Towards Practical Interval Constraint Solving in Logic Programming

C.K. Chiu and J.H.M. Lee
Department of Computer Science
The Chinese University of Hong Kong
Shatin, Hong Kong
{ckchiu,jlee}@cs.cuhk.hk

Abstract

Existing interval constraint logic programming languages, such as BNR Prolog, work under the framework of interval narrowing and are deficient in solving linear systems, which constitute an important class of problems in engineering and other applications. In this paper, an interval linear equality solver, which is based on generalized interval arithmetic and Gaussian elimination, is proposed. We show how the solver can be adapted to incremental execution and incorporated into a constraint logic programming language already equipped with a non-linear solver based on interval narrowing. The two solvers interact and cooperate during computation, resulting in a practical interval constraint arithmetic language CIAL. A prototype of CIAL, based on CLP($\mathcal{R}$), is constructed and compared favourably against several major constraint logic programming languages.

1 Introduction

The goal is to provide *sound* and *practical* arithmetic in logic programming. Towards the first aim, current status of Prolog arithmetic has much to be desired. First, Prolog arithmetic is functional in nature. Second, roundoff errors induced by floating-point arithmetic destroys the soundness of computation. The advent of constraint logic programming [9] presents a solution to the first problem but the implementation of CLP languages, such as CLP($\mathcal{R}$) [11], are mostly based on floating-point arithmetic. The second problem remains.

RISC-CLP($\mathcal{R}$) [8] uses symbolic algebra methods to tackle the roundoff error problem. Previous efforts in the sub-symbolic camp, such as BNR Prolog [18], employ interval methods [17] and belong to the family of consistency techniques [15]. We collectively call these techniques *interval narrowing* hereafter. Interval narrowing has been shown to be applicable to critical path scheduling [18], X-ray diffraction crystallography [18], boolean constraint solving [3], disjunctive constraint solving [3, 19], and finite domain constraint solving [3]. However, it is deficient in handling linear constraints.

For example, interval narrowing fails to narrow for such simple systems as "$\{X + Y = 5, X - Y = 6\}$." Cleary [4] proposes a form of case analysis technique, *domain splitting*, as a remedy. Domain splitting partitions an interval into two, visit one, and visit the other upon backtracking. This backtracking tree search is expensive to perform. Furthermore, interval narrowing may sometimes fail or take a long time to detect inconsistency of linear systems.[1] Thus, interval narrowing is opted for improvement in terms of *practicality*.

We propose CIAL (for *Constraint Interval Arithmetic Language*), an extension of ICLP($\mathcal{R}$) [12], the constraint solver of which is based on interval narrowing. The syntax and semantics of CIAL are almost identical to those of ICLP($\mathcal{R}$) [12], except that the arithmetic operators use slightly different, but self-explanatory, symbols. The enhancement is in terms of constraint solving capabilities. CIAL consists of a *linear interval constraint solver* in addition to an interval narrowing solver. The cooperation of the two solvers enables CIAL to solve larger number of classes of numerical problems than any of the similar systems when used alone. To demonstrate the feasibility of our proposal, we have constructed a prototype of CIAL and compared it to BNR Prolog, CLP(BNR), CLP($\mathcal{R}$), and Echidna using different types of numerical examples.

The paper is organized as follows. Section 2 reviews related work. In section 3, we define the notion of linear interval constraints and discuss adaptation of ordinary Gaussian elimination procedures to their interval versions. This is the basis of the linear solver of CIAL. Section 4 presents the operational semantics of CIAL and each of the components of a CIAL architecture in details. In particular, the interaction between the two constraint solvers is discussed in details. In section 5, the prototype implementation of CIAL and its comparisons to other major interval systems are described. Section 6 summarizes the contributions of the paper and sheds light on further work.

2 Related Work

Cleary [4] introduces "logical arithmetic," a relational version of interval arithmetic, into Prolog. He describes distinct algorithms, one for each kind of constraint over intervals, that narrow intervals associated with a constraint by removing values that do not satisfy the constraint. A constraint relaxation cycle is needed to coordinate the execution of the narrowing algorithms for a network of constraints. BNR-Prolog [18] and its sequel CLP(BNR) [3] provide relational interval arithmetic in a way that is loosely based on Cleary's pre-publication idea, differing somewhat in particulars. Sidebottom and

[1] Detailed analysis of the two claims requires knowledge of interval narrowing, which is out of the scope of this paper. We justify our assertion by experiments with a few interval narrowing implementations, as described in section 5.

