Ann Math Artif Intell (2009) 55:295–354
DOI 10.1007/s10472-009-9129-6

Enhancing numerical constraint propagation using multiple inclusion representations

Xuan-Ha Vu · Djamila Sam-Haroud · Boi Faltings

Published online: 18 March 2009

Abstract Building tight and conservative enclosures of the solution set is of crucial importance in the design of efficient complete solvers for *numerical constraint satisfaction problems* (NCSPs). This paper proposes a novel generic algorithm enabling the cooperative use, during constraint propagation, of multiple enclosure techniques. The new algorithm brings into the constraint propagation framework the strength of techniques coming from different areas such as interval arithmetic, affine arithmetic, and mathematical programming. It is based on the *directed acyclic graph* (DAG) representation of NCSPs whose flexibility and expressiveness facilitates the design of fine-grained combination strategies for general factorable systems. The paper presents several possible combination strategies for creating practical instances of the generic algorithm. The experiments reported on a particular instance using interval constraint propagation, interval arithmetic, affine arithmetic, and linear programming illustrate the flexibility and efficiency of the approach.

Keywords Interval constraint propagation · Branch and prune · Interval arithmetic · Affine arithmetic

Mathematics Subject Classification (2000) 68T20

X.-H. Vu
Cork Constraint Computation Centre, University College Cork,
14 Washington Street West, Cork, Ireland
e-mail: vuxuanha@yahoo.com

D. Sam-Haroud (✉) · B. Faltings
Artificial Intelligence Laboratory, Ecole Polytechnique Fédérale de Lausanne (EPFL),
Batiment IN, Station 14, 1015 Lausanne, Switzerland
e-mail: jamila.sam@epfl.ch

B. Faltings
e-mail: boi.faltings@epfl.ch

1 Introduction

A numerical constraint satisfaction problem (NCSP) consists of variables that may take on a value from a given set of values from $\mathbb{R}$ and of constraints that restrict the possible value combinations between variables. A solution is a set of assignments of values to variables so that all the constraints of the problem are satisfied. Constraints that bear on numerical variables are equations and inequalities expressed using arithmetic expressions. These expressions may be simple or complex, linear or non-linear and may involve elementary transcendental functions. Chemical or mechanical models, process descriptions, building codes, or cost restrictions are most often expressed using this type of constraints.

A solution technique for NCSPs is said to be *complete* if it is able to find a solution if there is one or else prove that there are no solutions to the problem. Only complete solvers can be relied upon to provide all the relevant alternatives, to avoid any inconsistencies and to guarantee that all the constraints—e.g. security or tolerance criteria—are satisfied [35].

The most commonly used strategy for the *complete solving* of numerical CSP is *branch-and-prune*, which interleaves branching steps with pruning steps. The role of *branching* is to divide a given problem into subproblems whose union is equivalent to the initial one with respect to the solution set. The role of *pruning* is to reduce the search space by cutting off the regions that violate some of the constraints. Domain reduction via constraint propagation algorithms [3, 6, 7, 11, 16, 42], which reduces the domains of variables without discarding any solution, is a widely used pruning technique. Constraint propagation has been the object of intensive research over the last twenty years.

Building tight and conservative enclosures of the solution set for a given problem is the key behind efficient domain reduction. Interval-analytic methods or linear relaxations constitute the "usual machinery" employed for computing conservative enclosures. In the reminder, we call *inclusion representation* any kind of representation providing a conservative enclosure of the solution set for a numerical CSP.

This paper builds on the idea that *combining multiple inclusion representations* can significantly enhance the quality of domain reduction, as suggested by the example of Fig. 1.

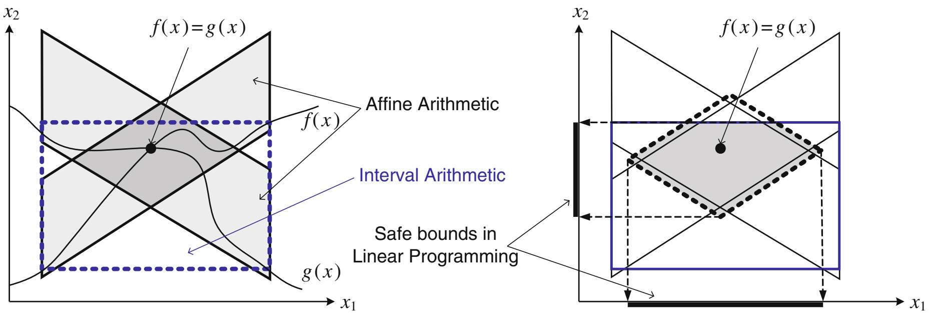


Fig. 1 Combining interval arithmetic, affine arithmetic, and safe linear programming provides a tighter enclosure of the solution set (i.e., a smaller bounding box)

We propose a novel generic algorithm called CIRD,[1] which enables the cooperative use of multiple inclusion representations during constraint propagation. The CIRD scheme is based on the *direct acyclic graph* (DAG) representation of NCSPs [38] and is applicable to virtually any factorable constraint system.

We propose concrete instances of the generic algorithm which involve inclusion representations coming from interval arithmetic, affine arithmetic, constraint propagation, and linear programming (see Section 5).

Our experiments (Section 6) show that the CIRD scheme is able to provide propagation techniques superior in performance and quality to the state-of-the-art interval-based constraint propagators. It is capable of outperforming some recent mathematical and constraint programming techniques, even when specifically designed to solve particular constraint systems.

2 Background and definition

In the following, we assume the reader familiar with the foundation of constraint satisfaction and interval arithmetic. Extended introductions can be found in the following:

- Apt [2] for constraint satisfaction techniques;
- Moore [33, 34], and [1] for interval analysis;
- Neumaier [37] for interval methods applied to systems of equations;
- Hansen and Walster [13] for interval methods applied to optimization problems;
- Jaulin et al. [16] for real-world applications.

An overview of standard affine arithmetic is given in Appendix A. As an incidental contribution of the paper, we propose in this section a revised form of interval arithmetic which is more accurate for particular computations that we focus on.

In this paper, we will target NCSPs expressed using *factorable* expressions and build on the directed acyclic graph (DAG) representation of constraint systems to extend the well known **HC4** algorithm [4].

The reminder of this section gives the necessary background and definition related to these concepts.

2.1 Factorable numerical constraint satisfaction problems

In practice, most functions for modeling real-world applications can be composed of elementary operations or functions such as $+$, $-$, $*$, $/$, sqr, exp, ln, and sin. Such operations or functions are said to be *factorable*. They play a significant role in algorithms for solving not only numerical CSPs but also other numerical problems such as optimization problems and automatic differentiation computations. For completeness, we recall in this section the concepts of *factorability*.

Definition 1 (Elementary Operations) Denote by $\mathbb{E}_1$ the set of standard *elementary unary functions*, namely, $\mathbb{E}_1 = \{\text{abs, sqr, sqrt, exp, ln, sin, cos, arctan}\}$. Denote by $\mathbb{E}_2$ the set of standard *elementary binary operations*, namely, $\mathbb{E}_2 = \{+, -, *, /, \hat{}\,\}$.

[1]CIRD stands for "Combining Inclusion Representation using DAGs".

If an expression is recursively composed of standard elementary operations and functions, it is called an *arithmetic expression* [37, p. 13] or a *factorable expression* [26, 27]. In this paper, we extend the concept of an arithmetic expression to include other elementary operations.

Definition 2 (Factorable Expression) Let R be a nonempty set, $\{x_1, \dots, x_n\}$ a set of variables taking values in R, F a finite set of *elementary operations* of the form $f : R^k \to R$. An expression is said to be *factorable* in the (formal) variables $x_1, \dots, x_n$ using operations in F if it is a member of the minimal set $\mathcal{F} \equiv \mathcal{F}(R, F; x_1, \dots, x_n)$ satisfying the following *composition rules*:

1. $R \subseteq \mathcal{F}$;
2. $x_i \in \mathcal{F}$ for all $i = 1, \dots, n$;
3. If $f : R^k \to R$ is in F and $e_1, \dots, e_k \in \mathcal{F}$; then $f(e_1, \dots, e_k) \in \mathcal{F}$.

Definition 3 We denote $\mathcal{E}(x_1, \dots, x_n) \equiv \mathcal{F}(\mathbb{R}, \mathbb{E}_1 \cup \mathbb{E}_2; x_1, \dots, x_n)$.

If an expression E is factorable in variables $X \equiv \{x_1, \dots, x_n\}$ using operations in F as in Definition 2 and if either $F = \mathbb{E}_1 \cup \mathbb{E}_2 \wedge R = \mathbb{R}$ holds or F is known from the context, then we say that E is factorable in X.

For example, the expression $f(x, y) = 2x^y + \sin x$ is factorable using the elementary operations in $\{+, *, \text{^}, \sin\}$. The composition is given as follows: $f_1 = x\text{^}y\ (\equiv x^y)$, $f_2 = 2 * f_1$, $f_3 = \sin(x)$, and $f = f_2 + f_3$.

Definition 4 (Factorable Function) A function f is said to be *factorable* in variables $x_1, \dots, x_n$ using the operations in a finite set F of elementary operations if it can be expressed by an expression that is factorable in variables $x_1, \dots, x_n$ using elementary operations in F. If $F = \mathbb{E}_1 \cup \mathbb{E}_2$ or F is known from the context, we could just say for short that f is factorable, namely, in variables $x_1, \dots, x_n$.

For example, the function $f(x, y) = 2x^y + \sin x$ is factorable using the operations in $\{+, *, \text{^}, \sin\}$, and is not factorable using only the operations in $\{+, *, \text{^}\}$.

The factorability can be defined for constraints as follows.

Definition 5 (Factorable Constraint) A constraint is said to be *factorable* in variables $x_1, \dots, x_n$ using a finite set F of elementary operations if it can be expressed as a relation involving expressions that are factorable in variables $x_1, \dots, x_n$ using operations in F. In the composition of a factorable constraint, each constraint representing an elementary operation is called a *primitive constraint*.

In this paper, we restrict, for simplicity, the relation in a factorable constraint to be $\leq, <, \geq, >, =$ or $\neq$. For example, the constraint $2x^y + \sin x \leq 0$ is factorable in variables x and y using the operations in $\{+, *, \text{^}, \sin\}$. Its primitive constraints are $f_1 = x\text{^}y\ (\equiv x^y)$, $f_2 = 2 * f_1$, $f_3 = \sin(x)$, and $f_2 + f_3 \leq 0$; where x and y are *initial variables*, and f_1, f_2 and f_3 are *auxiliary variables*.

The factorability can also be defined for a CSP as follows.

Definition 6 A CSP is said to be *factorable* (using a set F of elementary operations) if all of its constraints are factorable (using operations in F).

2.2 Numerical constraint propagation

As mentioned in the introduction, the usual strategy for the complete solving of NCSPs is *branch-and-prune*. The pruning steps often compute *domain reduction* for the variable using *interval constraint propagation* algorithms (see [3, 6, 7, 11, 16, 42]). These algorithms enforce particular consistency properties such as *hull consistency* [6, 7], *k*B-consistency [23] or *box consistency* [5] on the constraint network. In particular, [4] proposed an algorithm called **HC4** which achieves hull consistency for individual constraints, and then propagates the reduction of the variables' domains from constraint to constraint. The **HC4** algorithm represents each individual constraint by a tree whose nodes and edges represent subexpressions and computational flows, respectively. Each node of the tree is associated with the possible range (an enclosure of the exact range) of the corresponding subexpression.

In order to reduce the variables' domains of a given constraint, the technique recursively performs a sequence of *forward evaluations* and then a sequence of *backward projections* on the whole tree representing the constraint. These two steps respectively compute the possible ranges of nodes based on their children' and parents' ranges. When several constraints are involved, the **HC4** algorithm performs forward evaluations and backward projections individually on each constraint, and then propagates the reduction of the variables' domains from tree to tree by using a variant of *arc consistency*, **AC3** [24].

2.3 DAG representations for numerical CSPs

The fact that each constraint is propagated individually is one of the main limitations of the **HC4** algorithm. In particular, the fact that some subexpressions are shared by several constraints is not properly taken into account : **HC4** is unable to propagate domain reduction of a given subexpression immediately to the other parts of the constraints system where the same subexpression appears.

Recently, a fundamental framework for interval analysis on *directed acyclic graphs* (DAGs) has been proposed in [38] which overcomes this limitation. The authors suggested to replace trees with DAGs (see Fig. 3) for representing the constraint system and showed how to perform forward evaluations and backward projections using this particular representation. The shift to DAGs potentially reduces the amount of computation on common subexpressions shared by constraints, and explicitly relates constraints to constraints in the natural way they are composed. This enhances the propagation process.

The CIRD algorithm we propose in this paper builds on the schema proposed by [38]. The detailed description of DAG representations can be found in [39], among which we recall the following fundamentals:

Definition 7 (Parent, Child, Ancestor, Descendant) Consider a directed acyclic multigraph $\mathcal{G} \equiv (V, E, f)$. Let v_1 and v_2 be two vertices in V. We say that v_2 is a *parent* of v_1 and that v_1 is a *child* of v_2 if there exists an edge $e \in E$ such that $f(e) = (v_1, v_2)^{\mathrm{T}}$. The set of all parents (respectively, all children) of a vertex $v \in V$ is denoted by $\mathsf{parents}(v)$ (respectively, $\mathsf{children}(v)$). We say that v_2 is an *ancestor* of v_1 and that v_1 is a *descendant* of v_2 if there exists a directed path from v_1 to v_2. The set of all ancestors (respectively, all descendants) of a vertex $v \in V$ is denoted by $\mathsf{ancestors}(v)$ (respectively, $\mathsf{descendants}(v)$).

Theorem 1 *For every directed acyclic multigraph (V, E, f), there exists a total order $\preceq$ on the vertices V such that for every $v \in V$ and every $u \in \mathsf{ancestors}(v)$, we have $v \preceq u$.*

For convenience, we agree on the following notation and convention related to DAGs.

Notation 2 *Each node* **N** *in the DAG representation is associated with an interval, denoted as $\tau_{\mathbf{N}}$ and called the* node range *of* **N***, in which the exact range of the associated subexpression must lie.* **N** *is also associated with a real variable, denoted by $\vartheta_{\mathbf{N}}$, that represents the value of the subexpression represented by* **N**.

For efficiency and compactness, the standard elementary operations in the DAG representation are replaced with more general operations. For example, multiple applications of binary elementary operations such as $\{x + y,\ x - y,\ x + a,\ a + x,\ x - a,\ a - x,\ ax\}$ are replaced with a k-ary operation $a_0 + a_1 x_1 + \cdots + a_k x_k$, which is interpreted as a k-ary operation $+$ (see Fig. 2a), where $1 \leq k \in \mathbb{N}$. Similarly, multiple applications of the binary multiplication $x * y$ are replaced with a k-ary multiplication $a_0 * x_1 * \cdots * x_k$, which is interpreted as the k-ary operation $*$ (see Fig. 2b), where $2 \leq k \in \mathbb{N}$. In general, each edge of a DAG is associated with the coefficient related to the operation represented by its target. When not specified in figures, this coefficient equals to 1. The other constants involving an operation are stored at the node representing the operation (see Fig. 2). As a result, the DAG representation no longer has nodes representing constants as it is the case in the tree representation.

In fact, we use multigraphs instead of simple graphs for DAGs because some particular operations can take the same input more than once. For example, the expression x^x can be represented by the binary power operation x^y without introducing a new unary operation x^x. In all cases, a normal directed acyclic graph is sufficient to represent a numerical CSP (NCSP), provided that we introduce new elementary operations such as the unary operation x^x. The ordering of edges is needed for non-commutative operations like the division. For convenience, a *ground node*, called **G**, is added to each DAG representation to be the parent of all nodes that represents the constraints. The ground node can be interpreted as the logical `AND` operation.

