PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1992,
edited by ICOT. © ICOT, 1992

Adapting CLP($\mathcal{R}$) To Floating-Point Arithmetic

J.H.M. Lee and M.H. van Emden

Logic Programming Laboratory, Department of Computer Science
University of Victoria, Victoria, B.C., Canada V8W 3P6
{jlee,vanemden}@csr.uvic.ca

Abstract

As a logic programming language, Prolog has shortcomings. One of the most serious of these is in arithmetic. CLP($\mathcal{R}$) though a vast improvement, assumes perfect arithmetic on reals, an unrealistic requirement for computers, where there is strong pressure to use floating-point arithmetic. We present an adaptation of CLP($\mathcal{R}$) where the errors due to floating-point computation are absorbed by the use of intervals in such a way that the logical status of answers is not jeopardized. This system is based on Cleary's "squeezing" of floating-point intervals, modified to fit into Mackworth's general framework of the Constraint-Satisfaction Problem. Our partial implementation consists of a meta-interpreter executed by an existing CLP($\mathcal{R}$) system. All that stands in the way of correct answers involving real numbers is the planned addition of outward rounding to the current prototype.

1 Introduction

Mainstream computing holds that programming should be improved by gradual steps, as exemplified by the methods of structured programming and languages such as Pascal and Ada. Revolutionaries such as Patrick Hayes and Robert Kowalski advocated radical change, as embodied in Hayes's motto: "Computation is deduction." [Hayes 1973] According to this approach, programs are definitions in a declarative language and every computation step is a valid inference, so that results are logical consequences of program and data. Logic programming, Prolog and the CLP scheme are examples of this radical alternative in programming languages and method. Where Prolog coincides with logic programming, *certainty* of knowledge is obtained.

In numerical analysis there is a similar tension between mainstream thinking and the radicals. The first is satisfied without rigorous control of errors. When successive approximations differ by a small amount, it is assumed that a result has been obtained with an error of approximately that amount. Of course sophisticated error analyses can be made to suggest more certain knowledge. But such analyses are typically valid only asymptotically. In practice one does not know whether one is close enough to the true value for the asymptotic analysis to be applicable at all.

The radical alternative in numerical computation is represented by interval methods, where the ideal is to be sure that the true value is contained in an interval. It is then the purpose of iteration to shrink such an interval till it is no greater than an acceptable width. Here again the goal is *certainty* of knowledge.

In the research reported in this paper, we bring these two radical streams together. Both streams are, in their present form, deficient. Logic programming lacks in control of numerical errors. Interval methods rely on conventional algorithmic languages and hence lack computation as deduction. We show that the two can be combined in such a way that rigorously justified claims can be made about the error in numerical computation even if conventional floating-point arithmetic is used.

Problem statement. Logic programming, exemplified by Prolog, is the most successful realization of Hayes's motto. In certain application areas, Prolog can be used to program efficient computations that are also logical deductions. However, Prolog arithmetic primitives, which are functional in nature, are incompatible with the relational paradigm of logic programming. The advent of CLP($\Re$), an instance of the CLP scheme [Jaffar and Lassez 1987], takes us closer to relational arithmetic but its implementation CLP($\mathcal{R}$)[1] [Jaffar *et al.* 1990] is obtained by substituting each real number by a single floating-point approximation. As a result, round-off errors destroy soundness and disqualify CLP($\mathcal{R}$) computation as deduction.

Solution. Our solution consists of three parts. First, we tackle the round-off error problem with interval arithmetic introduced in [Moore 1966]. Instead of operating on individual floating-point numbers, interval arithmetic

[1]In this paper, we use CLP($\Re$) to denote the CLP instance with $\Re$ being the algebraic structure of finite trees of reals; and CLP($\mathcal{R}$) is the name of a CLP($\Re$) implementation.

manipulates intervals. The guaranteed inclusion property of interval arithmetic ensures the soundness of computation.

Second, traditional interval arithmetic is functional and has been embedded in functional or imperative languages. To develop the required relational version, we use an interval narrowing operation based on work in [Cleary 1987] and similar to the one used in [Sidebottom and Havens 1992].

Finally, we make a modification to the CLP scheme by including an operation that reduces goal to normal form and show that interval narrowing is such an operation. By modifying a meta-interpreter for CLP($\mathcal{R}$) accordingly, we obtain a prototype implementation.

The paper is organized as follows. Section 2 discusses related work. Relational interval arithmetic, which consists of interval narrowing and a relaxation algorithm, is presented in section 3. In section 4, we describe the semantics of ICLP($\mathcal{R}$), which is CLP($\mathcal{R}$) extended with relational interval arithmetic. We summarize and conclude in section 5.

2 Related Work

Interval arithmetic. While it is important to derive new and more efficient interval arithmetic algorithms and ensure delivery of practical interval bounds, recent development in interval arithmetic [Moore 1988] emphasizes: (1) automatic verification of computed answers, and (2) clear mathematical description of the problem. Users of numerical programs are usually only interested in the solution of a problem. They do not want to take the burden (a) to understand how the problem is solved, (b) to validate the correctness of the answers, and (c) to calculate error bounds. Logic programming shares these goals.