Havens [19] design and implement a version of relational interval arithmetic in the constraint reasoning system Echidna [7]. Based on hierarchical consistency techniques [16], Echidna can handle unions of disjoint intervals. Lhomme [14] analyzes the complexity of consistency techniques for numeric CSP's and proposes partial consistency techniques, whose complexities can be tuned by adjusting the bound width of the resulting intervals. Lee and van Emden [12, 13] generalize Cleary's algorithms for narrowing intervals constrained by any relations p on $I(\mathbb{R})^n$. They also show how the generalized algorithm can be incorporated in CLP($\mathcal{R}$) [11] and CHIP [5] in such a way that the languages' logical semantics is preserved. Benhamou *et al* [2] replaces the usual interval narrowing operator of previous CLP(Interval) languages by an operator based on interval Newton method to speed up non-linear constraint solving.

3 An Interval Gaussian Elimination Solver

Motivated by the success of the linear solver of CLP($\mathcal{R}$), CIAL's linear solver is also an adaptation of the Gaussian elimination procedure. There are several variants of the Gaussian elimination procedure for solving linear equalities.[2] An interval version of any of them can be obtained by simply replacing each ordinary arithmetic operator by the corresponding interval arithmetic counterpart. Answers generated using this naive approach will not be as sharp as possible, in general, due to *outward-rounding* and the *variable dependency* problem. The former is unavoidable in performing interval arithmetic in a floating-point system. Generalized interval arithmetic by Hansen [6] presents a solution to reduce the effect of the latter.

3.1 Basics of Interval Arithmetic

A good introduction to interval arithmetic can be found in [1]. In the following, we define only the necessary notations and concepts. To represent intervals, we use the usual mathematical notations, such as $(-\infty, 2]$ which denotes the set $\{x \mid x \leq 2\}$. We differentiate between *real intervals* and *floating-point intervals*. The bounds of the former can be any real numbers; while the bounds of the latter are restricted to floating-point numbers. In computer interval arithmetic, we deal with floating-point intervals.

If $\cdot \in \{+, -, \times, /\}$, we denote the corresponding real (or floating-point) interval operation by $\odot$:

$$A \odot B = \{a \cdot b \mid a \in A, b \in B\},$$

where A and B are real (or floating-point) intervals[3]. In the case of interval division, $\oslash$, B cannot contain 0. Floating-point intervals are not closed

[2] We refer the interested readers to the outstanding book by Wilkinson [20].

[3] Interval versions of other arithmetic operators, such as squaring, can be defined similarly.

under interval operations. In floating-point arithmetic, real numbers are approximated by floating-point numbers using rounding or truncation. In interval arithmetic, we approximate real intervals by floating-point intervals using the *outward-rounding* function ξ. Intuitively, if I is a non-empty real interval, $\xi(I)$ is the smallest floating-point interval containing I.

Constraints in CIAL are over real numbers. An interval is represented by an appropriate pair of inequality constraints bounding the value of a logical variable, which represents an unknown real number. We denote logical variables by such upper case letters as X, Y and Z. For example, $\{X > 3, X \leq 6\}$ denotes the interval $(3, 6]$. Mathematical variables (or constants) referring to non-empty floating-point intervals are denoted by upper (or lower) case letters with superscript I.

Before we discuss the linear interval constraint solver, we define the notion of a *linear interval constraint*. An interval is *non-narrowable* if its width is less than a user-defined value or if it cannot be further split in the underlying floating-point system (i.e. when the lower and upper bounds of the interval are "adjacent" in the floating-point line). Theoretically, CIAL provides only a set of primitive constraints.[4] Among others are the "$X+Y=Z$" and "$X*Y=Z$" constraints for addition and multiplication respectively, which are related to the notion of linearity. Here $X \in J_1^I$, $Y \in J_2^I$, and $Z \in J_3^I$. The addition constraint is *linear* by definition. The multiplication constraint is *linear* if either J_1^I or J_2^I becomes non-narrowable.

3.2 Naive Interval Gaussian Elimination

Naive interval Gaussian elimination can be obtained by replacing each ordinary arithmetic operator by the corresponding interval arithmetic counterpart. Outward-rounding is necessary to uphold the *guaranteed inclusion* property of computer interval arithmetic. We explain the variable dependency problem of naive interval Gaussian elimination in the following.