Let us consider the following constraint system

$$\begin{cases} \sqrt{x} + 2\sqrt{xy} + 2\sqrt{y} \leq 7, \\ 0 \leq x^2\sqrt{y} - 2xy + 3\sqrt{y} \leq 2, \\ x \in [1, 16],\ \ y \in [1, 16]. \end{cases}$$

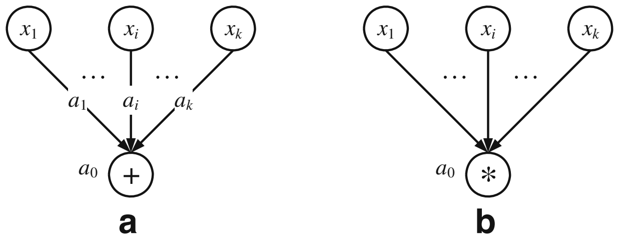


Fig. 2 A node and its computational flows in a DAG representation (**a**, **b**)

It can be written as follows:

$$\begin{cases} \sqrt{x} + 2\sqrt{xy} + 2\sqrt{y} \in [-\infty, 7], \\ x^2\sqrt{y} - 2xy + 3\sqrt{y} \in [0, 2], \\ x \in [1, 16], \ \ y \in [1, 16]. \end{cases} \tag{1}$$

The DAG representation of the constraint system (1) is depicted in Fig. 3. The two constraints of (1) are represented by two nodes $\mathbf{N}_9$ and $\mathbf{N}_{10}$. The two initial variables, x and y, are represented by two nodes $\mathbf{N}_1$ and $\mathbf{N}_2$, respectively. The sequence $(\mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_{10})$ of nodes given in Fig. 3 is an example of an ordering (see Theorem 1).

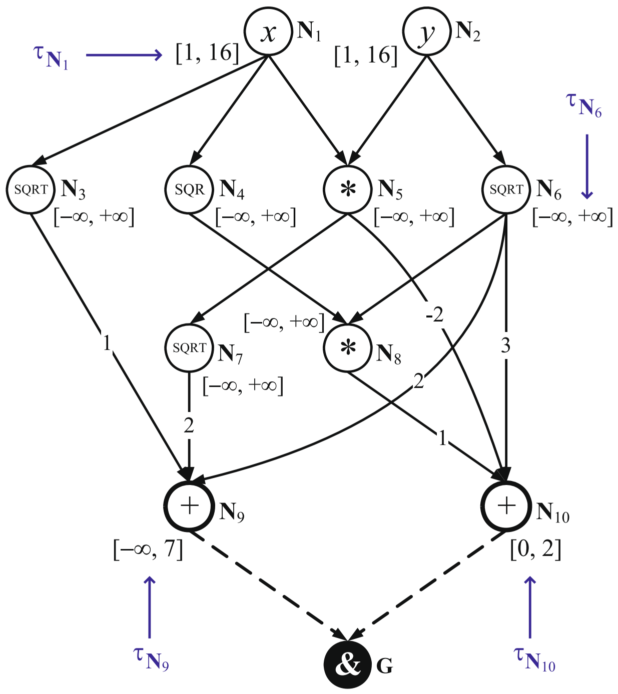
For the same constraint system, the DAG representation is much more concise than the equivalent tree representation used in the **HC4** algorithm.

2.4 Revised affine arithmetic

Affine arithmetic [9] is an extension of interval arithmetic which keeps track of correlations between computed and input quantities. In particular, a real-valued quantity x is represented by an *affine form* defined as follows

$$x \equiv x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n, \tag{2}$$

where $x_0, \ldots, x_n$ are real coefficients and $\varepsilon_1, \ldots, \varepsilon_n$ are *noise variables* (originally called *noise symbols* [9]) taking values in the real interval $[-1, 1]$. Similarly to interval arithmetic, affine arithmetic also enables the use of rounded floating-point arithmetic


x $\mathbf{N}_1$
y $\mathbf{N}_2$
$\tau_{\mathbf{N}_1}$
[1, 16]
[1, 16]
$\tau_{\mathbf{N}_6}$
SQRT $\mathbf{N}_3$ [−∞, +∞]
SQR $\mathbf{N}_4$ [−∞, +∞]
∗ $\mathbf{N}_5$ [−∞, +∞]
SQRT $\mathbf{N}_6$ [−∞, +∞]
SQRT $\mathbf{N}_7$ [−∞, +∞]
∗ $\mathbf{N}_8$ [−∞, +∞]
1
2
2
-2
3
1
+ $\mathbf{N}_9$ [−∞, 7]
+ $\mathbf{N}_{10}$ [0, 2]
$\tau_{\mathbf{N}_9}$
$\tau_{\mathbf{N}_{10}}$
& G
Fig. 3 The DAG representation of the constraint system (1)

to construct *rigorous enclosures* for the ranges of operations and functions [40]. In affine arithmetic, affine operations such as $\alpha x + \beta y + \gamma$ (where $\alpha, \beta, \gamma \in \mathbb{R}$) are exactly obtained, except the rounding errors, by the formula

$$\alpha x + \beta y + \gamma = (\alpha x_0 + \beta y_0 + \gamma) + \sum_{i=1}^{n} (\alpha x_i + \beta y_i)\varepsilon_i \tag{3}$$

However, non-affine operations can only be computed by approximations. In general, the exact result of a non-affine operation has the form $f^*(\varepsilon_1, \ldots, \varepsilon_n)$, where f^* is a nonlinear function. In practice, this result is then approximated by an affine function $f^a(\varepsilon_1, \ldots, \varepsilon_n) = z_0 + z_1\varepsilon_1 + \cdots + z_n\varepsilon_n$. A new term $z_k\varepsilon_k$ is used to represent the difference between f^* and f^a, hence, the result has the affine form

$$z \equiv z_0 + z_1\varepsilon_1 + \cdots + z_n\varepsilon_n + z_k\varepsilon_k, \tag{4}$$

where the maximum absolute error z_k satisfies

$$z_k \geq \sup\left\{|f^*(\varepsilon) - f^a(\varepsilon)| : \forall\varepsilon = (\varepsilon_1, \ldots, \varepsilon_n) \in [-1, 1]^n\right\}.$$

An important goal is to keep the maximum absolute error as small as possible. This is a subject of *Chebyshev approximation theory*—a well-developed field with a vast literature.

In its standard version, affine arithmetic has several performance limitations when used in long-running computations.

First, the number of noise variables grows quickly during computations since each nonlinear operation adds a new noise variable. In general, the cost of operations in affine arithmetic heavily depends on the number of noise variables (e.g., linearly or quadratically). The computations in which affine arithmetic is used to generate linear relaxations suffer a lot from this drawback. Second, standard affine forms are not capable of handling half-lines like $[-\infty, a]$ and $[a, +\infty]$, while this is needed in many computation methods such as constraint propagation and exhaustive search.

Inspired by the ideas in [18, 19, 30], we propose a revised affine form similar to (4) but the new term $z_k\varepsilon_k$ is replaced by an accumulative error $[-e_z, e_z]$ which represents the maximum absolute error z_k of non-affine operations. In other words, the *revised affine form* of a real-valued quantity $\hat{x}$ is defined as

$$\hat{x} \equiv x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n + e_x[-1, 1], \tag{5}$$

which consists of two separated parts: the standard affine part of length n, and the interval part. Where the magnitude of the accumulative error, $e_x \geq 0$, is represented by the interval part. That is, for each value x of the quantity $\hat{x}$ (say $x \in \hat{x}$), there exist $\varepsilon_x \in [-1, 1]$, $\varepsilon_i \in [-1, 1]$ (for all $i = 1, \ldots, n$) such that $x = x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n + e_x\varepsilon_x$. We then say it is of length n. The affine operation $\hat{z} \equiv \alpha\hat{x} + \beta\hat{y} + \gamma$ is now defined as (where $\varepsilon_0 \equiv 1$)

$$\hat{z} \equiv \gamma + \sum_{i=0}^{n} (\alpha x_i + \beta y_i)\varepsilon_i + (|\alpha|e_x + |\beta|e_y)[-1, 1] \tag{6}$$

Note that during computations the lengths of revised affine forms will never exceed the number of noise symbols present at the beginning, i.e., the number of variables of the input constraint system.

We refer the reader to Appendix B for a detailed description of this arithmetic and its theoretical foundation.

3 Generalization of inclusion concepts

3.1 Inclusion representation

We start by generalizing the concepts related to *interval forms*, as defined in interval arithmetic. The objective is to provide a common view of different possible kind of conservative enclosures. This will facilitate the presentation of our generic constraint propagation scheme, presented in Section 4.

Definition 8 (Inclusion Representation) Given a set $\mathcal{A}$, a pair $\mathcal{I} \equiv (\mathcal{R}, \mu)$, where $\mathcal{R}$ is a nonempty set and μ is a function from $\mathcal{R}$ to $2^{\mathcal{A}}$, is called an *inclusion representation* of $\mathcal{A}$ if there exists a function $\zeta : 2^{\mathcal{A}} \to \mathcal{R}$ such that $\mu(\zeta(\emptyset)) = \emptyset$ and $\forall S \subseteq \mathcal{A} : S \subseteq \mu(\zeta(S))$. In this case, each $T \in \mathcal{R}$ is called a *representation object* in $\mathcal{I}$ (or in $\mathcal{R}$), ζ is called the *representing function* of $\mathcal{I}$, and μ is called the *evaluating function* of $\mathcal{I}$.

The function ζ in the previous definition expresses the fact that each subset of $\mathcal{A}$ is "included" in at least one representation object of $\mathcal{R}$. The next definition identifies a special class of inclusion representations used in Section 4.

Definition 9 (Real Representation) Let $\mathcal{I} \equiv (\mathcal{R}, \mu)$ be an inclusion representation of the real set $\mathbb{R}$. It is called a *real inclusion representation* of $\mathbb{R}$ if each representation object $T \in \mathcal{R}$ is a tuple consisting of real numbers, and the value of μ at T can be represented as

$$\mu(T) \equiv \big\{ f_T(V_T) \mid V_T \in D_T[V_T] \big\}, \tag{7}$$

where D_T (with T as a tuple of parameters) is a sequence of the domains of a finite sequence V_T of variables and other auxiliary variables, $D_T[V_T]$ denotes the subsequence of the domains of V_T in D_T, and f_T is a real-valued function on variables V_T. The representation (7) is called a *real representation* of μ.

The domains in D_T can be explicitly given by constant domains such as an interval $[a, b]$, or implicitly given by constraints. If D_T is given by

$$D_T = \bigcap_{i=1}^{m} \big\{ f_{i,T}(V_{i,T}) \mid V_{i,T} \in D_{i,T}[V_{i,T}] \big\}, \tag{8}$$

and f_T is an identity function, we then have

$$\mu(T) \equiv \big\{ x \mid x \in D_T[x] \big\} = \bigcap_{i=1}^{m} \big\{ f_{i,T}(V_{i,T}) \mid V_{i,T} \in D_{i,T}[V_{i,T}] \big\}. \tag{9}$$

Hereafter, we give some examples to illustrate the concept of real inclusion representation.

Example 1 Obviously, the standard representation of reals is directly equivalent, although not exactly, to a real inclusion representation of $\mathbb{R}$, where $T=(x)$, $V_T=(x)$, f_T is an identity function, and $D_T=\{x\}$.

Example 2 It is easy to see that the representation of intervals in the form

$$a \le x \le b \tag{10}$$

is equivalent to a real inclusion representation of the form (7), called the *interval representation*, by defining

$$T = (a, b) \in \mathbb{R}^2, \tag{11a}$$

$$V_T = (x), \tag{11b}$$

$$D_T = [a, b], \tag{11c}$$

$$f_T(V_T) = x, \tag{11d}$$

$$\mu(T) \equiv \{x \mid x \in [a, b]\}. \tag{11e}$$

The function f_T in (11) is an identity function.

Example 3 The union form of intervals can also be viewed as a real inclusion representation. For example, the union $\bigcup_{i=1}^{m}[a_i, b_i]$ can be interpreted by

$$T = (a_1, b_1, \ldots, a_m, b_m), \tag{12a}$$

$$V_T = (x), \tag{12b}$$

$$D_T = \bigcup_{i=1}^{m}[a_i, b_i], \tag{12c}$$

$$f_T(V_T) = x, \tag{12d}$$

$$\mu(T) \equiv \left\{ x \;\middle|\; x \in \bigcup_{i=1}^{m}[a_i, b_i] \right\}. \tag{12e}$$

The function f_T in (12) is an identity function.

Example 4 The affine forms (2), namely $\hat{x} \equiv x_0 + x_1\epsilon_1 + \cdots + x_n\epsilon_n$, can also be seen as a real inclusion representation of the form (7), called the *standard affine representation*, by defining that

$$T = (x_0, \ldots, x_n), \tag{13a}$$

$$V_T = (\epsilon_1, \ldots, \epsilon_n), \tag{13b}$$

$$D_T = [-1, 1]^n, \tag{13c}$$

$$f_T(V_T) = x_0 + \sum_{i=1}^{n} x_i\epsilon_i, \tag{13d}$$

$$\mu(T) \equiv \left\{ x_0 + \sum_{i=1}^{n} x_i\epsilon_i \;\middle|\; (\epsilon_1, \ldots, \epsilon_n) \in [-1, 1]^n \right\}. \tag{13e}$$

The function f_T in (13) is a *linear function* on V_T. Another real representation of the form (7) for the above real inclusion representation is defined as follows:

$$T = (x_0, \dots, x_n), \tag{14a}$$

$$V_T = (x), \tag{14b}$$

$$D_T = \left\{ x_0 + \sum_{i=1}^{n} x_i \epsilon_i \;\middle|\; (\epsilon_1, \dots, \epsilon_n) \in [-1, 1]^n \right\}, \tag{14c}$$

$$f_T(V_T) = x, \tag{14d}$$

$$\mu(T) \equiv \{x \mid x \in D_T\}, \tag{14e}$$

where D_T is implicitly given via its variables.

Example 5 Similarly to the affine forms, the revised affine forms defined by (5) can also be seen as a real inclusion representation of the form (7), called the *revised affine representation*, by defining that

$$T = (x_0, \dots, x_n, e_x), \tag{15a}$$

$$V_T = (\epsilon_1, \dots, \epsilon_n, \epsilon_x), \tag{15b}$$

$$D_T = [-1, 1]^{n+1}, \tag{15c}$$

$$f_T(V_T) = x_0 + \sum_{i=1}^{n} x_i \epsilon_i + e_x \epsilon_x, \tag{15d}$$

$$\mu(T) \equiv \left\{ x_0 + \sum_{i=1}^{n} x_i \epsilon_i + e_x \epsilon_x \;\middle|\; (\epsilon_1, \dots, \epsilon_n, \epsilon_x) \in [-1, 1]^{n+1} \right\}. \tag{15e}$$

The function f_T in (15) is a *linear function* on $(n + 1)$ variables in V_T.

Example 6 The *Kolev affine form* (see Definition 15), a generalization of interval and affine form, is also a real inclusion representation. It can be defined as an affine function on the variables κ_i:

$$\tilde{x} = c_x + \sum_{i=1}^{n} x_i \kappa_i + \kappa_x, \quad \kappa_i \in \mathbf{v}_i, \;\; \kappa_x \in \mathbf{v}_x, \tag{16}$$

where $\mathbf{v}_i \equiv [-v_i, v_i]$ (for all $i = 1, \dots, n$) and $\mathbf{v}_x \equiv [-v_x, v_x]$ are symmetric intervals, x_i (for all $i = 1, \dots, n$) are real coefficients, and $c_x \in \mathbb{R}$. Similarly to revised affine forms, Kolev affine forms can also be interpreted as a real inclusion representation (7), called the *Kolev affine representation*, by defining that

$$T = (c_x, x_1, \dots, x_n, v_1, \dots, v_n, v_x), \tag{17a}$$

$$V_T = (\kappa_1, \dots, \kappa_n, \kappa_x), \tag{17b}$$

$$D_T = [-v_1, v_1] \times \cdots \times [-v_n, v_n] \times [-v_x, v_x], \tag{17c}$$

$$f_T(V_T) = c_x + \sum_{i=1}^{n} x_i \kappa_i + \kappa_x, \tag{17d}$$

$$\mu(T) \equiv \left\{ c_x + \sum_{i=1}^{n} x_i \kappa_i + \kappa_x \;\middle|\; (\kappa_1, \dots, \kappa_n, \kappa_x) \in D_T \right\}. \tag{17e}$$

The function f_T in (17) is a *linear function* on $(n+1)$ variables in V_T.