Constraint interval arithmetic. Constraint interval arithmetic stems from constraint propagation techniques. It is a form of "label inference," where the labels are intervals [Davis 1987]. ENVISION [de Kleer and Brown 1984] performs qualitative reasoning about the behaviour of physical systems over time. TMM [Dean 1985] is a temporal constraint system that records and reasons with changes to the world over time. SPAM [McDermott and Davis 1984] performs spatial reasoning. These systems are based on consistency techniques [Mackworth 1977], which handle only static constraint networks. To be able to generate constraints, the described systems are equipped with programming languages tailored to the application.

Constraint logic programming. Cleary incorporates a relational version of interval arithmetic, which he calls *Logical Arithmetic* [Cleary 1987], into Prolog. He introduces a new term "interval", which requires an extension of the unification algorithm. Cleary presents several "squeezing" algorithms that reduce arithmetic constraints over intervals. A constraint relaxation cycle coordinates the execution of the squeezing algorithms. However, there is a semantic problem in this approach. Variables bound to intervals, which are terms in the Herbrand universe, can be re-bound to smaller intervals. This is not part of resolution, where only a variable can be bound. It is not clear in what other, if any, sense this may be a logical inference. BNR Prolog [Older and Vellino 1990] has a partial implementation of logical arithmetic, which only handles closed intervals. The Echidna constraint reasoning system also supports relational interval arithmetic [Sidebottom and Havens 1992]. It is based on hierarchical consistency techniques [Mackworth *et al.* 1985]. Echidna is close to CHIP [Dincbas *et al.* 1988]; whereas we remain within the CLP framework.

3 Relational Interval Arithmetic

Cleary describes several algorithms to reduce constraints on intervals [Cleary 1987]. These algorithms work under a basic principle: they narrow intervals associated with a constraint by removing values that do not satisfy the constraint. We study the set-theoretic aspect of the algorithms and generalize them for narrowing intervals constrained by a relation p on $\mathbb{R}^n$. We then discuss interval narrowing for several common arithmetic relations. Interval narrowing is designed for the reduction of a single constraint. Typically, several constraints interact with one another by sharing intervals, resulting in a constraint *network*. We present an algorithm that coordinates the applications of interval narrowing to constraints in the network.

3.1 Basics of Interval Arithmetic

We use $\mathbb{R}$ to denote the set of real numbers and $\mathbb{F}$ a set of floating-point numbers. We also distinguish between *real intervals* and *floating-point intervals*. The set of real intervals, $I(\mathbb{R})$, is defined by

$$\begin{aligned} I(\mathbb{R}) \;=\; & \{(a,b] \mid a \in \mathbb{R} \cup \{-\infty\}, b \in \mathbb{R}\} \;\cup \\ & \{[a,b) \mid a \in \mathbb{R}, b \in \mathbb{R} \cup \{+\infty\}\} \;\cup \\ & \{[a,b] \mid a, b \in \mathbb{R}\} \;\cup \\ & \{(a,b) \mid a \in \mathbb{R} \cup \{-\infty\}, b \in \mathbb{R} \cup \{+\infty\}\}. \end{aligned}$$

Replacing $\mathbb{R}$ by $\mathbb{F}$ in the definition of $I(\mathbb{R})$, we obtain the definition of floating-point intervals. The symbols $-\infty$ and $+\infty$ are used to represent intervals without lower and upper bounds respectively. Every interval has the usual set denotation. For example, $[e,\pi) = \{x \mid e \le x < \pi\}$, $(-\infty, 4.5] = \{x \mid x \le 4.5\}$, and $(-\infty,+\infty) = \mathbb{R}$. We impose a partial ordering on real intervals; an interval I_1

is *smaller than or equal to* an interval I_2 if and only if $I_1 \subseteq I_2$. Given a set of intervals $\mathcal{I}$. $I \in \mathcal{I}$ is the *smallest* interval in $\mathcal{I}$ if I is smaller than or equal to I' for all $I' \in \mathcal{I}$.

Conventionally, real numbers are approximated by floating-point numbers by means of rounding or truncation. We approximate real intervals by floating-point intervals using the *outward rounding* function, $\xi : I(\mathbb{R}) \rightarrow I(\mathbb{F})$; if I is a real interval, $\xi(I)$ is the smallest floating-point interval containing I. It follows that $\xi(I) = I$ for each floating-point interval I. The IEEE floating-point standard [IEEE 1987] provides three user-selectable *directed rounding modes*: round toward $+\infty$, round toward $-\infty$, and round toward 0. The first two modes are essential and sufficient to implement outward rounding: we round toward $+\infty$ at the upper bound and toward $-\infty$ at the lower bound. In most hardware that conforms to the IEEE standard, performing directed rounding amounts to setting a hardware flag before performing the arithmetic operation.

We state without proof the following properties of the outward rounding function.