The variable dependency problem is caused by the fact that multiple occurrences of a given variable in an interval computation are treated as different variables in each occurrence. This widens the computed interval unnecessarily. The simplest examples for illustrating this problem are $A^I \ominus A^I$ and $A^I \oslash A^I$. It can be checked easily that $[0,0] \subset A^I \ominus A^I$ and $[1,1] \subset A^I \oslash A^I$. Consider the following system of interval linear equations of two variables and their associated analytical solution.

$$
\begin{aligned}
&\left\{\begin{array}{lcl} a_{11}^I \otimes X^I \oplus a_{12}^I \otimes Y^I & = & b_1^I \\ a_{21}^I \otimes X^I \oplus a_{22}^I \otimes Y^I & = & b_2^I \end{array}\right. \\
\Rightarrow \quad &\left\{\begin{array}{lcl} X^I & = & (b_1^I \otimes a_{22}^I \ominus b_2^I \otimes a_{12}^I) \oslash (a_{11}^I \otimes a_{22}^I \ominus a_{12}^I \otimes a_{21}^I) \\ Y^I & = & (a_{11}^I \otimes b_2^I \ominus a_{21}^I \otimes b_1^I) \oslash (a_{11}^I \otimes a_{22}^I \ominus a_{12}^I \otimes a_{21}^I) \end{array}\right.
\end{aligned}
$$

[4] Users can write the constraints in more convenient notation, such as "$X*Y+3*Z > 5*Y$," but the constraints are eventually translated to conjunctions of primitive constraints, possibly with the aid of extra variables.

The variable dependency problem occurs in the calculation of the value of each variable. Substituting the coefficients with some concrete data, we get the following system of equalities.

$$\left\{ \begin{array}{lcl} [3.0002, 3.0003] \otimes X^I \oplus [4.0005, 4.0006] \otimes Y^I & = & [1, 1] \\ [2.0001, 2.0002] \otimes X^I \oplus [1.0002, 1.0003] \otimes Y^I & = & [2, 2] \end{array} \right. \tag{1}$$

Solving the above system using naive interval Gaussian elimination yields the following results.

$$\left\{ \begin{array}{lcl} X^I & = & (1.39966625\ldots, 1.40028177\ldots) \\ Y^I & = & (-0.80016132\ldots, -0.79975469\ldots) \end{array} \right.$$

In the following, we show that better results can be obtained by adopting operators from generalized interval arithmetic in Gaussian elimination.

3.3 Generalized Interval Gaussian Elimination

A *generalized interval Gaussian elimination* procedure is obtained by replacing the ordinary arithmetic operators in an Gaussian elimination procedure by generalized interval arithmetic operators [6].

Interval computation on generalized interval arithmetic is somewhat similar to the use of interval mean-value theorem [17] for obtaining sharper results. However, calculating the derivative of functions can be avoided. An interval $X^I = [a, b]$ is represented as a *generalized interval* in the form,

$$X^I = Y^I \oplus [-c, c] \otimes Z^I \quad \text{where } y - c = a, y + c = b, Y^I = [y, y] \text{ and } Z^I = [1, 1].$$

If an interval X_i^I is computed depending on n other input intervals, the resulting interval is expressed as a generalized interval,

$$X_i^I = Y_i^I \oplus \sum_{r=1}^{n} ([-c_r, c_r] \otimes Z_{ir}^I)$$

Y_i^I and Z_{ir}^I are numerical intervals computed by ordinary interval arithmetic operators. Note that a generalized interval keeps as many subterms as the number of intervals on which it depends. These subterms provide information to locate multiple occurrences of a variable during computation, so as to reduce the effect of the variable dependency problem. The computation rules for Y_i^I and Z_{ir}^I are beyond the scope of this paper. We study the generalized interval minus as an example and refer the readers to [6]. The rule for generalized interval minus (denoted by $-_g$) for $X_k^I = X_i^I -_g X_j^I$ is:

$$Y_k^I = Y_i^I \ominus Y_j^I \ , \ Z_{kr}^I = Z_{ir}^I \ominus Z_{jr}^I.$$

Suppose we want to compute $X_k^I = X_i^I -_g X_i^I$, where $X_i^I = [a, b]$. We get $X_k^I = [0, 0]$ instead of $[a - b, b - a]$, since $Y_i^I \ominus Y_i^I = [\frac{a+b}{2}, \frac{a+b}{2}] - [\frac{a+b}{2}, \frac{a+b}{2}] = [0, 0]$ and $Z_{ir}^I \ominus Z_{ir}^I = [1, 1] \ominus [1, 1] = [0, 0]$.