Note 1 In practice, a representation object T in the above affine forms often contains many zero coefficients; hence, we should store only nonzero coefficients and their indices instead of all coefficients. For example, an affine form $0.1 + 2.1\epsilon_2 + 9.1\epsilon_9$ should be stored in T by (0.1, 2.1, 9.1; 2, 9) instead of (0.1, 0.0, 2.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 9.1).

Example 7 Hansen's generalized interval [12] is given as follows

$$\tilde{x} = [\underline{c}_x, \overline{c}_x] + \sum_{i=1}^{n} [\underline{x}_i, \overline{x}_i]\kappa_i, \tag{18}$$

where the notations are the same as in the Kolev affine form (16). Hansen's generalized interval can be interpreted as a real inclusion representation, called the *Hansen interval representation*, by defining

$$T = \left(\underline{c}_x, \overline{c}_x, \underline{x}_1, \overline{x}_1, \dots, \underline{x}_n, \overline{x}_n, v_1, \dots, v_n\right), \tag{19a}$$

$$V_T = (\kappa_1, \dots, \kappa_n, c_x, x_1, \dots, x_n), \tag{19b}$$

$$D_T = [-v_1, v_1] \times \cdots \times [-v_n, v_n] \times [-v_x, v_x] \times [\underline{x}_1, \overline{x}_1] \times \dots [\underline{x}_n, \overline{x}_n], \tag{19c}$$

$$f_T(V_T) = c_x + \sum_{i=1}^{n} x_i \kappa_i, \tag{19d}$$

$$\mu(T) \equiv \left\{ c_x + \sum_{i=1}^{n} x_i \kappa_i \;\middle|\; (\kappa_1, \dots, \kappa_n, c_x, x_1, \dots, x_n) \in D_T \right\}. \tag{19e}$$

Note that the function f_T in (19) is a quadratic function on $(2n+1)$ variables in V_T.

Example 8 Linear relaxations and (convex) polyhedral enclosures can also be viewed as real inclusion representations. Indeed, they are given as the intersection of m half-spaces

$$H_i \equiv \left\{ (x_1, \dots, x_n) \mid a_{i0} + \sum_{j=1}^{n} a_{ij} x_j \leq 0 \right\} \quad (\text{for } i = 1, \dots, m),$$

and are often restricted to a domain $\mathbf{B}$ that is usually a box. We can therefore obtain a real inclusion representation (9), called the *linear relaxation representation*, by defining that

$$T = (a_{10}, \dots, a_{1n}, \dots, a_{m0}, \dots, a_{mn}), \tag{20a}$$

$$V_T = (x_k), \quad \text{for some } k \in \{1, \dots, n\} \tag{20b}$$

$$D_T = \mathbf{B} \cap \bigcap_{i=1}^{m} H_i, \tag{20c}$$

$$f_T(V_T) = x_k, \tag{20d}$$

$$\mu(T) \equiv \{x_k \mid x_k \in D_T[x_k]\} = \mathbf{B} \cap \bigcap_{i=1}^{m} H_i. \tag{20e}$$

This is the intersection of $\mathbf{B}$ and the convex polyhedron $\bigcap_{i=1}^{m} H_i$.

By a similar argument, one can see that the concept of *inclusion representation* covers almost all existing inclusions for real numbers. It does not introduce new inclusion techniques, but provides an abstract view of different existing inclusion techniques.

The following theorems characterize the properties of inclusion representations.

Theorem 3 *Let $\mathcal{A}$ be a nonempty set and $\mathcal{A}'$ a subset of $\mathcal{A}$. Suppose $\mathcal{I} \equiv (\mathcal{R}, \mu)$ is an inclusion representation of $\mathcal{A}$. Then the pair $\mathcal{I}' \equiv (\mathcal{R}', \mu')$ is an inclusion representation of $\mathcal{A}'$, where*

$$\mathcal{R}' := \{T \in \mathcal{R} \mid \mu(T) \neq \emptyset \Rightarrow \mu(T) \cap \mathcal{A}' \neq \emptyset\}, \tag{21a}$$

$$\mu'(T) := \mu(T) \cap \mathcal{A}' \qquad \text{for all } T \in \mathcal{R}'. \tag{21b}$$

Proof By Definition 8, there exists a representing function ζ of $\mathcal{I}$ that maps from $2^{\mathcal{A}}$ to $\mathcal{R}$. We define a function $\zeta' : 2^{\mathcal{A}'} \to \mathcal{R}$ simply by the rule $\zeta'(S) := \zeta(S)$ for all $S \in 2^{\mathcal{A}'}$. It follows from (21) that $\zeta'(S) \in \mathcal{R}'$ because $\mu(\zeta'(S)) = \mu(\zeta(S)) \supseteq S$ and $\mu(\zeta'(\emptyset)) = \mu(\zeta(\emptyset)) = \emptyset$. Hence, ζ' is a function from $2^{\mathcal{A}'}$ to $\mathcal{R}'$ as required. Moreover, for all $S \in 2^{\mathcal{A}'}$, we have $\mu'(\zeta'(S)) = \mu(\zeta(S)) \cap \mathcal{A}' \supseteq S$. □

Theorem 4 *Let $\mathcal{I} \equiv (\mathcal{R}, \mu)$ and $\mathcal{I}' \equiv (\mathcal{R}', \mu')$ be inclusion representations of two sets $\mathcal{A}$ and $\mathcal{A}'$, respectively. Then $\mathcal{I}'' \equiv (\mathcal{R} \times \mathcal{R}', (\mu, \mu')^{\mathrm{T}})$ is an inclusion representation of $\mathcal{A} \times \mathcal{A}'$. We denote $\mathcal{I}''$ by $\mathcal{I} \times \mathcal{I}'$.*

Proof By Definition 8, there exist two representing functions ζ and ζ' of $\mathcal{I}$ and $\mathcal{I}'$, respectively. The function $\zeta'' = (\zeta, \zeta')^{\mathrm{T}}$ is a representing function of $\mathcal{I}''$. □

Notation 5 $\mathbb{I}$, $\hat{\mathbb{A}}$, *and* $\mathbb{A}$ *will refer respectively the real inclusion representations using:*

- *Interval arithmetic, as defined in* (11)*;*
- *Affine arithmetic, as defined in* (13)*;*
- *Revised affine arithmetic, as defined in* (15).

As a result, we will use $0.1 + 2.1\epsilon_2 + 9.1\epsilon_9$ to refer to the representation object $(0.1, 0.0, 2.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 9.1)$ in the standard affine representation $(\hat{\mathbb{A}}, \mu_{\hat{\mathbb{A}}})$ defined in (13). Similarly, we will use the form $[1, 3]$ to refer to the representation object $(1, 3)$ in the interval representation $(\mathbb{I}, \mu_{\mathbb{I}})$ defined in (11).

3.2 Inclusion function

We now generalize the notion of *inclusion function* as defined in interval arithmetic to make it compatible with the notion of inclusion representation.

Definition 10 (Inclusion Function) Given two sets X, Y and a function $f : X \to Y$. Let $\mathcal{I}_X = (\mathcal{R}_X, \mu_X)$ and $\mathcal{I}_Y = (\mathcal{R}_Y, \mu_Y)$ be two inclusion representations of X and Y, respectively. A function $F : \mathcal{R}_X \to \mathcal{R}_Y$ is called an *inclusion function* for f if $\forall S \subseteq X$ and $\forall T \in \mathcal{R}_X$ we have

$$S \subseteq \mu_X(T) \Rightarrow f(S) \equiv \{f(x) \mid x \in S\} \subseteq \mu_Y(F(T)). \tag{22}$$

The inclusion function defined in [16, p. 27] for intervals can be seen as special cases of Definition 10. All interval forms of a function (or multifunction) f are inclusion functions of f using the interval representation (11).

In practice, one often extends real-valued functions in a natural way to evaluate the ranges of the real-valued functions as follows.

Definition 11 (Natural Extension) Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be a factorable function that is recursively composed of a finite set, E, of elementary operations defined on $\mathbb{R}$ in a given way. Suppose that $\mathcal{I} = (\mathcal{R}, \mu)$ is an inclusion representation of $\mathbb{R}$ such that there exist a set, $E_{\mathcal{R}}$, of elementary operations defined on $\mathcal{R}$, and a bijection $\eta : E \to E_{\mathcal{R}}$.[2] A function $\mathbf{f} : \mathcal{R}^n \to \mathcal{R}^m$ is called the *natural extension* of f in $\mathcal{I}$ (using operations in $E_{\mathcal{R}}$) with respect to the given composition of f if $\mathbf{f}$ is constructed from the composition of f by replacing each real variable, constant, or operation by its counterpart. If $\mathbf{f}$ is also an inclusion function for f, we call $\mathbf{f}$ the *natural inclusion function* for f with respect to the composition.

Various interval inclusion functions are described in [16]; some among them are *natural*, *centered*, *mixed-centered* and *Newton inclusion functions*.

Example 9 In the composition of the real-valued function $f(x) = x * (x - 1)$, the real variable x occurs twice. The set of elementary operations used in the composition of f is $E = \{-, *\}$. The exact range $f([0, 1]) = [-0.25, 0]$. The natural extension of f in the interval representation $(\mathbb{I}, \mu_{\mathbb{I}})$ defined by (11) is a function $\mathbf{f} : \mathbb{I} \to \mathbb{I}$ defined as $\mathbf{f}(\mathbf{x}) = \mathbf{x} * (\mathbf{x} - \mathbf{1})$, where $\mathbf{1} = [1, 1] = 1$. Consider the representation object $T = (0, 1)$ in the interval representation $(\mathbb{I}, \mu_{\mathbb{I}})$, which corresponds to the interval $[0, 1]$. In interval arithmetic, the value of $\mathbf{f}$ at T is

$$\mathbf{f}(T) = [0, 1] * ([0, 1] - \mathbf{1}) = [0, 1] * [-1, 0] = [-1, 0].$$

[2]We then call $e \in E$ the real-valued counterpart of $\eta(e)$.

We have $f([0,1]) \subseteq \mathbf{f}([0,1])$. Moreover, we can prove that $\forall x \in \mathbf{x} \in \mathbb{I} : f(x) \in \mathbf{f}(\mathbf{x})$. Hence, $\mathbf{f}$ is the natural inclusion function of f in the interval arithmetic representation with respect to the specified composition of f.

Example 10 The natural extension of the real-valued function $f(x) = x * (x-1)$ in the standard affine representation is the function $\hat{f}(\hat{x}) = \hat{x} * (\hat{x} - \hat{1})$, where $\hat{1} = 1$. A representation object $T = (0.5, 0.5)$ in the standard affine representation $\left(\hat{\mathbb{A}}, \mu_{\hat{\mathbb{A}}}\right)$ defined by (13), which corresponds to the affine form $0.5 + 0.5\epsilon_1$, has the following real evaluation at T:

$$\mu_{\hat{\mathbb{A}}}(T) = \{0.5 + 0.5\epsilon_1 \mid \epsilon_1 \in [-1,1]\} = [0,1].$$

In affine arithmetic, the value of $\hat{f}$ at $T = 0.5 + 0.5\epsilon_1$ is

$$\begin{aligned}\hat{f}(T) &= (0.5 + 0.5\epsilon_1) * ((0.5 + 0.5\epsilon_1) - 1)\\ &= (0.5 + 0.5\epsilon_1) * (-0.5 + 0.5\epsilon_1)\\ &= -0.25 + 0.25\epsilon_{\text{new}},\end{aligned}$$

where ϵ_{new} is a new noise variable taking its value in $[-1,1]$. Hence, $\hat{f}(T)$ has the real evaluation $\mu_{\hat{\mathbb{A}}}(-0.25 + 0.25\epsilon_{\text{new}}) = [-0.5, 0]$. We can prove that the natural extension $\hat{f}$ is the natural inclusion function of f in $\left(\hat{\mathbb{A}}, \mu_{\hat{\mathbb{A}}}\right)$ – the standard affine representation.

Note that in revised affine arithmetic, $\hat{f}(T) = -0.125 + 0.125\epsilon_{\text{new}}$. Hence, it has the real evaluation $\mu_{\mathbb{A}}(-0.125 + 0.125\epsilon_{\text{new}}) = [-0.25, 0]$. This evaluation provides a tighter enclosure than the one provided by affine arithmetic.

Example 11 The natural extension of the real-valued function $f(x) = x * (x-1)$ in the revised affine representation is the function $\widehat{f}(\widehat{x}) = \widehat{x} * (\widehat{x} - \widehat{1})$, where $\widehat{1} = 1$. A representation object $T = (0.5, 0.5, 0.0)$ in the revised affine representation $(\mathbb{A}, \mu_{\mathbb{A}})$ defined by (15), which corresponds to the revised affine form $0.5 + 0.5\epsilon + 0[-1,1]$, has the following real evaluation at T:

$$\mu_{\mathbb{A}}(T) = \{0.5 + 0.5\epsilon + 0[-1,1] \mid \epsilon \in [-1,1]\} = [0,1].$$

In revised affine arithmetic, The value of $\widehat{f}$ at T is

$$\begin{aligned}\widehat{f}(T) &= (0.5 + 0.5\epsilon) * ((0.5 + 0.5\epsilon) - 1)\\ &= (0.5 + 0.5\epsilon) * (-0.5 + 0.5\epsilon)\\ &= -0.125 + 0.125[-1,1].\end{aligned}$$

Hence, $\widehat{f}(T)$ has the real evaluation $\mu_{\mathbb{A}}(-0.125 + 0.125[-1,1]) = [-0.25, 0]$. We can prove that the natural extension $\widehat{f}$ is the natural inclusion function of f in the revised affine representation with respect to the given composition of f.

Next, we define the conversion from an inclusion representation to another one without loss of the inclusion property.

Theorem 6 (Composite Inclusion Function) *Let $\mathcal{I}_X = (\mathcal{R}_X, \mu_X)$, $\mathcal{I}_Y = (\mathcal{R}_Y, \mu_Y)$ and $\mathcal{I}_Z = (\mathcal{R}_Z, \mu_Z)$ be inclusion representations of three sets X, Y and Z, respectively. If $F : \mathcal{R}_X \to \mathcal{R}_Y$ and $G : \mathcal{R}_Y \to \mathcal{R}_Z$ are inclusion functions of two functions $f : X \to Y$ and $g : Y \to Z$ respectively, then the composite function $G \circ F$ is an inclusion function of the composite function $g \circ f$.*

Proof Let $T \in \mathcal{R}_X$, $U = F(T)$, $V = G(U)$ and $S \subseteq X \cap \mu_X(T)$. By Definition 10, we have $f(S) \subseteq \mu_Y(U)$, thus, $f(S) \subseteq Y \cap \mu_Y(U)$. Therefore, also by Definition 10, we have $g(f(S)) \subseteq \mu_Z(G(U)) = \mu_Z(V)$. That is, $g \circ f(S) \subseteq \mu_Z(G \circ F(T))$ holds for every $S \subseteq X \cap \mu_X(T)$. That is, $G \circ F$ is an inclusion function of $g \circ f$. □

Corollary 1 *Let $\mathcal{I} = (\mathcal{R}, \mu)$ be an inclusion representation of $\mathbb{R}$. If elementary operations defined over $\mathcal{R}$ are inclusion functions of their counterparts over $\mathbb{R}$, then all factorable functions built over $\mathcal{R}$ by using these elementary operations are also inclusion functions of their counterparts over $\mathbb{R}$.*

Proof Corollary 1 is an obvious consequence of Theorem 6. Therefore, the proof is omitted. □

In our implementation, the elementary operations of interval arithmetic and affine arithmetic are constructed to be inclusion functions of their real-valued counterparts. It follows from Corollary 1 that all the factorable operations/functions defined in interval arithmetic (or affine arithmetic) by using these elementary operations will also be inclusion functions of their real-valued counterparts.