Lemma 1: If $A \in I(\mathbb{R})$ and $a' \in \xi(A)$, then there exists $a \in A$ such that $a' \in \xi([a, a])$. ∎

Lemma 2: If $A \in I(\mathbb{F})$, $a' \in A$, and $a \in \xi([a', a'])$, then $a \in A$. ∎

3.2 Interval Narrowing

An *i-constraint* is of the form $(p, \vec{I})$, where p is a relation on $\mathbb{R}^n$ and $\vec{I} = \langle I_1, \ldots, I_n \rangle$ is a tuple of floating-point intervals. Note that the number of intervals in the tuple $\vec{I}$ is equal to the arity of p. For any relation p of arity n, we can associate n set-valued functions with p:

$$\begin{aligned} & F_i(p)(S_1, \ldots, S_{i-1}, S_{i+1}, \ldots, S_n) \\ = & \ \{s_i \mid (s_1, \ldots, s_n) \in \mathcal{S}(p)\} \\ = & \ \pi_i(\mathcal{S}(p)), \end{aligned}$$

where $i = 1, \ldots, n$, the S_i's are sets, π_i is the projection function defined by $\pi_i(p) = \{s_i \mid (s_1, \ldots, s_n) \in p\}$, and $\mathcal{S}(p) = (S_1 \times \cdots \times S_{i-1} \times \pi_i(p) \times S_{i+1} \times \cdots \times S_n) \bigcap p$.

To ensure that the result of narrowing is an interval, we consider only relations p on $\mathbb{R}^n$, such that each $F_i(p)$ maps intervals to intervals. We now specify *interval narrowing* as an input-output pair.

Input: $\vec{I} = \langle I_1, \ldots, I_n \rangle$, where
I_i is a floating-point interval $(1 \leq i \leq n)$.
Output: $\vec{I'} = \langle I'_1, \ldots, I'_n \rangle$, where
$I'_i = I_i \bigcap \xi(F_i(p)(I_1, \ldots, I_{i-1}, I_{i+1}, \ldots, I_n))$.

The application of ξ in the formula ensures that the output intervals are floating-point intervals. If one or more I'_i is empty, then interval narrowing fails and the i-constraint $(p, \vec{I})$ is *i-inconsistent*. Otherwise it succeeds with $I'_1, \ldots, I'_n$ as output. Note that the output interval I'_i is a subset of the corresponding input interval I_i.

For example, the $F_i(\texttt{add})$'s of the relation $\texttt{add} = \{(x, y, z) \mid x, y, z \in \mathbb{R}, x + y = z\}$ are

$$F_1(\texttt{add})(I_2, I_3) = I_3 \ominus I_2, \quad F_2(\texttt{add})(I_1, I_3) = I_3 \ominus I_1,$$
$$F_3(\texttt{add})(I_1, I_2) = I_1 \oplus I_2,$$

where $A \oplus B = \{a + b \mid a \in A, b \in B\}$ and $A \ominus B = \{a - b \mid a \in A, b \in B\}$.

The following theorem shows the soundness and completeness of interval narrowing.

Theorem 3: Let C be $(p, \langle I_1, \ldots, I_n \rangle)$. If $(x_1, \ldots, x_n) \in p$ and $\langle I'_1, \ldots, I'_n \rangle$ are the output intervals obtained from interval narrowing of C, then $(x_1, \ldots, x_n) \in I_1 \times \cdots \times I_n$ if and only if $(x_1, \ldots, x_n) \in I'_1 \times \cdots \times I'_n$.

Proof: Since $I'_1 \times \cdots \times I'_n \subseteq I_1 \times \cdots \times I_n$, the if-part of the lemma is true. In the following, we prove the only-if-part of the lemma.

Suppose $(x_1, \ldots, x_n) \in I_1 \times \cdots \times I_n \cap p$. We have $x_i \in I_i$ for $i = 1, \ldots, n$ and $x_i \in F_i(p)(I_1, \ldots, I_{i-1}, I_{i+1}, \ldots, I_n)$ by definition. Therefore $x_i \in I'_i$ and $(x_1, \ldots, x_n) \in I'_1 \times \cdots \times I'_n$. ∎

The next lemma assists in expressing interval narrowing in terms of relational operations.

Lemma 4: For $A \in I(\mathbb{F})$ and $B \in I(\mathbb{R})$, $A \bigcap \xi(B) = \xi(A \bigcap B)$. ∎

We rewrite the output of interval narrowing as follows:

$$\begin{aligned} I'_i &= I_i \bigcap \xi(F_i(p)(I_1, \ldots, I_{i-1}, I_{i+1}, \ldots, I_n)) \\ &= \xi(I_i \bigcap F_i(p)(I_1, \ldots, I_{i-1}, I_{i+1}, \ldots, I_n)) \text{ by lemma 4} \\ &= \xi(I_i \bigcap \pi_i(\mathcal{I}(p))) \\ &= \xi(I_i \bigcap \{x_i \mid (x_1, \ldots, x_n) \in \mathcal{I}(p)\}) \\ &= \xi(\{x_i \in I_i \mid (x_1, \ldots, x_n) \in \mathcal{I}(p)\}) \\ &= \xi(\{x_i \mid (x_1, \ldots, x_n) \in ((I_1 \times \cdots \times I_n) \bigcap p)\}) \\ &= \xi(\pi_i((I_1 \times \cdots \times I_n) \bigcap p)), \end{aligned}$$

where $\mathcal{I}(p) = (I_1 \times \cdots \times I_{i-1} \times \pi_i(p) \times I_{i+1} \times \cdots \times I_n) \bigcap p$. In essence, interval narrowing computes the intersection of $I_1 \times \cdots \times I_n$ and p, and outward-rounds each projection of the resulting relation. We show in [Lee and van Emden 1991b] that interval narrowing is an instance of the LAIR rule [Van Hentenryck 1989], which is based on the arc consistency techniques [Mackworth 1977]. Figure 1 illustrates the interval narrowing of the constraint $(\texttt{le}, \langle I_1, I_2 \rangle)$, where $\texttt{le} = \{(x, y) \mid x, y \in \mathbb{R}, x \leq y\}$. In the diagram, the initial floating-point intervals are I_1 and I_2. The dotted region denotes the relation $\texttt{le}$; the region for $I_1 \times I_2$ is shaded with a straight-line pattern. Interval narrowing returns I'_1 and I'_2 by taking the projections of the intersection of the two regions. There is no need to perform outward-rounding in this example since the bounds of I'_1 and I'_2 share those of I_1 and I_2.