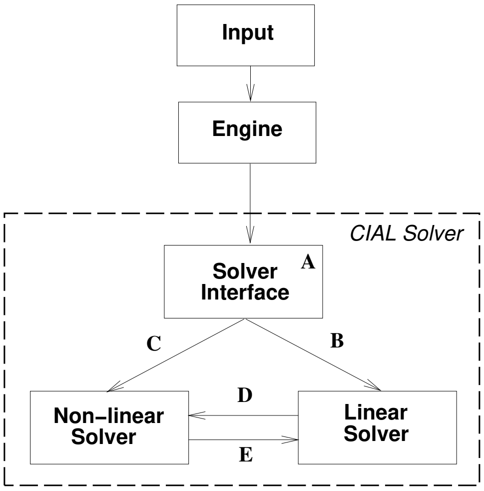


Figure 1: An CIAL Architecture

Solving the system of linear equations (1) using generalized interval Gaussian elimination yields a sharper result.

$$\begin{cases} X^I & = & (1.39981395\ldots, 1.40013404\ldots) \\ Y^I & = & (-0.80010553\ldots, -0.79981048\ldots) \end{cases}$$

In our implementation of generalized interval Gaussian elimination, we represent real coefficients in the original (before transformation) system of equations as generalized intervals.

4 An Overview of the CIAL System

Figure 1 reviews an architecture for CIAL. The *input* and the *engine* components are an adaptation of a Prolog interpreter. Their job is to perform unification (simply equality constraint solving), goal reduction, and to deliver the constraints collected at each derivation step to the *solver interface*. The interface in turn distributes the constraints to the *linear solver* and the *non-linear solver* accordingly. During execution, some constraints which cannot be solved in the linear solver will be moved to the non-linear solver. Also, the constraints in a solver will be waken up by another solver due to the modification of common variables.

In the following, we describe the operational semantics of CIAL, each component of the architecture and the interaction between the two solvers in more details.

4.1 Operational Semantics

CIAL shares the same syntax and declarative semantics [9] as CLP($\mathcal{R}$). Its operational semantics is based on the generalized $\mathcal{M}_\mathcal{X}$ derivation [12].

Let P be a CLP($\mathcal{X}$) program, where $\mathcal{X}$ is a domain of computation with structure $\mathcal{M}_\mathcal{X}$, and $\leftarrow G_i = \leftarrow c, B_1, \ldots, B_k, \ldots, B_n$, $n > 0$ a goal. $\leftarrow G_{i+1}$ is *derived in $\mathcal{M}_\mathcal{X}$ in a generalized sense* from $\leftarrow G_i$ if

1. there exists a clause in P with variant $A \leftarrow c', \vec{B}$ such that $c'' = c \cup c' \cup \{B_k = A\}$ is solvable in $\mathcal{M}_\mathcal{X}$,
2. $\leftarrow G' = \leftarrow c'', B_1, \ldots, \vec{B}, \ldots, B_n$,
3. $\leftarrow G_{i+1} = \nu(\leftarrow G')$, where ν is a *normal-form function* that maps from goal to goal such that $P \models_{\mathcal{M}_\mathcal{X}} \exists(G') \Leftrightarrow P \models_{\mathcal{M}_\mathcal{X}} \exists(G_{i+1})$.

In a generalized $\mathcal{M}_\mathcal{X}$ derivation, constraint solving is used as a tool for simplifying constraints and is encapsulated in the normal-form function ν. A derivation is a, possibly infinite, sequence of goals. We say that the derivation is *successful* if it is finite and the last goal contains no atoms, and *finitely-failed* if it is finite but the last goal has one or more atoms and condition 1 does not hold. Generalized $\mathcal{M}_\mathcal{X}$ derivation is not new. It has been implemented in CLP($\mathcal{R}$) for constraint simplification and output [10].

4.2 The Inference Engine

The structure of the engine resembles that of a standard structure-sharing Prolog interpreter and there is no need to elaborate further. Some changes to the unification algorithm, however, is needed to unify interval variables with other terms in CIAL. Additional binding mechanisms are defined.

A logical variable X becomes an *interval variable* when it is involved in such simple inequality constraints as "$X > 3, X \leq 6$," or such interval equalities as "$X + 2 * Y = Z, X * X = Y$." In the first example, we say that the interval $(3, 6]$ is *associated* with the variable X. Semantically, an interval variable is an ordinary logical variable. We distinguish interval variables from logical variables purely for implementation efficiency. Resembling domain variables in CHIP [5], an interval variable is represented as a variable with an associated interval. Interval variable can unify with other unbound logical variables, interval variables, or numbers. To unify an interval with an unbound variable, we simply bind the interval variable to the unbound variable. To unify two interval variables X and Y, we compute the intersection of their associated intervals. If the intersection J^I is non-empty, we choose one of X and Y (for efficiency reason, we choose the variable which does not require trailing, if possible), say X, and bind it to the other, Y in this case. Then we replace the associated interval of the chosen variable X by the intersection J^I. Otherwise, failure is reported. We treat a number as an interval variable, the associated interval which has the number as closed upper and lower bounds. An interval variable is *non-narrowable* if its associated interval is non-narrowable.