4 Combining multiple inclusion representations

Using the concept of inclusion representation as presented in Section 3.1, we now propose a novel generic scheme to perform constraint propagation on DAG representations. This scheme allows inclusion techniques to work in cooperation in order to obtain the effect of domain reduction. The input constraint system is represented by a DAG. The computational data stored at each node, $\mathbf{N}$, of the DAG representation consist of a representation object for each real inclusion representation $\mathcal{I} = (\mathcal{R}, \mu)$ of $\mathbb{R}$ and a *constraint range of node* which is often an interval. Choosing a node for constraint propagation is based on the computational data associated to the node. Some specific combination will be described in Section 5. They are based on interval arithmetic, affine arithmetic, interval constraint propagation, and linear programming.

4.1 Node range evaluations

The following constraint system, an NCSP, will be used in examples throughout this section.

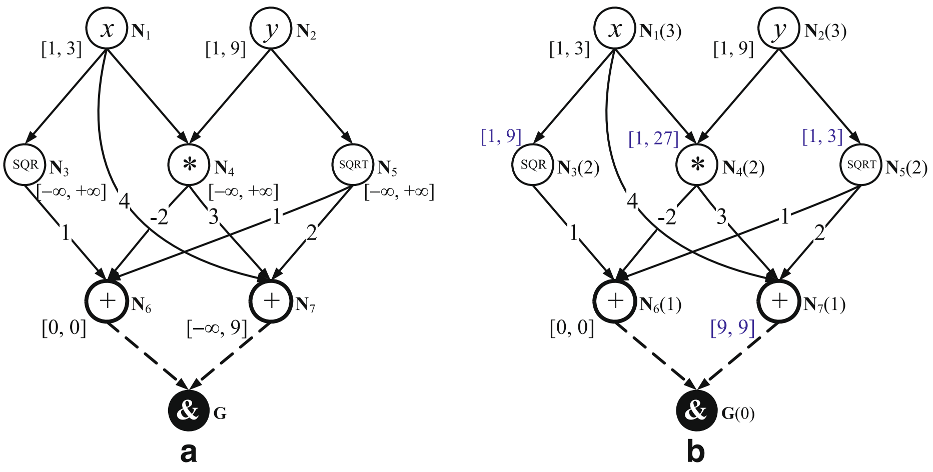


Fig. 4 The DAG representation: **a** before interval evaluations; and **b** after interval evaluations

Example 12 The DAG representation of the following NCSP is depicted in Fig. 4:

$$\left\langle x^2 - 2xy + \sqrt{y} = 0,\ 4x + 3xy + 2\sqrt{y} \le 9;\ x \in [1, 3],\ y \in [1, 9] \right\rangle.$$

The sequence $(\mathbf{N}_1, \mathbf{N}_2, \mathbf{N}_3, \mathbf{N}_4, \mathbf{N}_5, \mathbf{N}_6, \mathbf{N}_7)$ is an ordering of the nodes (see Theorem 1).

In order to combine multiple inclusion techniques during constraint propagation, we use multiple inclusion representations at each node **N** of the DAG representation. The computational data stored at **N** consists of a constraint range $\tau_{\mathbf{N}}$ and multiple representation objects.

Notation 7 *For each real inclusion representation $\mathcal{I} = (\mathcal{R}, \mu)$ under consideration, one representation object, denoted by $\mathcal{R}(\mathbf{N})$, in $\mathcal{I}$ is stored at* **N**.

Notation 8 *Let $\mathcal{I} = (\mathcal{R}, \mu)$ be a real inclusion representation, $T \in \mathcal{R}$, $T' \in \mathcal{R}$, and $\mathbf{x} \subseteq \mathbb{R}$. The notation $T \Cap \mathbf{x}$ denotes some object $T'' \in \mathcal{R}$ such that $\mu(T) \cap \mathbf{x} \subseteq \mu(T'')$ holds and $\mu(T'') \subseteq \mu(T)$ or $\mu(T'') \subseteq \mathbf{x}$ holds. The notation $T \Cap T'$ denotes some object $T'' \in \mathcal{R}$ such that $\mu(T) \cap \mu(T') \subseteq \mu(T'')$ holds and $\mu(T'') \subseteq \mu(T)$ or $\mu(T'') \subseteq \mu(T')$ holds.*

During computations, the inclusion property at each node will be maintained. That is, the exact value of the subexpression at a node **N** always lives in the node range $\tau_{\mathbf{N}}$ and in the image $\mu(\mathcal{R}(\mathbf{N}))$ of every representation object $\mathcal{R}(\mathbf{N})$ stored at **N**. Hereafter, we present a concept that allows reducing the node range of a node basing on the node ranges of its children. This concept is a generalization of the idea of *forward evaluation* [4].

Definition 12 (Node Evaluation, `NEV`) Let **N** be a node of the DAG representation of a constraint system, $\{\mathbf{C}_i\}_{i=1}^k$ the children of **N**, $f : \mathbb{R}^k \to \mathbb{R}$ the elementary operation represented by **N**, and $\mathcal{I} = (\mathcal{R}, \mu)$ a real inclusion representation. Also let

$f_{\mathcal{I}} : \mathcal{R}^k \to \mathcal{R}$ be an inclusion function for f. The following assignment is called the *node evaluation* at node $\mathbf{N}$ in the inclusion representation $\mathcal{I}$ (if $\mathbf{N} \neq \mathbf{G}$):

$$\mathtt{NEV}(\mathbf{N}, \mathcal{I}) \equiv \left\{ \begin{array}{ll} \mathcal{R}(\mathbf{N}) & := \mathcal{R}(\mathbf{N}) \cap \tau(\mathbf{N}) \cap f_{\mathcal{I}}\big(\{\mathcal{R}(\mathbf{C}_i)\}_{i=1}^{k}\big); \\ \tau(\mathbf{N}) & := \tau(\mathbf{N}) \cap \mu(\mathcal{R}(\mathbf{N})); \end{array} \right\}$$

Example 13 Consider the problem in Example 12 and the notations in Fig. 4. At the beginning, we have (see Fig. 4a):

$$\begin{array}{llll} \tau_{\mathbf{N}_1} = \mathbb{I}(\mathbf{N}_1) = [1, 3]; & \mathbb{A}(\mathbf{N}_1) = \hat{\mathbb{A}}(\mathbf{N}_1) = 2 + \epsilon_1; & \\ \tau_{\mathbf{N}_2} = \mathbb{I}(\mathbf{N}_2) = [1, 9]; & \mathbb{A}(\mathbf{N}_2) = \hat{\mathbb{A}}(\mathbf{N}_2) = 5 + 4\epsilon_2; & \\ \tau_{\mathbf{N}_i} = \mathbb{I}(\mathbf{N}_i) = [-\infty, +\infty]; & \mathbb{A}(\mathbf{N}_i) = \hat{\mathbb{A}}(\mathbf{N}_i) = [-\infty, +\infty] & (i = 3, 4, 5); \\ \tau_{\mathbf{N}_6} = \mathbb{I}(\mathbf{N}_6) = [0, 0]; & \mathbb{A}(\mathbf{N}_6) = \hat{\mathbb{A}}(\mathbf{N}_6) = 0; & \\ \tau_{\mathbf{N}_7} = \mathbb{I}(\mathbf{N}_7) = [-\infty, 9]; & \mathbb{A}(\mathbf{N}_7) = \hat{\mathbb{A}}(\mathbf{N}_7) = [-\infty, 9]; & \end{array}$$

The operation corresponding to $\mathbf{N}_3$ is the square operation; therefore, we have

$$\mathtt{NEV}(\mathbf{N}_3, \mathbb{I}) \equiv \left\{ \begin{array}{ll} \mathbb{I}(\mathbf{N}_3) & := \big((\mathbb{I}(\mathbf{N}_1))^2 \Cap \tau_{\mathbf{N}_3}\big) \Cap \mathbb{I}(\mathbf{N}_3); \\ \tau_{\mathbf{N}_3} & := \tau_{\mathbf{N}_3} \cap \mathbb{I}(\mathbf{N}_3); \end{array} \right\},$$

$$\mathtt{NEV}(\mathbf{N}_3, \hat{\mathbb{A}}) \equiv \left\{ \begin{array}{ll} \hat{\mathbb{A}}(\mathbf{N}_3) & := \Big(\big(\hat{\mathbb{A}}(\mathbf{N}_1)\big)^2 \Cap \tau_{\mathbf{N}_3}\Big) \Cap \hat{\mathbb{A}}(\mathbf{N}_3); \\ \tau_{\mathbf{N}_3} & := \tau_{\mathbf{N}_3} \cap \mu_{\hat{\mathbb{A}}}(\hat{\mathbb{A}}(\mathbf{N}_3)); \end{array} \right\},$$

$$\mathtt{NEV}(\mathbf{N}_3, \mathbb{A}) \equiv \left\{ \begin{array}{ll} \mathbb{A}(\mathbf{N}_3) & := \big((\mathbb{A}(\mathbf{N}_1))^2 \Cap \tau_{\mathbf{N}_3}\big) \Cap \mathbb{A}(\mathbf{N}_3); \\ \tau_{\mathbf{N}_3} & := \tau_{\mathbf{N}_3} \cap \mu_{\mathbb{A}}(\mathbb{A}(\mathbf{N}_3)); \end{array} \right\}.$$

After the node evaluation $\mathtt{NEV}(\mathbf{N}_3, \mathbb{I})$, we have

$$\begin{array}{lll} \mathbb{I}(\mathbf{N}_3) = \big(([1, 3])^2 \Cap [-\infty, +\infty]\big) \Cap [-\infty, +\infty] & = [1, 9], \\ \tau_{\mathbf{N}_3} \quad = [-\infty, +\infty] \cap [1, 9] & = [1, 9]. \end{array}$$

After performing $\mathtt{NEV}(\mathbf{N}_3, \hat{\mathbb{A}})$ and $\mathtt{NEV}(\mathbf{N}_3, \mathbb{A})$ by using affine arithmetic and revised affine arithmetic, we have

$$\begin{array}{ll} \hat{\mathbb{A}}(\mathbf{N}_3) = \big((2 + \epsilon_1)^2 \Cap [1, 9]\big) \Cap [-\infty, +\infty] & = 4.5 + 4\epsilon_1 + 0.5\epsilon_3, \\ \tau_{\mathbf{N}_3} \quad = [1, 9] \cap \mu_{\hat{\mathbb{A}}}(4.5 + 4\epsilon_1 + 0.5\epsilon_3) & = [1, 9], \\ \mathbb{A}(\mathbf{N}_3) = \big((2 + \epsilon_1)^2 \Cap [1, 9]\big) \Cap [-\infty, +\infty] & = 4.5 + 4\epsilon_1 + 0.5[-1, 1], \\ \tau_{\mathbf{N}_3} \quad = [1, 9] \cap \mu_{\mathbb{A}}(4.5 + 4\epsilon_1 + 0.5[-1, 1]) & = [1, 9]. \end{array}$$

Similarly, after performing node evaluations at the other nodes we have

$$\begin{array}{ll} \mathbb{I}(\mathbf{N}_4) = \tau_{\mathbf{N}_4} = [1, 27], & \hat{\mathbb{A}}(\mathbf{N}_4) = 10 + 5\epsilon_1 + 8\epsilon_2 + 4\epsilon_4, \\ & \mathbb{A}(\mathbf{N}_4) = 10 + 5\epsilon_1 + 8\epsilon_2 + 4[-1, 1], \\ \mathbb{I}(\mathbf{N}_5) = \tau_{\mathbf{N}_5} = [1, 3], & \hat{\mathbb{A}}(\mathbf{N}_5) = 2.125 + \epsilon_2 + 0.125\epsilon_5, \\ & \mathbb{A}(\mathbf{N}_5) = 2.125 + \epsilon_2 + 0.125[-1, 1], \\ \mathbb{I}(\mathbf{N}_6) = \tau_{\mathbf{N}_6} = [0, 0], & \hat{\mathbb{A}}(\mathbf{N}_6) = -13.375 - 6\epsilon_1 - 15\epsilon_2 + 0.5\epsilon_3 - 8\epsilon_4 + 0.125\epsilon_5, \\ & \mathbb{A}(\mathbf{N}_6) = -13.375 - 6\epsilon_1 - 15\epsilon_2 + 8.625[-1, 1], \\ \mathbb{I}(\mathbf{N}_7) = \tau_{\mathbf{N}_7} = [9, 9], & \hat{\mathbb{A}}(\mathbf{N}_7) = 42.25 + 19\epsilon_1 + 26\epsilon_2 + 12\epsilon_4 + 0.25\epsilon_5, \\ & \mathbb{A}(\mathbf{N}_7) = 42.25 + 19\epsilon_1 + 26\epsilon_2 + 12.25[-1, 1]. \end{array}$$

In practice, the node range $\tau_{\mathbf{N}}$ and the interval object $\mathbb{I}(\mathbf{N})$ should be merged into one object because they are of the same type.

4.2 Induced constraint systems for domain reduction

In order to present the concept of a *pruning constraint system* concisely, we rely on the following concept.

Definition 13 (Inclusion Constraint System, ICS) Let $(\{R\}, \mu)$ be a real inclusion representation of $\mathbb{R}$ defined by (7), $\mathbf{N}$ a node of the DAG representation. The *inclusion constraint system* induced by a object $T \equiv \{R\}(\mathbf{N})$ and a *constraint range* $D \subseteq \mathbb{R}$ is defined as

$$\mathtt{ICS}(T, D) \equiv \begin{cases} \left\{\vartheta_{\mathbf{N}} \in D_T \cap D\right\} \text{ (i.e., } V_T \equiv \{\vartheta_{\mathbf{N}}\}\text{) if } f_T \text{ is the identity,} \\ \left\{ f_T(V_T) = \vartheta_{\mathbf{N}};\ V_T \in D_T;\ \vartheta_{\mathbf{N}} \in D\right\} \text{ otherwise;} \end{cases}$$

where the set of variables consists of the variable $\vartheta_{\mathbf{N}}$, the variables in V_T, and the variables used to describe D_T.

Roughly speaking, the inclusion constraint system induced by a representation object T at a node $\mathbf{N}$ is a set of redundant constraints that can be inferred from T (the inclusion $\vartheta_{\mathbf{N}} \in \mu$ always hold). Some example of inclusion constraint systems are given below.

Example 14 Inclusion constraint systems for different inclusion representations:

– An inclusion constraint system for the interval form (11):

$$\mathtt{ICS}(T, [c, d]) \equiv \{\vartheta_{\mathbf{N}} \in [c, d] \cap [a, b]\},$$

where the set of variables is $\{\vartheta_{\mathbf{N}}\}$. This system is conjunctive and has the form of bound constraint.

– An inclusion constraint system for the interval union form (12):

$$\mathtt{ICS}(T, [c, d]) \equiv \left\{\vartheta_{\mathbf{N}} \in [c, d];\ \vartheta_{\mathbf{N}} \in \bigcup_{i=1}^{m} [a_i, b_i]\right\},$$

where the set of variables is $\{\vartheta_{\mathbf{N}}\}$. This system has the form of the disjunction of bound constraints.