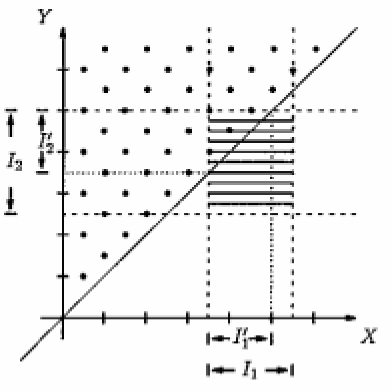

Figure 1: Pictorial illustration of interval narrowing.

3.3 Arithmetic Primitives

A useful relational interval arithmetic system should support some primitive arithmetic constraints, such as addition and multiplication. More complex constraints can then be built from these primitives. To ensure that a relation p is suitable for interval narrowing, we need to check that each $F_i(p)$ maps from real intervals to real intervals. If p is one of

$$\begin{aligned}
\texttt{eq} &= \{(x,x) \mid x \in \mathbb{R}\},\\
\texttt{add} &= \{(x,y,z) \mid x,y,z \in \mathbb{R}, x+y=z\},\\
\texttt{le} &= \{(x,y) \mid x,y \in \mathbb{R}, x \le y\},\\
\texttt{lt} &= \{(x,y) \mid x,y \in \mathbb{R}, x < y\},
\end{aligned}$$

we can verify easily that the $F_i(p)$'s satisfy the criterion.

The case for the multiplication relation $\texttt{multiply} = \{(x,y,z) \mid x,y,z \in \mathbb{R}, xy=z\}$, requires further explanation. Consider

$$F_1(\texttt{multiply})(I_2, I_3) = I_3 \oslash I_2 \text{ and}$$
$$F_2(\texttt{multiply})(I_1, I_3) = I_3 \oslash I_1,$$

where $A \oslash B = \{a/b \mid a \in A, b \in B, b \neq 0\}$. Note that $A \oslash B$ is not an interval in general. For example, $[1,1] \oslash [-2,3] = (-\infty, -1/2] \cup [1/3, +\infty)$ is a union of two disjoint intervals. The multiply relation does not satisfy the criterion for interval narrowing.

As suggested in [Cleary 1987], we can circumvent the problem by partitioning multiply into $\texttt{multiply}^+$ and $\texttt{multiply}^-$, where

$$\begin{aligned}
\texttt{multiply}^+ &= \{(x,y,z) \mid x,y,z \in \mathbb{R}, x \ge 0, xy=z\},\\
\texttt{multiply}^- &= \{(x,y,z) \mid x,y,z \in \mathbb{R}, x < 0, xy=z\}.
\end{aligned}$$

By restricting interval narrowing to one partition or the other, we can guarantee that the result of interval division is an interval. When a multiply constraint is encountered, we choose one of the partitions and perform narrowing; the other partition is visited upon automatic backtracking or under user control.

An advantage of relational interval arithmetic is that we do not have the division-by-zero problem. For example, the i-constraint $(\texttt{multiply}^+, \langle(4,+\infty), 0, [-3,5)\rangle)$ is reduced to $(\texttt{multiply}^+, \langle(4,+\infty), 0, 0\rangle)$.

Relations induced from transcendental functions and the disequality relation, such as $\texttt{sin} = \{(x,y) \mid x,y \in \mathbb{R}, y = \sin(x)\}$ and $\texttt{dif} = \{(x,y) \mid x,y \in \mathbb{R}, x \neq y\}$, also suffer from the same problem as the multiply relation. Similarly, we can solve the problem by appropriate partitioning of the relations.

3.4 Constraint Networks

The interval narrowing discussed so far reduces individual i-constraints. In practice, we have more than one constraint in a problem. These constraints may depend on one another by sharing intervals. By naming an interval by a variable and by having a variable occur in more than one constraint, we indicate that constraints share intervals. Note that the material in this section is not related to logic programming but is in conventional notation with destructive assignment[2]. We define an *i-network* to be a set of i-constraints. Consider the quadratic equation $x^2 - x - 6 = 0$, which can be rewritten to $x(x-1) = 6$. Suppose our initial guess for the positive root of the equation is $[1, 100]$. We can express the equation by the following i-network:

$$\{(\texttt{add}, \langle V_1, [1,1], V\rangle), (\texttt{multiply}^+, \langle V, V_1, [6,6]\rangle)\},$$

where the variables V and V_1 are intervals $[1, 100]$ and $(-\infty, +\infty)$ respectively.