Like variables in finite-domain languages, each interval variable in CIAL keeps a constraint list to store the constraints in which this variable appears. If two interval variables are unified successfully, their constraint lists are merged and all related constraints are waken up.

4.3 Incrementality of the Linear Solver

Incremental execution capability is essential for solvers in constraint logic programming languages. CIAL is no exception. In the following, we present such an adaptation for the generalized Gaussian elimination procedure and its interaction with the non-linear solver.

Our algorithm is based on that of CLP($\mathcal{R}$) [11]. We present only the parts that differ from their counterparts in CLP($\mathcal{R}$): *detection of redundancy/inconsistency of newly added linear equation* and *the selection of non-parametric variables*.

All linear equations in the linear solver will be stored in parametric solved form $X = \sum_{r=1}^{n}(c_r \otimes T_r) \oplus c_{n+1}$, where X is a *non-parametric variable*, T_r is a *parametric variable* and c_r is a non-narrowable interval variable that we treat as constant. Assume we have a collection of consistent linear equations $(E_1, E_2, ...E_{n-1})$ in solved form and a new linear equation E_n is added. Let C_n be the result of substituting out all the non-parametric variables in E_n.

Detection of Inconsistency/Redundancy

Let C_n be in the form $0 = f(T_i)$.

1. If $f(T_i)$ does not contain 0, it implies that the new equation E_n is inconsistent with the stored constraints. Backtracking is needed.

2. If $f(T_i) = 0$, the new equality is implied by the stored constraints and this new constraint can be removed.

3. A linear constraint is said to be *fixed* if all its parametric variables (T_i) are non-narrowable. If C_n is fixed and $f(T_i)$ contains 0, E_n is also redundant.

Selection of Non-parametric Variables

The candidates of non-parametric variable must not have coefficients containing 0. If there is no variable satisfying this criterion, it implies C_n cannot be solved by generalized Gaussian elimination at this time and will be moved to non-linear solver.

1. If C_n contains new variable(s) that is not currently in the linear solver, choose one of them as the non-parametric variable.

2. If C_n contains only parametric variable(s), choose one which appears in the non-linear solver. This criterion is on the contrary to the corresponding one in CLP($\mathcal{R}$). CIAL hopes that non-linear constraints can be further narrowed with the aid of linear solver, while CLP($\mathcal{R}$) tries to avoid invocation of the inequality solver.

3. Otherwise, choose the parametric variable with maximum width because it will most probably be narrowed in interval propagation, in general.

In CIAL, backward substitution is an important step since it can help to eliminate variables and obtain sharper intervals for non-parametric variables. This can in turn re-activate constraints containing the non-parametric variables in the non-linear solver. Backward substitution is, however, also both time and memory consuming. In such cases as $\{X = Y + Z, Y = A\}$, the transformation to eliminate Y is fruitless computation. This kind of backward substitution should be delayed. We propose two optimizations here but they have not been implemented in the current version of CIAL.

- Backward substitution will not be performed between two constraints if they do not share any common parametric variable.

- A constraint will not be used for backward substitution if it contains an unbound parametric variable X (i.e. $X \in (-\infty, \infty)$) and X does not appear as parametric variable in the constraint being substituted.

4.4 Solver Interaction

There are two solvers in CIAL. The non-linear solver is based on interval narrowing; while the linear solver is based on generalized Gaussian elimination. The linear solver is only responsible for linear constraints. Non-linear constraints and inequalities belong to the non-linear solver. We explain in the following how the two solvers cooperate in one constraint solving step.

1. A new interval constraint will be decomposed into a linear constraint and a conjunction of primitive constraints. For example, the constraint $X^2 + 2 * Y > Z$ will be translated into three constraints, $X * X = T$, $T + 2 * Y = U$ and $U > Z$ **(A)**.

2. Linear constraints will be sent to the linear solver, processed by generalized interval Gaussian elimination and marked active **(B)**.