– An inclusion constraint system for the affine form (13):

$$\mathtt{ICS}(T, [c, d]) \equiv \left\{x_0 + \sum_{i=1}^{n} x_i\epsilon_i = \vartheta_{\mathbf{N}};\ (\epsilon_1, \ldots, \epsilon_n) \in [-1, 1]^n;\ \vartheta_{\mathbf{N}} \in [c, d]\right\},$$

where the set of variables is $\{\epsilon_1, \ldots, \epsilon_n, \vartheta_{\mathbf{N}}\}$. This system is conjunctive and linear.

– An inclusion constraint system for the revised affine form (15):

$$\texttt{ICS}(T,[c,d]) \equiv \left\{ x_0 + \sum_{i=1}^{n} x_i\epsilon_i + e_x\epsilon_x = \vartheta_{\mathbf{N}};\ (\epsilon_1,\dots,\epsilon_n,\epsilon_x) \in [-1,1]^{n+1};\ \vartheta_{\mathbf{N}} \in [c,d] \right\},$$

where the set of variables is $\{\epsilon_1,\dots,\epsilon_n,\epsilon_x,\vartheta_{\mathbf{N}}\}$. This system is conjunctive and linear.

– An inclusion constraint system for the Kolev affine form (17):

$$\texttt{ICS}(T,[c,d]) \equiv \left\{ c_x + \sum_{i=1}^{n} x_i\kappa_i + \kappa_x = \vartheta_{\mathbf{N}};\ \kappa_i \in [-v_i,v_i];\ \kappa_x \in [-v_x,v_x];\ \vartheta_{\mathbf{N}} \in [c,d] \right\},$$

where the set of variables is $\{\kappa_1,\dots,\kappa_n,\kappa_x,\vartheta_{\mathbf{N}}\}$. This system is conjunctive and linear.

– An inclusion constraint system for the Hansen interval form (19):

$$\texttt{ICS}(T,[c,d]) \equiv \left\{ c_x + \sum_{i=1}^{n} x_i\kappa_i = \vartheta_{\mathbf{N}}; \kappa_i \in [-v_i,v_i]; c_x \in [\underline{c}_x,\overline{c}_x]; x_i \in [\underline{x}_i,\overline{x}_i]; \vartheta_{\mathbf{N}} \in [c,d] \right\},$$

where the set of variables is $\{\kappa_1,\dots,\kappa_n,c_x,x_1,\dots,x_n,\vartheta_{\mathbf{N}}\}$. This system is conjunctive and quadratic.

– An inclusion constraint system for the linear relaxations/polyhedral enclosures (20):

$$\texttt{ICS}(T,[c,d]) \equiv \left\{ a_{i0} + \sum_{j=1}^{n} a_{ij}x_j \le 0\ (i=1,\dots,m);\ \vartheta_{\mathbf{N}} \equiv x_k \in [c,d];\ (x_1,\dots,x_n) \in \mathbf{B} \right\},$$

where the set of variables is $\{x_1,\dots,x_n\}$. This system is conjunctive and linear. Note that $\vartheta_{\mathbf{N}}$ is one of the variables $x_1,\dots,x_n$.

We now define the constraint systems constructed for pruning the node ranges, using the representation objects stored at related nodes.

Definition 14 (Pruning Constraint System, PCS) Let $\mathbf{N}$ be a node of the DAG representation of a constraint system, $(\mathbf{C}_i)_{i=1}^{k}$ the children of $\mathbf{N}$, $h:\mathbb{R}^k \to \mathbb{R}$ the operation represented by $\mathbf{N}$, and $\mathcal{S}$ a finite set of real inclusion representations. The following constraint system is called the *pruning constraint system* induced by $\mathcal{S}$ at $\mathbf{N}$:

$$\texttt{PCS}(\mathbf{N},\mathcal{S}) \equiv \begin{cases} \left\{ \bigwedge_{i=1}^{k} \texttt{ICS}(\mathcal{R}(\mathbf{C}_i),\tau_{\mathbf{C}_i}); \right\} & \text{if } \mathbf{N} \text{ is ground,} \\ \left\{ \begin{array}{l} h(\vartheta_{\mathbf{C}_1},\dots,\vartheta_{\mathbf{C}_k}) = \vartheta_{\mathbf{N}}; \\ \bigwedge_{(\mathcal{R},\mu)\in\mathcal{S}} \texttt{PCSub}(\mathbf{N},\mathcal{R},\mu); \end{array} \right\} & \text{otherwise;} \end{cases}$$

where $\texttt{PCSub}(\mathbf{N}, \mathcal{R}, \mu)$ is a *pruning constraint subsystem* defined as

$$\texttt{PCSub}(\mathbf{N}, \mathcal{R}, \mu) \equiv \left(\texttt{ICS}(\mathcal{R}(\mathbf{N}), \tau_{\mathbf{N}}) \wedge \bigwedge_{i=1}^{k} \texttt{ICS}(\mathcal{R}(\mathbf{C}_i), \tau_{\mathbf{C}_i})\right).$$

Roughly speaking, the pruning constraint system induced by $\mathcal{S}$ at $\mathbf{N}$ includes all inclusion constraint systems induced by all representation objects stored at $\mathbf{N}$ and the children of $\mathbf{N}$. The operation h is a constraint itself in the pruning constraint system. The concept of a pruning constraint system is depicted in Fig. 5.

Notation 9 *In the rest of the paper, we will abuse the notations* $\mathbb{I}$ *and* $\mathbb{A}$ *to denote the real inclusion representations,* $(\mathbb{I}, \mu_{\mathbb{I}})$ *and* $(\mathbb{A}, \mu_{\mathbb{A}})$*, respectively defined on interval arithmetic and revised affine arithmetic; where the function* $\mu_{\mathbb{I}}$ *is defined by* (11) *and the function* $\mu_{\mathbb{A}}$ *is defined by* (15).

Example 15 Let us consider the problem in Example 12 and continue the computations in Example 13. We have, for instance, the following inclusion constraint systems:

$$\begin{aligned}
\texttt{ICS}(\mathbb{I}(\mathbf{N}_1), \tau_{\mathbf{N}_1}) &\equiv \{\vartheta_{\mathbf{N}_1} \in [1, 3]\};\\
\texttt{ICS}(\mathbb{I}(\mathbf{N}_2), \tau_{\mathbf{N}_2}) &\equiv \{\vartheta_{\mathbf{N}_2} \in [1, 9]\};\\
\texttt{ICS}(\mathbb{I}(\mathbf{N}_3), \tau_{\mathbf{N}_3}) &\equiv \{\vartheta_{\mathbf{N}_3} \in [1, 9]\};\\
\texttt{ICS}(\mathbb{I}(\mathbf{N}_4), \tau_{\mathbf{N}_4}) &\equiv \{\vartheta_{\mathbf{N}_4} \in [1, 27]\};\\
\texttt{ICS}(\mathbb{I}(\mathbf{N}_5), \tau_{\mathbf{N}_5}) &\equiv \{\vartheta_{\mathbf{N}_5} \in [1, 3]\};\\
\texttt{ICS}(\mathbb{I}(\mathbf{N}_6), \tau_{\mathbf{N}_6}) &\equiv \{\vartheta_{\mathbf{N}_6} \in [0, 0]\};\\
\texttt{ICS}(\mathbb{I}(\mathbf{N}_7), \tau_{\mathbf{N}_7}) &\equiv \{\vartheta_{\mathbf{N}_7} \in [9, 9]\};\\
\texttt{ICS}\big(\hat{\mathbb{A}}(\mathbf{N}_1), \tau_{\mathbf{N}_1}\big) &\equiv \left\{\begin{array}{l} 2 + \epsilon_1 = \vartheta_{\mathbf{N}_1};\\ \vartheta_{\mathbf{N}_1} \in [1, 3];\ \epsilon_1 \in [-1, 1]; \end{array}\right\};\\
\texttt{ICS}\big(\hat{\mathbb{A}}(\mathbf{N}_2), \tau_{\mathbf{N}_2}\big) &\equiv \left\{\begin{array}{l} 5 + 4\epsilon_2 = \vartheta_{\mathbf{N}_2};\\ \vartheta_{\mathbf{N}_2} \in [1, 9];\ \epsilon_2 \in [-1, 1]; \end{array}\right\};\\
\texttt{ICS}\big(\hat{\mathbb{A}}(\mathbf{N}_3), \tau_{\mathbf{N}_3}\big) &\equiv \left\{\begin{array}{l} 4.5 + 4\epsilon_1 + 0.5\epsilon_3 = \vartheta_{\mathbf{N}_3};\\ \vartheta_{\mathbf{N}_3} \in [1, 9];\ \epsilon_i \in [-1, 1]\ \ (i = 1, 3); \end{array}\right\};
\end{aligned}$$

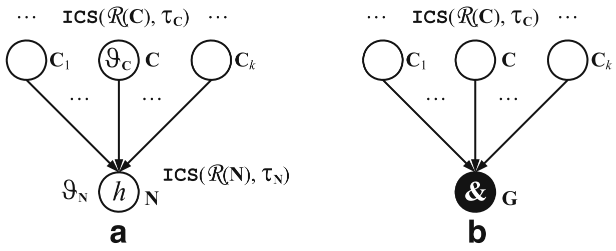


Fig. 5 A pruning constraint system at $\mathbf{N}$: **a** in case $\mathbf{N}$ is not the ground, all $\texttt{ICS}$ systems and the relation h on the nodes $\mathbf{N}, \mathbf{C}_1, \ldots, \mathbf{C}_k$ are included; and **b** in case $\mathbf{N}$ is the ground, only $\texttt{ICS}$ systems on the child nodes $\mathbf{C}_1, \ldots, \mathbf{C}_k$ are included

$$\texttt{ICS}\big(\hat{\mathbb{A}}(\mathbf{N}_4), \tau_{\mathbf{N}_4}\big) \equiv \left\{\begin{array}{l} 10 + 5\epsilon_1 + 8\epsilon_2 + 4\epsilon_4 = \vartheta_{\mathbf{N}_4}; \\ \vartheta_{\mathbf{N}_4} \in [1, 27];\ \epsilon_i \in [-1, 1]\ \ (i = 1, 2, 4); \end{array}\right\};$$

$$\texttt{ICS}\big(\hat{\mathbb{A}}(\mathbf{N}_5), \tau_{\mathbf{N}_5}\big) \equiv \left\{\begin{array}{l} 2.125 + \epsilon_2 + 0.125\epsilon_5 = \vartheta_{\mathbf{N}_5}; \\ \vartheta_{\mathbf{N}_5} \in [1, 3];\ \epsilon_i \in [-1, 1]\ \ (i = 2, 5); \end{array}\right\};$$

$$\texttt{ICS}\big(\hat{\mathbb{A}}(\mathbf{N}_6), \tau_{\mathbf{N}_6}\big) \equiv \left\{\begin{array}{l} -13.375 - 6\epsilon_1 - 15\epsilon_2 + 0.5\epsilon_3 - 8\epsilon_4 + 0.125\epsilon_5 = \vartheta_{\mathbf{N}_6}; \\ \vartheta_{\mathbf{N}_6} \in [0, 0];\ \epsilon_i \in [-1, 1]\ \ (i = 1, \ldots, 5); \end{array}\right\};$$

$$\texttt{ICS}\big(\hat{\mathbb{A}}(\mathbf{N}_7), \tau_{\mathbf{N}_7}\big) \equiv \left\{\begin{array}{l} 42.25 + 19\epsilon_1 + 26\epsilon_2 + 12\epsilon_4 + 0.25\epsilon_5 = \vartheta_{\mathbf{N}_7}; \\ \vartheta_{\mathbf{N}_7} \in [9, 9];\ \epsilon_i \in [-1, 1]\ (i = 1, 2, 4, 5); \end{array}\right\};$$

$$\texttt{ICS}(\mathbb{A}(\mathbf{N}_1), \tau_{\mathbf{N}_1}) \equiv \left\{\begin{array}{l} 2 + \epsilon_1 = \vartheta_{\mathbf{N}_1}; \\ \vartheta_{\mathbf{N}_1} \in [1, 3];\ \epsilon_1 \in [-1, 1]; \end{array}\right\};$$

$$\texttt{ICS}(\mathbb{A}(\mathbf{N}_2), \tau_{\mathbf{N}_2}) \equiv \left\{\begin{array}{l} 5 + 4\epsilon_2 = \vartheta_{\mathbf{N}_2}; \\ \vartheta_{\mathbf{N}_2} \in [1, 9];\ \epsilon_2 \in [-1, 1]; \end{array}\right\};$$

$$\texttt{ICS}(\mathbb{A}(\mathbf{N}_3), \tau_{\mathbf{N}_3}) \equiv \left\{\begin{array}{l} 4.5 + 4\epsilon_1 + 0.5\epsilon_{\mathbf{N}_3} = \vartheta_{\mathbf{N}_3}; \\ \vartheta_{\mathbf{N}_3} \in [1, 9];\ (\epsilon_1, \epsilon_{\mathbf{N}_3}) \in [-1, 1]^2; \end{array}\right\};$$

$$\texttt{ICS}(\mathbb{A}(\mathbf{N}_4), \tau_{\mathbf{N}_4}) \equiv \left\{\begin{array}{l} 10 + 5\epsilon_1 + 8\epsilon_2 + 4\epsilon_{\mathbf{N}_4} = \vartheta_{\mathbf{N}_4}; \\ \vartheta_{\mathbf{N}_4} \in [1, 27];\ (\epsilon_1, \epsilon_2, \epsilon_{\mathbf{N}_4}) \in [-1, 1]^3; \end{array}\right\};$$

$$\texttt{ICS}(\mathbb{A}(\mathbf{N}_5), \tau_{\mathbf{N}_5}) \equiv \left\{\begin{array}{l} 2.125 + \epsilon_2 + 0.125\epsilon_{\mathbf{N}_5} = \vartheta_{\mathbf{N}_5}; \\ \vartheta_{\mathbf{N}_5} \in [1, 3];\ (\epsilon_2, \epsilon_{\mathbf{N}_5}) \in [-1, 1]^2; \end{array}\right\};$$

$$\texttt{ICS}(\mathbb{A}(\mathbf{N}_6), \tau_{\mathbf{N}_6}) \equiv \left\{\begin{array}{l} -13.375 - 6\epsilon_1 - 15\epsilon_2 + 8.625\epsilon_{\mathbf{N}_6} = \vartheta_{\mathbf{N}_6}; \\ \vartheta_{\mathbf{N}_6} \in [0, 0];\ (\epsilon_1, \epsilon_2, \epsilon_{\mathbf{N}_6}) \in [-1, 1]^3; \end{array}\right\};$$

$$\texttt{ICS}(\mathbb{A}(\mathbf{N}_7), \tau_{\mathbf{N}_7}) \equiv \left\{\begin{array}{l} 42.25 + 19\epsilon_1 + 26\epsilon_2 + 12.25\epsilon_{\mathbf{N}_7} = \vartheta_{\mathbf{N}_7}; \\ \vartheta_{\mathbf{N}_7} \in [9, 9];\ (\epsilon_1, \epsilon_2, \epsilon_{\mathbf{N}_7}) \in [-1, 1]^3; \end{array}\right\}.$$

