interval constraints and interval arithmetic, in: (Michael Maher and Jean-Fran**c**cois Puget, eds., *Principles and Practice of Constraint Programming — CP98*, pp. 250–264, Springer-Verlag, 1998, Lecture Notes in Computer Science 1520.

[10] L. Jaulin, M. Kieffer, O. Didrit, É. Walter, *Applied Interval Analysis*, Springer-Verlag, 2001.

[11] W. M. Kahan, A more complete interval arithmetic, Technical report, University of Toronto, Canada, 1968.

[12] R. E. Moore, *Interval Analysis*, Prentice-Hall, 1966.

[13] D. Ratz, On extended interval arithmetic and inclusion isotonicity, Technical report, Institut für Angewandte Mathematik, Universität Karlsruhe, 1996.

[14] D. Waltz, Understanding line drawings in scenes with shadows, in: Patrick Henry Winston, editor, *The Psychology of Computer Vision*, pp. 19–91. McGraw-Hill, 1975.