Our goal is to use interval narrowing to reduce i-networks. Note the following two observations. First, the reduction of an i-constraint C in the i-network affects other i-constraints that share variables with C. Second, interval narrowing is idempotent as shown in the following lemma.

Lemma 5: Let $\vec{I} = \langle I_1, \ldots, I_n\rangle$, $\vec{I'} = \langle I_1', \ldots, I_n'\rangle$, and $\vec{I''} = \langle I_1'', \ldots, I_n''\rangle$ be tuples of floating-point intervals and p a relation on $\mathbb{R}^n$. If, by interval narrowing, $\vec{I'}$ is obtained from $\vec{I}$ and $\vec{I''}$ is obtained from $\vec{I'}$, then $\vec{I'} = \vec{I''}$.

Proof: To prove the equality of $\vec{I'}$ and $\vec{I''}$, we prove $I_i' = I_i''$ for $i = 1, \ldots, n$. By the definition of interval narrowing and lemma 4, we have

$$I_i' = \xi(I_i \cap F_i(p)(I_1, \ldots, I_{i-1}, I_{i+1}, \ldots, I_n)),$$
$$I_i'' = \xi(I_i' \cap F_i(p)(I_1', \ldots, I_{i-1}', I_{i+1}', \ldots, I_n')).$$

It is obvious that $I_i'' \subseteq I_i'$. Next we prove $I_i' \subseteq I_i''$.

Suppose $a_i' \in I_i'$. There exists

$$a_i \in (I_i \bigcap F_i(p)(I_1, \ldots, I_{i-1}, I_{i+1}, \ldots, I_n))$$

[2]In section 4, we show how we use logical variables to replace the conventional variables.

such that $a_i' \in \xi([a_i, a_i])$ by lemma 1. By the definition of $F_i(p)$, for $j = 1, \ldots, i-1, i+1, \ldots, n$, there exists $a_j \in I_j$ such that $(a_1, \ldots, a_n) \in p$. Since $a_i \in I_i$, we have $a_j \in I_j'$ for each j. Thus, $a_i \in F_i(p)(I_1', \ldots, I_{i-1}', I_{i+1}', \ldots, I_n')$. This implies that $a_i \in I_i''$. By lemma 2, $a_i' \in I_i''$. ∎

An i-constraint $(p, \vec{I})$ is *stable* if applying interval narrowing on $\vec{I}$ results in $\vec{I}$. An i-network is *stable* if every i-constraint in the i-network is stable. The reduction of an i-network amounts to transforming it into a stable one.

A naive approach for the reduction of an i-network is to reduce each i-constraint in the i-network in turn until every i-constraint becomes stable. As suggested by lemma 5, this method is inefficient since much computation is wasted in reducing stable i-constraints. Algorithm 1, which is based on the constraint relaxation algorithm described in [Cleary 1987], is the pseudocode of a more efficient procedure. The algorithm tries to avoid reductions of stable i-constraints and, in this respect, it is similar to AC-3 [Mackworth 1977] and the Waltz algorithm [Waltz 1975]. Without loss of generality, we assume that every i-constraint in the i-network is of the form $(p, \langle V_1, \ldots, V_n \rangle)$, where V_i's are interval-valued variables.

```
initialize list $A$ to hold all i-constraints in the i-network
initialize $P$ to the empty list
while $A$ is not empty
    remove the first i-constraint, $(p, \vec{V})$, from $A$
    apply interval narrowing on $\vec{V}$ to obtain $\vec{V}'$
    if interval narrowing fails then
        exit with failure
    else if $\vec{V} \neq \vec{V}'$ then
        $\vec{V} \leftarrow \vec{V}'$
        foreach i-constraint $(q, \vec{Y})$ in $P$
            if $\vec{V}$ and $\vec{Y}$ share narrowed variable(s) then
                remove $(q, \vec{Y})$ from $P$ and append it to $A$
            endif
        endforeach
    endif
    append $(p, \vec{V})$ to the end of $P$
endwhile
```

Algorithm 1: A Relaxation Algorithm.

Algorithm 1 resembles a classical iterative numerical-approximation technique called "relaxation" [Southwell 1946], which was first adopted in a constraint system in [Sutherland 1963]. Numerical relaxation may have numerical stability problems; the procedure may fail to converge or terminate even when the constraints have a solution. Algorithm 1 does not suffer from this problem as shown in the following theorem.

Theorem 6: Algorithm 1 terminates. The resulting i-network is either i-inconsistent or stable. ∎

The validity of theorem 6 is easy to check since the precision of a floating-point system is finite and thus interval narrowing cannot occur indefinitely due to the use of outward rounding.

In the following, we show how algorithm 1 finds the positive root of the equation $x^2 - x - 6 = 0$ with initial guess in $[1, 100]$. Initially, the passive list, P, is empty and the active list of i-constraints, A, is $[C_1, C_2]$, where

$$\begin{aligned} C_1 &= (\texttt{add}, \langle V_1, [1,1], V \rangle) \text{ and} \\ C_2 &= (\texttt{multiply}^+, \langle V, V_1, [6,6] \rangle). \end{aligned}$$

We remove the first i-constraint C_1 from A and reduce it as shown in figure 2.