Therefore, we can infer the following pruning constraint systems as examples:

$$\texttt{PCS}\big(\mathbf{G}, \{\hat{\mathbb{A}}\}\big) \equiv \left\{\begin{array}{l} -13.375 - 6\epsilon_1 - 15\epsilon_2 + 0.5\epsilon_3 - 8\epsilon_4 + 0.125\epsilon_5 = \vartheta_{\mathbf{N}_6}; \\ 42.25 + 19\epsilon_1 + 26\epsilon_2 + 12\epsilon_4 + 0.25\epsilon_5 = \vartheta_{\mathbf{N}_7}; \\ \vartheta_{\mathbf{N}_6} \in [0, 0];\ \vartheta_{\mathbf{N}_7} \in [9, 9];\ \epsilon_i \in [-1, 1]\ \ (i = 1, \ldots, 5); \end{array}\right.$$

$$\texttt{PCS}(\mathbf{G}, \{\mathbb{A}\}) \equiv \left\{\begin{array}{l} -13.375 - 6\epsilon_1 - 15\epsilon_2 + 8.625\epsilon_{\mathbf{N}_6} = \vartheta_{\mathbf{N}_6}; \\ 42.25 + 19\epsilon_1 + 26\epsilon_2 + 12.25\epsilon_{\mathbf{N}_7} = \vartheta_{\mathbf{N}_7}; \\ \vartheta_{\mathbf{N}_6} \in [0, 0];\ \vartheta_{\mathbf{N}_7} \in [9, 9];\ (\epsilon_1, \epsilon_2, \epsilon_{\mathbf{N}_6}, \epsilon_{\mathbf{N}_7}) \in [-1, 1]^4; \end{array}\right.$$

$$\texttt{PCS}(\mathbf{N}_6, \{\mathbb{I}\}) \equiv \left\{\begin{array}{l} \vartheta_{\mathbf{N}_3} - 2\vartheta_{\mathbf{N}_4} + \vartheta_{\mathbf{N}_5} = \vartheta_{\mathbf{N}_6}; \\ \vartheta_{\mathbf{N}_3} \in [1, 9];\ \vartheta_{\mathbf{N}_4} \in [1, 27];\ \vartheta_{\mathbf{N}_5} \in [1, 3];\ \vartheta_{\mathbf{N}_6} \in [0, 0]; \end{array}\right.$$

$$\mathtt{PCS}\big(\mathbf{N}_6, \{\hat{\mathbb{A}}\}\big) \equiv \begin{cases} \vartheta_{\mathbf{N}_3} - 2\vartheta_{\mathbf{N}_4} + \vartheta_{\mathbf{N}_5} = \vartheta_{\mathbf{N}_6}; \\ 4.5 + 4\epsilon_1 + 0.5\epsilon_3 = \vartheta_{\mathbf{N}_3}; \\ 10 + 5\epsilon_1 + 8\epsilon_2 + 4\epsilon_4 = \vartheta_{\mathbf{N}_4}; \\ 2.125 + \epsilon_2 + 0.125\epsilon_5 = \vartheta_{\mathbf{N}_5}; \\ -13.375 - 6\epsilon_1 - 15\epsilon_2 + 0.5\epsilon_3 - 8\epsilon_4 + 0.125\epsilon_5 = \vartheta_{\mathbf{N}_6}; \\ \vartheta_{\mathbf{N}_3} \in [1,9];\ \vartheta_{\mathbf{N}_4} \in [1,27];\ \vartheta_{\mathbf{N}_5} \in [1,3];\ \vartheta_{\mathbf{N}_6} \in [0,0]; \\ (\epsilon_1, \ldots, \epsilon_5) \in [-1,1]^5; \end{cases}$$

$$\mathtt{PCS}(\mathbf{N}_6, \{\mathbb{A}\}) \equiv \begin{cases} \vartheta_{\mathbf{N}_3} - 2\vartheta_{\mathbf{N}_4} + \vartheta_{\mathbf{N}_5} = \vartheta_{\mathbf{N}_6}; \\ 4.5 + 4\epsilon_1 + 0.5\epsilon_{\mathbf{N}_3} = \vartheta_{\mathbf{N}_3}; \\ 10 + 5\epsilon_1 + 8\epsilon_2 + 4\epsilon_{\mathbf{N}_4} = \vartheta_{\mathbf{N}_4}; \\ 2.125 + \epsilon_2 + 0.125\epsilon_{\mathbf{N}_5} = \vartheta_{\mathbf{N}_5}; \\ -13.375 - 6\epsilon_1 - 15\epsilon_2 + 8.625\epsilon_{\mathbf{N}_6} = \vartheta_{\mathbf{N}_6}; \\ \vartheta_{\mathbf{N}_3} \in [1,9];\ \vartheta_{\mathbf{N}_4} \in [1,27]; \vartheta_{\mathbf{N}_5} \in [1,3];\ \vartheta_{\mathbf{N}_6} \in [0,0]; \\ (\epsilon_1, \epsilon_2, \epsilon_{\mathbf{N}_3}, \epsilon_{\mathbf{N}_4}, \epsilon_{\mathbf{N}_5}, \epsilon_{\mathbf{N}_6}) \in [-1,1]^6; \end{cases}$$

$$\mathtt{PCS}(\mathbf{N}_7, \{\mathbb{I}\}) \equiv \begin{cases} 4\vartheta_{\mathbf{N}_1} + 3\vartheta_{\mathbf{N}_4} + 2\vartheta_{\mathbf{N}_5} = \vartheta_{\mathbf{N}_7}; \\ \vartheta_{\mathbf{N}_1} \in [1,3];\ \vartheta_{\mathbf{N}_4} \in [1,27];\ \vartheta_{\mathbf{N}_5} \in [1,3];\ \vartheta_{\mathbf{N}_7} \in [9,9]; \end{cases}$$

$$\mathtt{PCS}\big(\mathbf{N}_7, \{\hat{\mathbb{A}}\}\big) \equiv \begin{cases} 4\vartheta_{\mathbf{N}_1} + 3\vartheta_{\mathbf{N}_4} + 2\vartheta_{\mathbf{N}_5} = \vartheta_{\mathbf{N}_7}; \\ 2 + \epsilon_1 = \vartheta_{\mathbf{N}_1}; \\ 10 + 5\epsilon_1 + 8\epsilon_2 + 4\epsilon_4 = \vartheta_{\mathbf{N}_4}; \\ 2.125 + \epsilon_2 + 0.125\epsilon_5 = \vartheta_{\mathbf{N}_5}; \\ 42.25 + 19\epsilon_1 + 26\epsilon_2 + 12\epsilon_4 + 0.25\epsilon_5 = \vartheta_{\mathbf{N}_7}; \\ \vartheta_{\mathbf{N}_1} \in [1,3];\ \vartheta_{\mathbf{N}_4} \in [1,27];\ \vartheta_{\mathbf{N}_5} \in [1,3];\ \vartheta_{\mathbf{N}_7} \in [9,9]; \\ \epsilon_i \in [-1,1]\ \ (i = 1,2,4,5); \end{cases}$$

$$\mathtt{PCS}(\mathbf{N}_7, \{\mathbb{A}\}) \equiv \begin{cases} 4\vartheta_{\mathbf{N}_1} + 3\vartheta_{\mathbf{N}_4} + 2\vartheta_{\mathbf{N}_5} = \vartheta_{\mathbf{N}_7}; \\ 2 + \epsilon_1 = \vartheta_{\mathbf{N}_1}; \\ 10 + 5\epsilon_1 + 8\epsilon_2 + 4\epsilon_{\mathbf{N}_4} = \vartheta_{\mathbf{N}_4}; \\ 2.125 + \epsilon_2 + 0.125\epsilon_{\mathbf{N}_5} = \vartheta_{\mathbf{N}_5}; \\ 42.25 + 19\epsilon_1 + 26\epsilon_2 + 12.25\epsilon_{\mathbf{N}_7} = \vartheta_{\mathbf{N}_7}; \\ \vartheta_{\mathbf{N}_1} \in [1,3];\ \vartheta_{\mathbf{N}_4} \in [1,27];\ \vartheta_{\mathbf{N}_5} \in [1,3];\ \vartheta_{\mathbf{N}_7} \in [9,9]; \\ (\epsilon_1, \epsilon_2, \epsilon_{\mathbf{N}_4}, \epsilon_{\mathbf{N}_5}, \epsilon_{\mathbf{N}_7}) \in [-1,1]^5. \end{cases}$$

We can see that either from $\mathtt{PCS}(\mathbf{N}_7, \{\mathbb{I}\})$ or from $\mathtt{PCS}(\mathbf{N}_7, \{\mathbb{A}\})$, we have $\vartheta_{\mathbf{N}_1} = 1$, $\vartheta_{\mathbf{N}_4} = 1$, and $\vartheta_{\mathbf{N}_5} = 1$. Therefore, the system has a unique solution $(x, y) = (1, 1)$.

4.3 **CIRD** – A combination scheme for propagation

We now describe our generic scheme for combining different real inclusion representations during constraint propagation. Each input constraint system, a NCSP, is represented by a DAG as described in Section 2.3. The computational data stored

Algorithm 1 CIRD—a Scheme for Combining Inclusion Representations on DAGs

1. **Initialization Phase.**
   - (a) **Initial Node Evaluation.** Traverse the DAG representation $\mathcal{G}$ in an order described in Theorem 1. When visiting a node $\mathbf{N} \in \mathcal{G}$, perform the node evaluation $\texttt{NEV}(\mathbf{N}, \mathcal{I})$ for each $\mathcal{I} \in \mathcal{E}$. It is encouraged to merge the assignments of multiple $\texttt{NEV}(\mathbf{N}, \mathcal{I})$, where $\mathcal{I} \in \mathcal{E}$, into a single process to avoid repeating the same assignments or computations.
   - (b) **Initialization of Waiting Lists.** Set $\mathcal{L}_\mathrm{e} := \emptyset$, $\mathcal{L}_\mathrm{p} :=$ {the list of all nodes representing the running constraints together with all the real inclusion representations of $\mathcal{E}$}.
2. **Propagation Phase.** Repeat this step until both $\mathcal{L}_\mathrm{e}$ and $\mathcal{L}_\mathrm{p}$ become empty or the limit, if any, on the number of iterations is reached.
   - (a) **Getting the Next Node.** Get a node $\mathbf{N}$ (and the set $\mathcal{S}$ of real inclusion representations associated with $\mathbf{N}$ in the corresponding list) from the two waiting lists $\mathcal{L}_\mathrm{e}$ and $\mathcal{L}_\mathrm{p}$, according to some *choosing strategy*.
   - (b) **Node Evaluation.** Do this step only if $\mathbf{N}$ was taken from $\mathcal{L}_\mathrm{e}$ at Step 2a. For each $\mathcal{I} = (\mathcal{R}, \mu) \in \mathcal{S}$, do the following steps:
     - i. Perform the node evaluation $\texttt{NEV}(\mathbf{N}, \mathcal{I})$. If this returns an empty set, the algorithm terminates with an infeasible status.
     - ii. If the changes of $\mathcal{R}(\mathbf{N})$ and $\tau_\mathbf{N}$ at Step 2b(i) are considered enough to re-evaluate the parents of $\mathbf{N}$, then put each node in $\mathsf{parents}(\mathbf{N})$ (associated with $\mathcal{I}$) into $\mathcal{L}_\mathrm{e}$, if $\mathbf{N}$ is not the ground node, or into $\mathcal{L}_\mathrm{p}$, otherwise.
     - iii. If the changes of $\mathcal{R}(\mathbf{N})$ and $\tau_\mathbf{N}$ at Step 2b(i) are considered enough to do a node pruning at $\mathbf{N}$ again, then put $(\mathbf{N}, \mathcal{I})$ into $\mathcal{L}_\mathrm{p}$.
   - (c) **Node Pruning.** Do this step only if $\mathbf{N}$ was taken from $\mathcal{L}_\mathrm{p}$ at Step 2a.
     - i. Choose a subset $\mathcal{T} \subseteq \mathcal{S}$ such that, for each $\mathcal{I} \in \mathcal{T}$, there are efficient domain reduction techniques for the constraint system $\texttt{PCS}(\mathbf{N}, \mathcal{I})$.
     - ii. Partition $\mathcal{T}$ into subsets such that, for each subset $\mathcal{U}$ of the partition, choose a domain reduction technique which may efficiently reduce the domains of the variables of the system (or a subsystem of) $\texttt{PCS}(\mathbf{N}, \mathcal{U})$. Afterward, apply the chosen domain reduction technique to each system (or a subsystem of) $\texttt{PCS}(\mathbf{N}, \mathcal{U})$ in a certain order. If this process returns an empty set, the algorithm terminates with an infeasible status.
     - iii. Let $\mathcal{K}$ be the set of all nodes for which the evaluating functions (in the form (7)) contain some variables whose domains were reduced at Step 2c(ii). Choose a subset $\mathcal{H}$ of $\mathcal{K}$ for node range updates, *for example*, such that each node $\mathbf{M}$ in the set $\mathcal{H}$ is a descendant of $\mathbf{N}$.
       Node Range Update: For each node $\mathbf{M}$ in $\mathcal{H}$ and each real inclusion representation $\mathcal{I} = (\mathcal{R}, \mu) \in \mathcal{E}$ such that the representation of $\mu(\mathcal{R}(\mathbf{M}))$ in the form (7) contains some variables (in V_T) whose domains have just been reduced at Step 2c(ii), update $\mathcal{R}(\mathbf{M})$ by using these newly reduced domains, then update $\tau_\mathbf{M} := \tau_\mathbf{M} \cap \mu(\mathcal{R}(\mathbf{M}))$. If an empty representation is obtained, the algorithm terminates with an infeasible status.
       - A. If the changes of $\mathcal{R}(\mathbf{M})$ and $\tau_\mathbf{M}$ are considered enough to re-evaluate the parents of $\mathbf{M}$, put each node in $\mathsf{parents}(\mathbf{M})$ associated with $\mathcal{I}$ into $\mathcal{L}_\mathrm{e}$.
       - B. If the changes of $\mathcal{R}(\mathbf{M})$ and $\tau_\mathbf{M}$ are considered enough to do a node pruning at $\mathbf{M}$, put $(\mathbf{M}, \mathcal{I})$ into $\mathcal{L}_\mathrm{p}$.

at each node are representation objects as described in Section 4.2. The scheme uses node evaluations and pruning constraint systems, as defined in Section 4.2. Relevant domain reduction techniques are used to reduce node ranges and, in particular, domains of variables.

Let $\mathcal{G}$ be the DAG representation of the input constraint system. The **CIRD** algorithm, uses two waiting lists. The first one, $\mathcal{L}_e$, stores the nodes waiting for node evaluation. The second waiting list, $\mathcal{L}_p$, stores the nodes waiting for node range pruning. Note that each node can appear once at a time in one waiting list, but may appear in both waiting lists. The set of real inclusion representations used in the scheme is denoted by $\mathcal{E}$. Suppose each real inclusion representation in $\mathcal{E}$ provides elementary operations that are inclusion functions of their real-valued counterparts. In Algorithm 1, we present the main steps **CIRD** with inline detailed descriptions.

Proposition 1 *We define a function $F : \mathbb{I}^n \times 2^{\mathbb{R}^n} \to \mathbb{I}^n$ to represent the* **CIRD** *algorithm. This function takes as input the variable domains (in the form of an interval box* $\mathbf{B}$*) and the exact solution set, S, of the input problem. The function F returns an interval box, denoted by $F(\mathbf{B}, S)$, that represents the variable domains of the output of the* **CIRD** *algorithm. The* **CIRD** *algorithm terminates at a finite number of iterations and the following properties hold:*

(i) $F(\mathbf{B}, S) \subseteq \mathbf{B}$ *(Contractiveness)*
(ii) $F(\mathbf{B}, S) \supseteq \mathbf{B} \cap S$ *(Correctness)*

Proof The proof is trivial due to the finite nature of floating-point numbers and the fact that the node ranges are never inflated during the computations. □

Example 16 Figure 6 illustrates the distribution of auxiliary variables (ϵ_1 and ϵ_2) when using affine arithmetic and interval arithmetic to perform node evaluations on the DAG representation of the problem in Example 12. The sets $\mathcal{S}$ and $\mathcal{T}$ in Algorithm 1 can be chosen as follows: $\mathcal{S} = \mathcal{T} = \{\mathbb{I}, \mathbb{A}\}$. The set $\mathcal{T}$ can be partitioned into two subsets: $\mathcal{U}_1 = \{\mathbb{I}\}, \mathcal{U}_2 = \{\mathbb{A}\}$. Example 15 gives the pruning constraint systems (PCS) for each node.