3. Non-linear constraints or inequalities will be sent to the non-linear solver and marked active **(C)**.

4. After all constraints in a derivation step are collected, the linear constraint solver will be invoked first.

5. The value of the non-parametric variables of active linear constraints will be re-calculated. If any of the variables is changed, the constraints in the non-linear solver which share that variable will be marked active **(D)**. Changing non-parametric variables will never activate other linear constraints since no non-parametric variables can appear as parametric variables in other linear constraints.

6. Apply interval narrowing to narrow parametric variables (*only*) in active linear constraints in step 5. This step is important for eliminating infeasible values of parametric variables in the case where the number of equality constraints is less than the number of unknown variables. Since interval propagation in Gaussian elimination is functional (from parametric variables to non-parametric variables), we employ interval narrowing to achieve maximum infeasible value elimination of parametric variables. The constraints in both solvers that share the modified variables will be marked active.

7. Repeat steps 5 to 6 until the linear solver contains no active linear constraints.

8. Active constraints in the non-linear solver will be waken up and the interval narrowing process will be invoked. If a variable is further narrowed and it is also the parametric variable in some linear constraints, those constraints will be marked active **(E)**. Note that no constraint will be moved from the non-linear solver to the linear solver even when a constraint becomes linear. It is because all constraints in the non-linear solver is primitive and will contain at most 3 variables. When one variable becomes non-narrowable, the value of a narrowable variable can be easily propagated to constraints in the linear solver.

9. Repeat steps 5 to 8 until both solvers contain no active constraints.

5 Prototype Implementation and Examples

In order to demonstrate the feasibility of our proposal, we have constructed a CIAL prototype. Since CIAL has much in common with CLP($\mathcal{R}$) on the surface, we decided to use CLP($\mathcal{R}$) as the backbone of our implementation and tried to adopt as much original CLP($\mathcal{R}$) code as possible. It turns out that only the Prolog engine part of CLP($\mathcal{R}$) can be re-used in our implementation. The solver interface and the two solvers are implemented from scratch. We also modify the unification algorithm of the Prolog engine to cope with unification between interval variables and other terms. The current implementation is based on CLP($\mathcal{R}$) Version 1.2 and consists of about 2800 extra lines of C code.

In the following, we compare CIAL with BNR Prolog v3.1.0, CLP(BNR) (or BNR Prolog v4.2.3), Echidna Version 0.947 beta, and CLP($\mathcal{R}$) Version

1.2 over three numerical examples of various types: analysis of a simple DC circuit, inconsistent simultaneous equations, and the famous Wilkinson polynomial [8]. The examples range from purely linear constraints, to a mixture of linear and non-linear constraints, and to purely non-linear constraints. BNR Prolog runs on an Apple Mac II ($\sim$ 2 VAX MIPS with 5MB Ram) and the other systems run on a SUN SPARCstation 10 model 30 ($\sim$ 49 VAX MIPS with 32MB Ram).

5.1 Analysis of DC Circuit

Consider the simple DC circuit in figure 2. We are interested in the currents passing through the resistors.

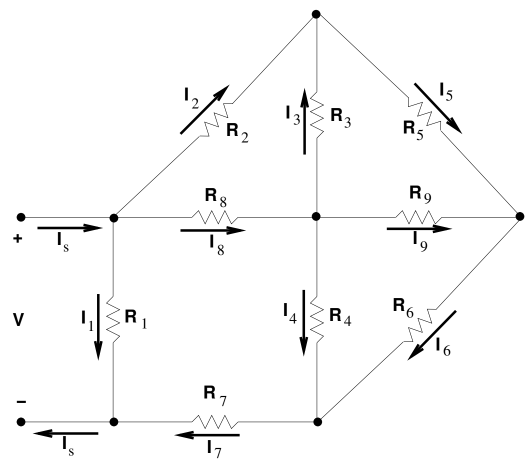


Figure 2: A Simple DC Circuit

Assume $V = 10$ volts and $R_i = i\ \Omega$ for $i = 1, 2, \ldots, 9$. The following system of linear equations are obtained from nodal and mesh analysis.

$$\left\{ \begin{array}{rcll} I_s - I_1 - I_2 - I_8 & = & 0 & , \qquad I_1 = 10 \\ -I_s + I_1 + I_7 & = & 0 & , \qquad 2I_2 - 3I_3 - 8I_8 = 0 \\ I_2 + I_3 - I_5 & = & 0 & , \qquad 3I_3 + 5I_5 - 9I_9 = 0 \\ -I_3 - I_4 + I_8 - I_9 & = & 0 & , \qquad -4I_4 + 6I_6 + 9I_9 = 0 \\ I_4 + I_6 - I_7 & = & 0 & , \qquad -I_1 + 4I_4 + 7I_7 + 8I_8 = 0 \\ I_5 - I_6 + I_9 & = & 0 & \end{array} \right.$$

Note that there are 11 linear equations but only 10 unknown variables. The redundant equation cannot be located in advance, however.