The updated values of V and V_1 are $[1, 100]$ and $[0, 99]$ respectively. Similar narrowing is performed on the `multiply`$^+$ i-constraint. The process repeats until the precision of the underlying floating-point system is reached and no more narrowing takes place. The history of the values of A, P, V, and V_1, with four significant figures, after each narrowing is summarized in table 1.

Table 1: Traces of A, P, V, and V_1.

A	P	V	V_1
$[C_1, C_2]$	$[]$	$[1, 100]$	$(-\infty, +\infty)$
$[C_2]$	$[C_1]$	$[1, 100]$	$[0, 99]$
$[C_1]$	$[C_2]$	$[1, 100]$	$[0.06, 6]$
$[C_2]$	$[C_1]$	$[1.06, 7]$	$[0.06, 6]$
$[C_1]$	$[C_2]$	$[1.06, 7]$	$(0.8571, 5.661)$
$[C_2]$	$[C_1]$	$(1.857, 6.661)$	$(0.8571, 5.661)$
$[C_1]$	$[C_2]$	$(1.857, 6.661)$	$(0.9009, 3.231)$
$[C_2]$	$[C_1]$	$(1.900, 4.231)$	$(0.9009, 3.231)$
$[C_1]$	$[C_2]$	$(1.900, 4.231)$	$(1.418, 3.157)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$[]$	$[C_1, C_2]$	$(2.999, 3.001)$	$(1.999, 2.001)$

It is well-known that arc-consistency techniques are "incomplete" [Mackworth 1977]: a network can be stable but neither a solution nor inconsistency is found. In the finite domain case, enumeration, instantiation, and backtracking can be used to find a particular solution after the constraint network becomes stable. This method is infeasible for interval domains, which are infinite sets. We use domain splitting [Van Hentenryck 1989] in place of enumeration and instantiation. When an i-network becomes stable, we split an interval into two partitions, choose one partition and visit the other upon automatic backtracking or user control.

Input Intervals = $\langle I_1, I_2, I_3\rangle$	$(-\infty,+\infty)$	$[1,1]$	$[1,100]$
$F_1(\mathtt{add}) = [1,100] \ominus [1,1]$	$[0,99]$		
$F_2(\mathtt{add}) = [1,100] \ominus (-\infty,+\infty)$		$(-\infty,+\infty)$	
$F_3(\mathtt{add}) = (-\infty,+\infty) \oplus [1,1]$			$(-\infty,+\infty)$
Output Intervals = $\langle I_1', I_2', I_3'\rangle$	$[0,99]$	$[1,1]$	$[1,100]$

Figure 2: Interval narrowing of an add constraint.

4 ICLP($\mathcal{R}$)

So far, we have explained how a network of constraints in terms of floating-point intervals can be made stable. We have not considered how such networks can be specified. One language is CLP($\mathcal{R}$). An i-constraint $(p, \langle I_1, \ldots, I_n\rangle)$ can be expressed in CLP($\mathcal{R}$) as

$$X_1 \in I_1, \ldots, X_n \in I_n, p(X_1, \ldots, X_n),$$

where $X_i \in I_i$ stands for an appropriate set of inequalities. In this representation, we don't need conventional variables. Constraints share intervals when they share logical variables.

When a query to a logic program is answered according to the CLP scheme at each step, a network of constraints is solved. A special case of such a constraint is a variable's membership of a domain. According to the CLP scheme, it is possible at any stage that the domain inclusions of the current set of constraints is inconsistent in Mackworth's sense of the Constraint-Satisfaction Problem. We modify the basic CLP scheme by inserting a constraint simplification step and show that interval narrowing is a constraint simplification operation.

In principle, any of the constraint-satisfaction algorithms by Mackworth can be used. In this paper we are concerned with real-valued variables. As we argued, the only known way of obtaining correct answers involving real numbers on a machine with floating-point numbers is to use interval arithmetic. Accordingly, we explain the theory of ICLP($\mathcal{R}$), where the sub-network consisting of i-constraints is narrowed according to the method described above.

An example. The i-constraint

$$(\mathtt{add}, \langle [0,2], [1,3], [4,6]\rangle)$$

is represented by

$$X \geq 0, X \leq 2, Y \geq 1, Y \leq 3, Z \geq 4, Z \leq 6, X + Y = Z.$$

However, if we submit the above example as a query to CLP($\mathcal{R}$) Version 2.02, we get the answer constraint

$$X = -Y + Z, 2 + Y \geq Z, Y \geq 1, 3 \geq Y, Z \geq 4, \\ 6 \geq Z, X \geq 0.$$

Examining the answer more carefully, we note that

$$X = -Y + Z \equiv X + Y = Z \text{ and } 2 + Y \geq Z \equiv X \leq 2.$$

Thus the answer constraint is the original query disguised in a slightly different form. CLP($\mathcal{R}$) only checks the solvability of the constraint but does not remove undesirable values from the intervals. A more useful answer constraint is

$$X \geq 1, X \leq 2, Y \geq 2, Y \leq 3, Z \geq 4, Z \leq 5, X + Y = Z.$$

The modified CLP scheme ICLP($\mathcal{R}$) is CLP($\mathcal{R}$) enhanced with interval narrowing and algorithm 1. The operational semantics of ICLP($\mathcal{R}$) is based on a generalization of $\mathcal{M}_{\mathcal{X}}$-derivations [Jaffar and Lassez 1986]. Let P be a CLP($\mathcal{X}$) program, where $\mathcal{X}$ is a structure with model $\mathcal{M}_{\mathcal{X}}$, and $\leftarrow G_i$ be a goal. $\leftarrow G_{i+1}$ is *$\mathcal{M}'_{\mathcal{X}}$-derived* from $\leftarrow G_i$ if