Consider the node $\mathbf{N}_7$ (Fig. 6a). Applying a linear programming technique to the system PCS($\mathbf{N}_7, \{\mathbb{A}\}$), we get $\epsilon_1 = 1$ and $\epsilon_2 = 1$. Any node involving an auxiliary variable of which the domain has just been reduced, ϵ_1 or ϵ_2, will be in the set $\mathcal{K}$ in Algorithm 1, namely, $\mathcal{K} = \{\mathbf{N}_1, \ldots, \mathbf{N}_7\}$. Of course, only a subset $\mathcal{H}$ of $\mathcal{K}$ should be considered for node range updates. More details are described in Section 5.5.2. This shows that, in some cases, *every node in a DAG representation may be chosen for a node range update, not only the descendants of the current node*. Therefore, combination strategies based on the **CIRD** scheme can freely choose the set $\mathcal{H}$ of nodes for node range updates, depending on the nature of the underlying inclusion techniques.

Consider the node $\mathbf{N}_6$ (Fig. 6b). Suppose we do not applying linear programming techniques to PCS($\mathbf{N}_6, \{\mathbb{A}\}$), but use some symbolic reasoning. For example, from the first equation of PCS($\mathbf{N}_6, \{\mathbb{A}\}$), we have an equivalent equation: $\vartheta_{\mathbf{N}_3} = 2\vartheta_{\mathbf{N}_4} - \vartheta_{\mathbf{N}_5} + \vartheta_{\mathbf{N}_6}$. Substituting $\vartheta_{\mathbf{N}_4}$, $\vartheta_{\mathbf{N}_5}$, and $\vartheta_{\mathbf{N}_6}$ from the last three equations of PCS($\mathbf{N}_6, \{\mathbb{A}\}$) into this equation, we will see that $\vartheta_{\mathbf{N}_3}$ contains not only the auxiliary variable ϵ_1 but

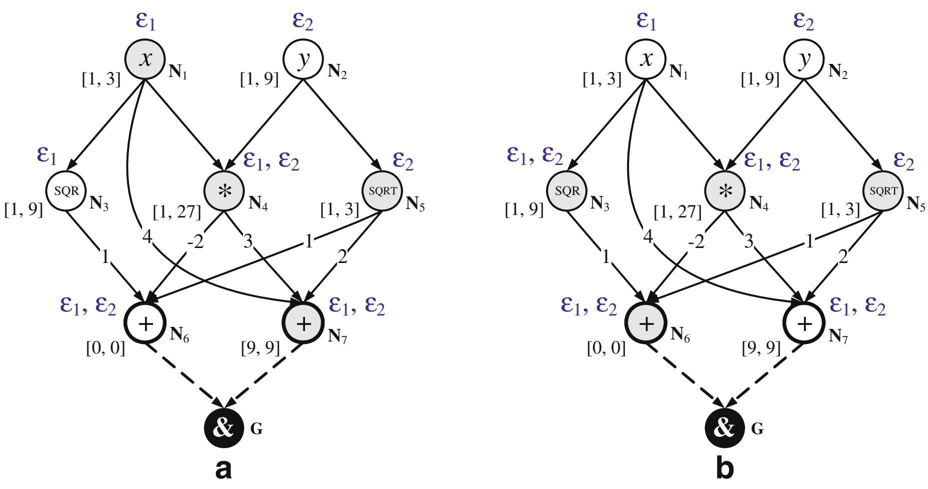


Fig. 6 The distribution of auxiliary variables (ϵ_1, ϵ_2) in the DAG representation in Example 12: the grey nodes are the nodes involving a pruning constraint system (PCS): **a** at $\mathbf{N}_7$; and **b** at $\mathbf{N}_6$

also the auxiliary variable ϵ_2, in general. This shows that *the distribution of auxiliary variables in a DAG representation may be changed during computations*.

5 Specific combination strategies

In this section, we propose some simple strategies for each step in the **CIRD** scheme. These strategies are based on two inclusion representations, $\mathbb{I}$ and $\mathbb{A}$. By combining different strategies at all the steps we get different combination strategies for constraint propagation, as instances of **CIRD**.

5.1 Step 1a: Initial node evaluation

In our implementation, we use a recursive evaluation procedure given in Procedure **RecursiveNodeEvaluation** for the visit at Step 1a. If this procedure exits with an infeasible status, the main algorithm invoking it will terminate with an infeasible status.

```
Procedure RecursiveNodeEvaluation(in: a node $\mathbf{N}$)
    if $\mathbf{N}$ is a leaf or $\mathbf{N}$ has been visited then return ;
    foreach $\mathbf{C} \in$ children($\mathbf{N}$) do RecursiveNodeEvaluation($\mathbf{C}$);
    foreach $\mathcal{I} \in \mathcal{E}$ do NEV($\mathbf{N}, \mathcal{I}$);        ◀ $\mathcal{E}$ is the set of real inclusion representations.
    Mark $\mathbf{N}$ as visited;
    if infeasible status is detected then exit(infeasible);
```

Example 17 We continue to consider the problem of Example 12. After performing the initial node evaluation by using interval arithmetic and revised affine arithmetic, we have:

$$\begin{aligned}
\tau_{\mathbf{N}_3} &= \mathbb{I}(\mathbf{N}_3) = [1,9]; & \mathbb{A}(\mathbf{N}_3) &= 4.5 + 4\epsilon_1 + 0.5[-1,1];\\
\tau_{\mathbf{N}_4} &= \mathbb{I}(\mathbf{N}_4) = [1,27]; & \mathbb{A}(\mathbf{N}_4) &= 10 + 5\epsilon_1 + 8\epsilon_2 + 4[-1,1];\\
\tau_{\mathbf{N}_5} &= \mathbb{I}(\mathbf{N}_5) = [1,3]; & \mathbb{A}(\mathbf{N}_5) &= 2.125 + \epsilon_2 + 0.125[-1,1];\\
\tau_{\mathbf{N}_6} &= \mathbb{I}(\mathbf{N}_6) = [0,0]; & \mathbb{A}(\mathbf{N}_6) &= -13.375 - 6\epsilon_1 - 15\epsilon_2 + 8.625[-1,1];\\
\tau_{\mathbf{N}_7} &= \mathbb{I}(\mathbf{N}_7) = [9,9]; & \mathbb{A}(\mathbf{N}_7) &= 42.25 + 19\epsilon_1 + 26\epsilon_2 + 12.25[-1,1];
\end{aligned}$$

(see the distribution of auxiliary variables in Fig. 6).

5.2 Step 1b: Initialization of waiting lists

We use the process described in the **CIRD** scheme in Algorithm 1.

Example 18 Consider Example 12. After performing this step with $\mathcal{S} = \{\mathbb{I}, \mathbb{A}\}$, we have:

$$\mathcal{L}_\mathrm{e} = \emptyset;\ \ \mathcal{L}_\mathrm{p} = \{(\mathbf{N}_6; \mathbb{I}, \mathbb{A}),\ (\mathbf{N}_7; \mathbb{I}, \mathbb{A})\}.$$

5.3 Step 2a: Getting the next node

At first, we assign a *node level* to each node in the DAG representation of the considered constraint system such that each node has a level smaller than the levels of their descendants. Hence, an ordering, as in Theorem 1, can be obtained easily by sorting the levels of nodes.

```
Procedure NodeLevel(in: $\mathbf{N}$;in/out:$V_{lvl}$)
    for each $\mathbf{C} \in children(\mathbf{N})$ do
        $V_{lvl}[\mathbf{C}] := \max\{V_{lvl}[\mathbf{C}], V_{lvl}[\mathbf{N}] + 1\}$;
        NodeLevel($\mathbf{C}, V_{lvl}$);
    end-for
end
```

Procedure **NodeLevel** gives a simple procedure to compute a vector V_{lvl} of node levels (it must be invoked invoked at all the nodes representing the active constraints). Figure 4 illustrates the node levels for the constraint system given in Example 12. The node levels are given in brackets next to the node names. Figure 6 illustrates the distribution of auxiliary variables. The list $\mathcal{L}_\mathrm{p}$ is sorted in the ascending order of node levels. This ensures that each node will be taken into pruning processes before its descendants, thus reducing redundant computations. Similarly, the list $\mathcal{L}_\mathrm{e}$ is sorted in the descending order of node levels to ensure that each node will be evaluated before its ancestors.

There are three simple strategies to get the next node from the union of two waiting lists, $\mathcal{L}_\mathrm{e} \cup \mathcal{L}_\mathrm{p}$:

- Get the next node from $\mathcal{L}_\mathrm{p}$ whenever $\mathcal{L}_\mathrm{p}$ is not empty. This strategy is called the *pruning-first strategy*, which gives the priority to the pruning phase.
- Get the next node from one of the two waiting lists until it becomes empty, then switch to the other list.
- Get the next node from one of the two waiting lists in a rotational manner.

In our implementation, we use the pruning-first strategy. More involved strategies for selecting the next node can be used as alternatives, for example, based on the pruning efficiency of nodes.

5.4 Step 2b: Node evaluation

For the node evaluation at each node **N**, we can perform $\texttt{NEV}(\mathbf{N}, \mathbb{A})$ and $\texttt{NEV}(\mathbf{N}, \mathbb{I})$ in any order, if **N** is not the ground node. At Step 2b(ii), Step 2b(iii) and Step 2c(iii), we only count on the change of $\tau_\mathbf{N}$ in our current implementation. The change of $\tau_\mathbf{N}$ is often considered enough if the ratio of the new width to the old width is less than a number predefined $r_\mathrm{w} \in (0, 1)$ and the difference between the old width and the new width is greater than a predefined number $d_\mathrm{w} > 0$. More complicated criteria that have been used in constraint programming can be used as alternatives.

5.5 Step 2c: Node pruning

The subset $\mathcal{T}$ (in the **CIRD**) at this step can be chosen as $\{\mathbb{I}, \mathbb{A}\}$. For node pruning, we use $\texttt{PCS}(\mathbf{N}, \{\mathbb{I}\})$ and a linear subsystem of $\texttt{PCS}(\mathbf{N}, \{\mathbb{A}\})$ defined as follows:

$$\texttt{PCS}(\mathbf{N}, \{\mathbb{I}\}) \equiv \left\{\begin{array}{l} f(\vartheta_{\mathbf{C}_1}, \ldots, \vartheta_{\mathbf{C}_k}) = \vartheta_\mathbf{N}; \\ \vartheta_\mathbf{N} \in \tau_\mathbf{N}; \\ \bigwedge_{i=1}^{k} (\vartheta_{\mathbf{C}_i} \in \tau_{\mathbf{C}_i}); \end{array}\right\} \qquad \text{if } \mathbf{N} \text{ is not ground;}$$

$$\texttt{PCS}_L(\mathbf{N}, \{\mathbb{A}\}) \equiv \begin{cases} \left\{\bigwedge_{i=1}^{k} \texttt{ICS}(\mathbb{A}(\mathbf{C}_i), \tau_{\mathbf{C}_i})\right\} & \text{if } \mathbf{N} \text{ is ground,} \\ \left\{\texttt{ICS}(\mathbb{A}(\mathbf{N}), \tau_\mathbf{N}) \wedge \bigwedge_{i=1}^{k} \texttt{ICS}(\mathbb{A}(\mathbf{C}_i), \tau_{\mathbf{C}_i})\right\} & \text{otherwise;} \end{cases}$$

where the inclusion constraint systems are defined as

$$\texttt{ICS}(\mathbb{A}(\mathbf{M}), D) \equiv \left\{\begin{array}{l} x_{\mathbf{M},0} + \sum_{i=1}^{k} x_{\mathbf{M},i}\epsilon_i + e_\mathbf{M}\epsilon_\mathbf{M} = \vartheta_\mathbf{M}; \\ \epsilon_i \in [-1, 1] \quad (i = 1, \ldots, n); \\ \epsilon_\mathbf{M} \in [-1, 1];\ \vartheta_\mathbf{M} \in D; \end{array}\right\}.$$

In general, we have

$$\texttt{PCS}(\mathbf{N}, \{\mathbb{A}\}) \equiv \texttt{PCS}(\mathbf{N}, \{\mathbb{I}\}) \wedge \texttt{PCS}_L(\mathbf{N}, \{\mathbb{A}\}).$$

The system $\texttt{PCS}_L(\mathbf{N}, \{\mathbb{A}\})$ contains the linear constraints of $\texttt{PCS}(\mathbf{N}, \{\mathbb{A}\})$.

Example 19 We continue with Example 12. Some examples of pruning constraint systems where given in Example 15. We then have:

$$\texttt{PCS}_L(\mathbf{G}, \{\mathbb{A}\}) \equiv \begin{cases} -13.375 - 6\epsilon_1 - 15\epsilon_2 + 8.625\epsilon_{\mathbf{N}_6} = \vartheta_{\mathbf{N}_6}; \\ 42.25 + 19\epsilon_1 + 26\epsilon_2 + 12.25\epsilon_{\mathbf{N}_7} = \vartheta_{\mathbf{N}_7}; \\ \vartheta_{\mathbf{N}_6} \in [0, 0]; \vartheta_{\mathbf{N}_7} \in [9, 9]; \\ (\epsilon_1, \epsilon_2, \epsilon_{\mathbf{N}_6}, \epsilon_{\mathbf{N}_7}) \in [-1, 1]^4; \end{cases}$$

$$\texttt{PCS}_L(\mathbf{N}_6, \{\mathbb{A}\}) \equiv \begin{cases} 4.5 + 4\epsilon_1 + 0.5\epsilon_{\mathbf{N}_3} = \vartheta_{\mathbf{N}_3}; \\ 10 + 5\epsilon_1 + 8\epsilon_2 + 4\epsilon_{\mathbf{N}_4} = \vartheta_{\mathbf{N}_4}; \\ 2.125 + \epsilon_2 + 0.125\epsilon_{\mathbf{N}_5} = \vartheta_{\mathbf{N}_5}; \\ -13.375 - 6\epsilon_1 - 15\epsilon_2 + 8.625\epsilon_{\mathbf{N}_6} = \vartheta_{\mathbf{N}_6}; \\ \vartheta_{\mathbf{N}_3} \in [1, 9];\ \vartheta_{\mathbf{N}_4} \in [1, 27];\ \vartheta_{\mathbf{N}_5} \in [1, 3];\ \vartheta_{\mathbf{N}_6} \in [0, 0]; \\ (\epsilon_1, \epsilon_2, \epsilon_{\mathbf{N}_3}, \epsilon_{\mathbf{N}_4}, \epsilon_{\mathbf{N}_5}, \epsilon_{\mathbf{N}_6}) \in [-1, 1]^6; \end{cases}$$

$$\texttt{PCS}_L(\mathbf{N}_7, \{\mathbb{A}\}) \equiv \begin{cases} 2 + \epsilon_1 = \vartheta_{\mathbf{N}_1}; \\ 10 + 5\epsilon_1 + 8\epsilon_2 + 4\epsilon_{\mathbf{N}_4} = \vartheta_{\mathbf{N}_4}; \\ 2.125 + \epsilon_2 + 0.125\epsilon_{\mathbf{N}_5} = \vartheta_{\mathbf{N}_5}; \\ 42.25 + 19\epsilon_1 + 26\epsilon_2 + 12.25\epsilon_{\mathbf{N}_7} = \vartheta_{\mathbf{N}_7}; \\ \vartheta_{\mathbf{N}_1} \in [1, 3];\ \vartheta_{\mathbf{N}_4} \in [1, 27];\ \vartheta_{\mathbf{N}_5} \in [1, 3];\ \vartheta_{\mathbf{N}_7} \in [9, 9]; \\ (\epsilon_1, \epsilon_2, \epsilon_{\mathbf{N}_4}, \epsilon_{\mathbf{N}_5}, \epsilon_{\mathbf{N}_7}) \in [-1, 1]^5; \end{cases}$$

All these systems are linear; hence, we can use linear programming techniques to reduce domains. Tight safe bounds can often be obtained by using the technique proposed in [36].