CIAL gives sharp results in 1.11s with 10 decimal places of accuracy. CLP($\mathcal{R}$) responds in less than 1/60 second. CLP($\mathcal{R}$)'s efficiency over CIAL is due to the fact that interval computation is time-consuming, especially for setting IEEE rounding directions and detecting exceptions. The solutions given by CLP($\mathcal{R}$) are, however, unsound. With initial guess $[-100, 100]$ for all variables, Echidna (in high precision mode) and BNR Prolog cannot terminate in 2 and 24 hours respectively by splitting 4 variables. CLP(BNR) cannot give any solution (except I_1) with width less than 100, although all variables are specified to split. This example demonstrates the inability of interval narrowing and splitting to solve even small system of linear constraints.

5.2 Inconsistent Simultaneous Equations

The following ad hoc constraint system is obviously inconsistent, since B and C should be equal with value either 1 or -2 according to the first three constraints. Neither value is, however, consistent with the initial bound of B.

$$\begin{cases} A + C & = & D \\ A + B & = & D \\ C(C+1) & = & 2 \\ A \in (0, \infty), B \in (-\infty, -5) & & \end{cases}$$

CLP(BNR) returns "yes." BNR Prolog does not terminate in 90 minutes. Echidna returns "yes" with default precision and exits abruptly with high precision. CLP($\mathcal{R}$) gives "maybe" with answer constraints.

CIAL can detect the inconsistency without using splitting but with the cooperation of 2 solvers. It is also interesting to find that none of CLP(BNR), BNR Prolog and Echidna can calculate the value of C without splitting, even when initial guess $[-100, 100]$ is given.

5.3 Wilkinson Polynomial

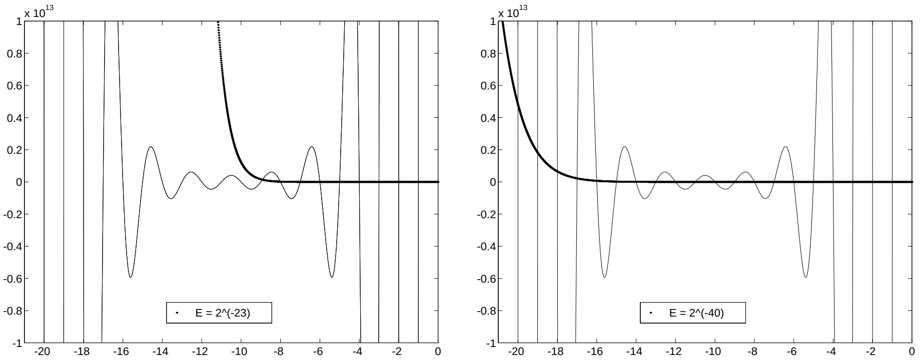
The final example is the famous Wilkinson polynomial equation. The problem is to find the real roots of the following equation.

$$\prod_{i=1}^{20}(X+i) + EX^{19} = 0$$

Let $E = 0$. The real roots of this unperturbed polynomial in the closed interval $[-20, -10]$ are -20,-19,-18,-17,-16,-15,14,-13,-12,-11,-10, respectively.

A slight perturbation of the polynomial by $E = 2^{-23}$ removes all roots in $[-20, -10]$, as shown in figure 3(a) which consists of the curves $Y = \prod_{i=1}^{20}(X+i)$ and $Y = -EX^{19}$. CIAL returns no answers as expected.

When $E = 2^{-40}$ (Figure 3(b)), CIAL finds all solutions with 10 decimal place accuracy by employing interval splitting. Echidna cannot give any solution in 15 minutes for $E = 2^{-23}$. For $E = 2^{-40}$, it aborts abnormally for default precision and does not terminate in 15 minutes for higher precision. Both BNR Prolog and CLP(BNR) return similar results as those of CIAL in both cases. CLP($\mathcal{R}$) cannot handle non-linear systems.



(a) (b)

Figure 3: The curves $y = \prod_{i=1}^{20}(X+i)$ and $y = -EX^{19}$

5.4 Discussion

The CIAL solvers subsume the Gaussian elimination and the interval narrowing technology. CLP($\mathcal{R}$) cannot handle non-linear constraints and interval narrowing fails to handle even small system of linear constraints, as shown in section 5.1. Thus CLP($\mathcal{R}$), BNR Prolog, CLP(BNR) and Echidna are deficient in solving mixtures of linear and non-linear constraints. A simple example can be obtained by adding a constraint $I_x(I_x - 1) = I_s$ to the system in section 5.1.