1. $\leftarrow G'$ is $\mathcal{M}_{\mathcal{X}}$-derived from $\leftarrow G_i$, and
2. $\leftarrow G_{i+1} = \nu(\leftarrow G')$, where ν is a *normal-form function* that maps from goal to goal such that $P \models_{\mathcal{M}_{\mathcal{X}}} \exists(G') \Leftrightarrow P \models_{\mathcal{M}_{\mathcal{X}}} \exists(G_{i+1})$.

An *$\mathcal{M}'_{\mathcal{X}}$-derivation* is a, possibly infinite, sequence of goals $G = G_0, G_1, G_2, \ldots$ such that G_{i+1} is $\mathcal{M}'_{\mathcal{X}}$-derived from G_i. A $\mathcal{M}'_{\mathcal{X}}$-derivation is *successful* if it is finite and the last goal contains no atoms. The soundness and completeness of $\mathcal{M}'_{\mathcal{X}}$-derivations follow directly from the soundness and completeness of $\mathcal{M}_{\mathcal{X}}$-derivations and the definition of the normal-form function. A $\mathcal{M}'_{\mathcal{X}}$-derivation is *finitely-failed* if it is finite, the last goal has one or more atoms, and condition 1 does not hold.

The $\mathcal{M}'_{\mathcal{X}}$-derivation step is not new. In fact, it has been implemented in other CLP systems as a constraint simplification step. The $\mathcal{M}_{\mathcal{X}}$-derivation step only checks the solvability of the constraint accumulated so far. Therefore, the answer constraint of a successful $\mathcal{M}_{\mathcal{X}}$-derivation is usually complex and difficult to interpret. A useful system should simplify the constraint to a more "readable" form. For example, CLP($\mathcal{R}$) simplifies the constraint $\{X + Y = 4, X - Y = 1\}$ to $\{X = 2.5, Y = 1.5\}$. Suppose the goal $\leftarrow c', \vec{A}'$ is $\mathcal{M}_{\mathcal{X}}$-derived from $\leftarrow c, \vec{A}$. CLP($\mathcal{R}$) simplifies c' to c'' such that $\models_{\mathcal{M}_{\mathcal{X}}} \exists(c') \Leftrightarrow \models_{\mathcal{M}_{\mathcal{X}}} \exists(c'')$ and thus

$$P \models_{\mathcal{M}_{\mathcal{X}}} \exists(c', \vec{A}') \Leftrightarrow P \models_{\mathcal{M}_{\mathcal{X}}} \exists(c'', \vec{A}');$$

CLP($\mathcal{R}$) is based on $\mathcal{M}'_{\mathcal{X}}$-derivation.

Theorem 7: If

$$\begin{aligned} \mathcal{C} &= \{X_1 \in I_1, \ldots, X_n \in I_n, p(X_1, \ldots, X_n)\} \text{ and} \\ \mathcal{C}' &= \{X_1 \in I'_1, \ldots, X_n \in I'_n, p(X_1, \ldots, X_n)\}, \end{aligned}$$

where I'_i is obtained from I_i by interval narrowing for $i = 1, \ldots, n$, then $\models_{\mathcal{M}_{\mathcal{X}}} \exists(\mathcal{C}) \Leftrightarrow \models_{\mathcal{M}_{\mathcal{X}}} \exists(\mathcal{C}')$.

Proof: The theorem follows directly from theorem 3. ■

Theorem 7 guarantees that interval narrowing transforms a constraint into a stable constraint with the same solution space. Algorithm 1, which performs narrowing repeatedly on i-constraints in a network, is thus a normal-form function.

Partitioning of relations can also be expressed compactly in ICLP($\mathcal{R}$). For example, the `multiply` relation can be defined by

```
multiply(X,Y,Z) :- X ≥ 0,multiply⁺(X,Y,Z).
multiply(X,Y,Z) :- X < 0,multiply⁻(X,Y,Z).
```

A meta-interpreter ICLP($\mathcal{R}$) written in CLP($\mathcal{R}$) is described in [Lee and van Emden 1991a]. We have not yet included outward rounding in the current implementation. Table 1 is derived from a trace produced by our prototype, except that the outward rounding has been added manually.

5 Concluding Remarks

We have developed the essential components of a relational interval arithmetic system. Interval narrowing establishes (1) the criterion that an arithmetic relation has to satisfy to be used as arithmetic constraint in relational interval arithmetic, and (2) the reduction of arithmetic constraint using the interval functions induced from the constraint. Algorithm 1 then coordinates the applications of narrowing to transform a constraint network into its stable form.

The incorporation of relational interval arithmetic in CLP($\mathcal{R}$) makes it possible to describe programs, constraints, queries, intervals, answers, and variables in a coherent and semantically precise language—*logic*. The semantics of ICLP($\mathcal{R}$) is based on $\mathcal{M}'_{\mathcal{X}}$-derivation, which is a logical deduction. Consequently, *numerical computation is deduction* in ICLP($\mathcal{R}$), which is a general-purpose programming language allowing compact description and dynamic growth of constraint networks. One advantage of ICLP($\mathcal{R}$) over CLP($\mathcal{R}$) is the ability to handle nonlinear constraints, which are delayed in CLP($\mathcal{R}$). It is important to note that ICLP($\mathcal{R}$) is not another instance of the CLP scheme. It is a correct implementation of CLP($\Re$).

The ICLP($\mathcal{R}$) meta-interpreter shows the feasibility of our approach. Future work includes extending CLP($\mathcal{R}$) at the source level, to ICLP($\mathcal{R}$) to improve efficiency. We also plan to investigate applications in such areas as finite element analysis, and spatial and temporal reasoning.

Acknowledgements

We thank Mantis Cheng, Bill Havens, Alan Mackworth, Clifford Walinsky, and the anonymous referees for helpful suggestions. Paul Strooper gave constructive comments to early drafts of this paper. Generous support was provided by the British Columbia Advanced Systems Institute, the Canada Natural Science and Engineering Research Council, the Canadian Institute for Advanced Research, the Institute of Robotics and Intelligent Systems, and the Laboratory for Automation, Communication and Information Systems Research.

References

[Cleary 1987] J.G. Cleary. Logical arithmetic. *Future Computing Systems*, 2(2):125–149, 1987.

[Davis 1987] E. Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32:281–331, 1987.

[de Kleer and Brown 1984] J. de Kleer and J.S. Brown. A qualitative physics based on confluences. *Artificial Intelligence*, 24:7–83, 1984.

[Dean 1985] T. Dean. Temporal imagery: An approach to reasoning about time for planning and problem solving. Technical Report 433, Yale University, New Haven, CT, USA, 1985.

[Dincbas *et al.* 1988] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The constraint logic programming language CHIP. In *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS'88)*, pages 693–702, Tokyo, Japan, December 1988.

[Hayes 1973] P.J. Hayes. Computation and deduction. In *Proceedings of the Second MFCS Symposium*, pages 105–118. Czechoslovak Academy of Sciences, 1973.

[Jaffar and Lassez 1986] J. Jaffar and J-L. Lassez. Constraint logic programming. Technical Report 72, Department of Computer Science, Monash University, Clayton, Victoria, Australia, 1986.

[Jaffar and Lassez 1987] J. Jaffar and J-L. Lassez. Constraint logic programming. In *Proceedings of the 14th ACM POPL Conference*, pages 111–119, Munich, January 1987.

[Jaffar *et al.* 1990] J. Jaffar, S. Michaylov, P.J. Stuckey, and R.H.C. Yap. The CLP($\mathcal{R}$) language and system. Technical Report CMU-CS-90-181, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, 1990.

[Lee and van Emden 1991a] J.H.M. Lee and M.H. van Emden. Adapting CLP($\mathcal{R}$) to floating-point arithmetic. Technical Report LP-18 (DCS-183-IR), Department of Computer Science, University of Victoria, Victoria, B.C., Canada, December 1991.

[Lee and van Emden 1991b] J.H.M. Lee and M.H. van Emden. Numerical computation can be deduction in CHIP. Technical Report LP-19 (DCS-184-IR), Department of Computer Science, University of Victoria, Victoria, B.C., Canada, December 1991. (submitted for publication).

[Mackworth 1977] A.K. Mackworth. Consistency in networks of relations. *AI Journal*, 8(1):99–118, 1977.

[Mackworth *et al.* 1985] A.K. Mackworth, J.A. Mulder, and W.S. Havens. Hierarchical arc consistency: Exploiting structured domains in constraint satisfaction problems. *Computational Intelligence*, 1:118–126, 1985.

[McDermott and Davis 1984] D.V. McDermott and E. Davis. Planning routes through uncertain territory. *Artificial Intelligence*, 22:107–156, 1984.

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

[Moore 1988] R.E. Moore, editor. *Reliability in Computing—The Role of Interval Methods in Scientific Computing.* Academic Press, 1988.

[IEEE 1987] Members of the Radix-Independent Floating-point Arithmetic Working Group. IEEE standard for radix-independent floating-point arithmetic. Technical Report ANSI/IEEE Std 854-1987, The Institute of Electrical and Electronics Engineers, New York, USA, 1987.

[Older and Vellino 1990] W. Older and A. Vellino. Extending Prolog with constraint arithmetics on real intervals. In *Proceedings of the Canadian Conference on Computer & Electrical Engineering*, Ottawa, Canada, 1990.

[Sidebottom and Havens 1992] G. Sidebottom and W.S. Havens. Hierarchical arc consistency for disjoint real intervals in constraint logic programming. *Computational Intelligence*, 8(2), May 1992.

[Southwell 1946] R.V. Southwell. *Relaxation Methods in Theoretical Physics.* Oxford University Press, 1946.

[Sutherland 1963] I.E. Sutherland. *SKETCHPAD: a Man-Machine Graphical Communication System.* PhD thesis, MIT Lincoln Labs, Cambridge, MA, 1963.

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

[Waltz 1975] D. Waltz. Understanding line drawings of scenes with shadows. In P. Winston, editor, *The Psychology of Computer Vision.* McGraw-Hill, 1975.