6 Concluding Remarks

The contribution of this paper is three-fold. First we have derived a practical linear interval equality solver, which is based on generalized interval arithmetic and Gaussian elimination. The solver is adapted for incremental execution. Second, we show how a linear interval solver can be incorporated into ICLP($\mathcal{R}$), which has already had a non-linear solver based on interval narrowing. We have derived an interaction scheme for the two solvers to cooperate towards solving a system of numerical constraints. The resulting system is CIAL. Third, we have constructed a prototype of CIAL and compared it with existing interval constraint solving languages. CIAL has been shown to be all rounded and can solve more classes of problems than any other existing systems when used alone.

There are a few directions for future work. First, the linear solver can only handle linear equalities. It would be interesting to investigate how linear inequalities can be accommodated. Second, it is worthwhile to study other linear interval constraint solving methods, such as preconditioned Gaussian elimination and preconditioned Gauss-Seidel. Third, the CIAL architecture is rudimentary. Further optimizations, such as the techniques used for CLP($\mathcal{R}$), might be applicable to CIAL. Fourth, on the theoretical side, it will be interesting to study the level of interval consistency attainable in generalized Gaussian elimination.

Acknowledgement

We are indebted to Maarten van Emden, Tak Wai Lee, Spiro Michaylov, Bo Ming Tong, and Roland Yap for numerous discussions and critical comments. We thank the anonymous referees for their insightful comments. Access to CLP($\mathcal{R}$), BNR Prolog, CLP(BNR), and Echidna is gratefully acknowledged. This research was supported in part by a Direct Grant (A/C 220500500) and a RGC Earmarked Grant (A/C 221500260), both by the Hong Kong University and Polytechnic Grants Committee.

References

[1] G. Alefeld and J. Herzberger. *Introduction to Interval Computations*. Academic Press, New York, 1983. Translated by J. Rokne from *Elemente der Intervallrechnung*.

[2] F. Benhamou, D. McAllester, and P. Van Hentenryck. CLP(Intervals) revisited. In *Logic Programming: Proceedings of the 1994 International Symposium*, 1994.

[3] F. Benhamou and W.J. Older. Applying interval arithmetic to real, integer and boolean constraints. *(to appear) Journal of Logic Programming*, 1994.

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

[5] 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.

[6] E.R. Hansen. A generalized interval arithmetic. In K.L. Nickel, editor, *Interval Mathematics: Proceedings of the International Symposium*, pages 7–18. Springer-Verlag, 1975.

[7] W.S. Havens, S. Sidebottom, G. Sidebottom, J. Jones, M. Cuperman, and R. Davison. Echidna constraint reasoning system: Next-generation expert system technology. Technical Report CSS-IS TR 90-09, Centre for Systems Science, Simon Fraser University, Burnaby, B.C., Canada, 1990.

[8] H. Hong. Non-linear real constraints in constraint logic programming. In *Proceedings of the Second International Conference on Algebraic and Logic Programming*, pages 201–212, 1992.

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

[10] J. Jaffar, M.J. Maher, P.J. Stuckey, and R.H.C. Yap. Output in CLP($\mathcal{R}$). In *Proceedings of the International Conference on Fifth Generation Computer Systems 1992 (to appear)*, Tokyo, Japan, June 1992.

[11] J. Jaffar, S. Michaylov, P.J. Stuckey, and R.H.C. Yap. The CLP($\mathcal{R}$) language and system. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, July 1992.

[12] J.H.M. Lee and M.H. van Emden. Adapting CLP($\mathcal{R}$) to floating-point arithmetic. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1992*, pages 996–1003, Tokyo, Japan, June 1992.

[13] J.H.M. Lee and M.H. van Emden. Interval computation as deduction in CHIP. *Journal of Logic Programming*, 16(3 & 4):255–276, 1993.

[14] O. Lhomme. Consistency techniques for numeric CSPs. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, 1993.

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

[16] 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.

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

[18] 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.

[19] G. Sidebottom and W.S. Havens. Hierarchical arc consistency for disjoint real intervals in constraint logic programming. *Computational Intelligence*, 8(4), 1992.

[20] J.H. Wilkinson. *The Algebraic Eigenvalue Problem*. Oxford University Press, 1965.