The combination of the following backward propagation and affine pruning techniques results in different strategies for the node pruning phase in the **CIRD** scheme.

5.5.1 Backward propagation

If **N** is not the ground, the domains of the variables of the constraint system $\texttt{PCS}(\mathbf{N}, \{\mathbb{I}\})$ can be pruned by a domain reduction technique called *backward propagation*. In brief, let f be the elementary operation represented by a node **N**, we then have the relation $\vartheta_{\mathbf{N}} = f(\vartheta_{\mathbf{C}_1}, \ldots, \vartheta_{\mathbf{C}_k})$. The purpose of backward propagation is to compute, at a low cost, an enclosure of the i-th projection of the relation $\vartheta_{\mathbf{N}} = f(\vartheta_{\mathbf{C}_1}, \ldots, \vartheta_{\mathbf{C}_k})$ onto $\vartheta_{\mathbf{C}_i}$ by using interval arithmetic. Only k nodes $\mathbf{C}_1, \ldots, \mathbf{C}_k$

possibly contain the variables of which the domains are pruned in the above backward propagation. Hence, after performing this backward propagation, at Step 2c(iii) we only need to consider k nodes $\mathcal{H} = \{\mathbf{C}_1, \ldots, \mathbf{C}_k\}$ for node range updates and for putting into the waiting lists.

5.5.2 Affine pruning

In revised affine arithmetic, $\mathbb{A}$, each variable of the input constraint system is associated with one noise symbol ϵ_i (for $i = 1, \ldots, n$). The system $\texttt{PCS}_L(\mathbf{N}, \{\mathbb{A}\})$ is a linear constraint system; therefore, the domains of the variables of $\texttt{PCS}_L(\mathbf{N}, \{\mathbb{A}\})$ can be pruned by using a linear programming technique supplemented with the technique proposed in [36]. If the operation represented by $\mathbf{N}$ is linear, we can apply a linear programming technique to $\texttt{PCS}(\mathbf{N}, \{\mathbb{A}\})$, instead of $\texttt{PCS}_L(\mathbf{N}, \{\mathbb{A}\})$, to get tighter bounds on the variables. For efficiency, only the domains of the variables $\vartheta_{\mathbf{C}_1}, \ldots, \vartheta_{\mathbf{C}_k}$ and/or $\epsilon_1, \ldots, \epsilon_n$ need to be pruned. We can devise three possible pruning strategies for Step 2c(iii):

1. The first strategy only requires to prune the domains of $\vartheta_{\mathbf{C}_1}, \ldots, \vartheta_{\mathbf{C}_k}$;
2. The second strategy only requires to prune the domains of the auxiliary variables $\epsilon_1, \ldots, \epsilon_n$;
3. The third strategy prunes the domains of both $\left\{\vartheta_{\mathbf{C}_1}, \ldots, \vartheta_{\mathbf{C}_k}\right\}$ and $\{\epsilon_1, \ldots, \epsilon_n\}$.

For the first strategy, we only need to consider $\mathcal{H} = \{\mathbf{C}_1, \ldots, \mathbf{C}_k\}$ for node range updates. For the last two strategies, the set $\mathcal{H}$ can be chosen as any subset of the set of $\mathsf{descendants}(\mathbf{N})$ for which the noise variables in $\mu_{\mathbb{A}}$ have just been pruned. In our implementation, we use the second pruning strategy with two options for $\mathcal{H}$: (*i*) the set $\mathsf{descendants}(\mathbf{N})$; and (*ii*) the set of initial variables associated with $\epsilon_1, \ldots, \epsilon_n$.

If, for each $i \in \{1, \ldots, n\}$, the new domain of the noise variable ϵ_i is $[a_i, b_i] \subseteq [-1, 1]$, then the node range update at $\mathbf{M} \in \mathcal{H}$ will be

$$\tau_{\mathbf{M}} := \tau_{\mathbf{M}} \cap \left(x_0^{\mathbf{M}} + \sum_{i=1}^{n} x_i^{\mathbf{M}} [a_i, b_i] + e_{\mathbf{M}}[-1, 1] \right), \tag{23}$$

where $\mathbb{A}(\mathbf{M}) = x_0^{\mathbf{M}} + \sum_{i=1}^{n} x_i^{\mathbf{M}} \epsilon_i + e_{\mathbf{M}}[-1, 1]$, and $x_i^{\mathbf{M}}$ (for $i = 1, \ldots, n$) are real numbers. It seems that Kolev affine forms (Section A.4) are more suitable for the node range update (23) than revised affine forms are because the radius of the domain of a noise variable is allowed to be less than 1. Hence, Kolev affine forms do not need to use the domain $[-1, 1]$ when restarting the computations at $\mathbf{M}$. In this case, we only need to replace the previous node range $[a_i, b_i]$ with the smallest symmetric interval containing $[a_i, b_i]$.

Remark 1 The cost of linear programming is high; therefore, we should use the affine pruning technique only if the pruning ratio is predicted to be high. For this, we propose to use the affine pruning technique only if the absolute accumulative error $e_{\mathbf{M}}$ of each node $\mathbf{M}$ involving the above linear systems is small enough. That is, the value of the operation represented by **M** lies in a thin slot between two hyperplanes, $x_0^{\mathbf{M}} + \sum_{i=1}^{n} x_i^{\mathbf{M}}\epsilon_i - e_{\mathbf{M}}$ and $x_0^{\mathbf{M}} + \sum_{i=1}^{n} x_i^{\mathbf{M}}\epsilon_i + e_{\mathbf{M}}$, in the space of the noise variables $(\epsilon_1, \ldots, \epsilon_n)$. Moreover, the affine pruning should only be used for nodes at low levels (i.e., the nodes near roots).

6 Experiments

6.1 Comparisons with interval constraint propagation techniques

We have carried out experiments on an implementation of the **CIRD[ai]** algorithm (an instance **CIRD** using affine arithmetic, interval arithmetic, interval constraint propagation, and linear programming) and two other well-known state-of-the-art interval constraint propagation techniques. The first one is a variant of box consistency [5] implemented in the commercial product ILOG Solver (v6.0), hereafter denoted by **BOX**. The second one is the **HC4** algorithm [4]. The experiments are carried out on 33 problems unbiasedly chosen form various published sources and divided into five test cases:

- The test case T_1 consists of eight easy problems with isolated solutions. These problems are solvable in short time by search using the three propagators. (See Section C.1.)
- The test case T_2 consists of four average problems with isolated solutions. These problems are solvable by the search using **CIRD[ai]** and **BOX** and cause the search using **HC4** being out of time without reaching 10^6 splits. (See Section C.2.)
- The test case T_3 consists of eight hard problems with isolated solutions. These problems cause the search using **HC4** being out of time without reaching 10^6 splits and cause the search using **BOX** either being out of time or being stopped due to running more than 10^6 splits. The search using **CIRD[ai]** accomplishes the solving for six of eight problems in this test case and runs more than 10^6 splits for the other two problems. (See Section C.3.)
- The test case T_4 consists of seven easy problems with a continuum of solutions. These problems are solvable in short time at the predefined precision 10^{-2}. (See Section C.4.)
- The test case T_5 consists of six hard problems with a continuum of solutions. These problems are solvable in short time at the predefined precision 10^{-1}. (See Section C.5.)

The timeout value is set to *10 hours* for all the test cases. *The timeout values will be used as the running time for the techniques that are out of time in the next result analysis*. For the first three test cases, the precision is 10^{-4} and search is done by bisection. For the last two test cases, we used **UCA6** a search algorithm for NCSPs with inequality constraints [44]. The comparison of interval constraint propagation techniques is based on the following measures:

- *The running time:* The relative ratio of the running time of each propagator to that of **CIRD[ai]** is called the *relative time ratio*.

Table 1 A comparison of three constraint propagation techniques **CIRD[ai]**, **BOX**, and **HC4** in solving NCSPs

Prop.	(*a*) Isolated solutions				(*b*) Continuum of solutions			
▼	Relative time ratio	Relative reduction ratio	Relative cluster ratio	Relative iteration ratio	Relative time ratio	Inner volume ratio	Relative cluster ratio	Relative iteration ratio
CIRD[ai]	1.000	1.000	1.000	1.000	1.000	0.945	1.000	1.000
BOX	1429.660	5.323	30.206	4.263	3.414	0.944	1.102	1.056
HC4	17283.614	7.722	105.825	5.515	60.101	0.941	1.168	1.118

In the section (*a*), the averages of the relative time ratios are taken over all the problems in the test cases T_1, T_2, T_3; and the averages of the other relative ratios are taken over the problems in the test case T_1 (i.e., taken over the problems that are solvable by all the techniques). In the section (*b*), the averages of the relative ratios are taken over all the problems in the test cases T_4, T_5

- *The number of boxes:* The relative ratio of the number of boxes in the output of each propagator to that of **CIRD[ai]** is called the *relative cluster ratio*.
- *The number of splits:* The number of splits in search needed to solve the problems. The relative ratio of the number of splits used by each propagator to that of **CIRD[ai]** is called the *relative iteration ratio*.
- *The volume of boxes (only for T_1, T_2, T_3):* We consider the reduction per dimension $\sqrt[d]{V/D}$; where d is the dimension, V is the total volume of the output boxes, D is the volume of the initial domains. The relative ratio of the reduction gained by each propagator to that of **CIRD[ai]** is called the *relative reduction ratio*.
- *The volume of inner boxes (only for T_4, T_5):* The ratio of the volume of inner boxes to the volume of all output boxes is called the *inner volume ratio*.

The lower the relative ratio is, the better the performance/quality is; and the higher the inner volume ratio is, the better the quality is.

The overviews of results in our experiments are given in Table 1 and Table 2. Clearly, **CIRD[ai]** is superior to **BOX** and **HC4** in performance and quality measures for the problems with isolated solutions on the chosen benchmarks. **CIRD[ai]** still outperforms the others for the problems with continuums of solutions in the benchmarks, while being a little better than the others in quality measures. Note

Table 2 The averages of the relative time ratios, which are taken over problems in each test case

Propagator	(*a*) Isolated solutions			(*b*) Continuum of solutions	
▼	Test case T_1	Test case T_2	Test case T_3	Test case T_4	Test case T_5
CIRD[ai]	1.00	1.00	1.00	1.00	1.00
BOX	8.33	6097.45	517.10	2.33	4.68
HC4	54.47	83009.81	1649.66	31.42	93.56

that the ratios for the test case T_3 are in fact much higher than shown because the solving processes using **BOX** and **HC4** do not terminate after *10 h* while the one using **CIRD[ai]** terminates in seconds or minutes.

6.2 Comparisons with linear relaxation based techniques

We now compare the proposed technique with a recent mathematical solution technique, called **A2**, in [19], which was specially designed to solve a nonlinear equation system $f(x) = 0$. The **A2** algorithm converts this system into *separable form* $g(x) = 0$, and then uses Kolev affine arithmetic to evaluate $g(x)$ and get a linear form $\mathcal{L}(x, y) = -Ax + By + b,\ x \in \mathbf{x},\ y \in \mathbf{y}$; where A and B are real matrices, b is a real vector, and $\mathbf{x}$ and $\mathbf{y}$ are interval vectors. This technique has to assume a posterior-condition that the matrix A is *invertible* in order to use the domain reduction rule of the form $\mathbf{x} := \mathbf{x} \cap (A^{-1}B\mathbf{y} + A^{-1}b)$. No *rigorous rounding control* is performed in [19]. We take the first problem in [19], which was used for illustrating the power of the **A2** algorithm in [19], for our comparison:

$$\begin{cases} ((4x_3 + 3x_6)x_3 + 2x_5)x_3 + x_4 = 0, \\ ((4x_2 + 3x_6)x_2 + 2x_5)x_2 + x_4 = 0, \\ ((4x_1 + 3x_6)x_1 + 2x_5)x_1 + x_4 = 0, \\ x_4 + x_5 + x_6 + 1 = 0, \\ (((x_2 + x_6)x_2 + x_5)x_2 + x_4)x_2 + (((x_3 + x_6)x_3 + x_5)x_3 + x_4)x_3 = 0, \\ (((x_1 + x_6)x_1 + x_5)x_1 + x_4)x_1 + (((x_2 + x_6)x_2 + x_5)x_2 + x_4)x_3 = 0, \\ x_1 \in [0.0333, 0.2173],\ x_2 \in [0.4000, 0.6000], \\ x_3 \in [0.7826, 0.9666],\ x_4 \in [-0.3071, -0.1071], \\ x_5 \in [1.1071, 1.3071],\ x_6 \in [-2.1000, -1.9000]. \end{cases} \tag{24}$$

This system has a unique solution. It is known to be very hard for interval constraint propagation techniques. To solve it on a 1.7 GHz Pentium PC at the precision 10^{-5} using a bisection search; **A2** has to perform 917 iterations in 3.46 s to reduce the problem to five boxes (see [19]); while an instance of the **CIRD** scheme, called **CIRD[ai]**, performs 54 iterations in only 0.118 s to reduce the problem to three boxes. Hence, **CIRD[ai]** is about 29.3 times faster than **A2** for the system (24), while it is more rigorous and accurate than **A2**.

Another technique against which we have compared **CIRD** is **Quad** [22] which was specifically designed to process quadratic constraints, and further extended to address power terms [21]. Note that this technique only applies to systems with (many) power terms. Again, we take as example two problems, called *Gough-Steward* and *Yama196*, which were used to illustrate the power of **Quad** in [22] and

[21], respectively. *Gough-Steward* is a non-sparse quadratic equation system of nine variables in robotics, which has four solutions [22]:

$$\begin{cases} x_1^2 + y_1^2 + z_1^2 = 31, \\ x_2^2 + y_2^2 + z_2^2 = 39, \\ x_3^2 + y_3^2 + z_3^2 = 29, \\ x_1x_2 + y_1y_2 + z_1z_2 + 6x_1 - 6x_2 = 51, \\ x_1x_3 + y_1y_3 + z_1z_3 + 7x_1 - 2y_1 - 7x_3 + 2y_3 = 50, \\ x_2x_3 + y_2y_3 + z_2z_3 + x_2 - 2y_2 - x_3 + 2y_3 = 34, \\ -12x_1 + 15y_1 - 10x_2 - 25y_2 + 18x_3 + 18y_3 = -32, \\ -14x_1 + 35y_1 - 36x_2 - 45y_2 + 30x_3 + 18y_3 = 8, \\ 2x_1 + 2y_1 - 14x_2 - 2y_2 + 8x_3 - y_3 = 20, \\ x_1 \in [-2.00; 5.57],\ \ y_1 \in [-5.57, 2.70],\ \ z_1 \in [0.00, 5.57], \\ x_2 \in [-6.25, 1.30],\ \ y_2 \in [-6.25, 2.70],\ \ z_2 \in [-2.00, 6.25], \\ x_3 \in [-5.39, 0.70],\ \ y_3 \in [-5.39, 3.11],\ \ z_3 \in [-3.61, 5.39]. \end{cases} \tag{25}$$

Yama196 is a series of high-dimensional sparse problems of n variables and n equations:

$$(n+1)^2x_{i-1} - 2(n+1)^2x_i + (n+1)^2x_{i+1} + e^{x_i} = 0,\ x_i \in [-10, 10] \quad (\text{for } i = 1, \ldots, n),$$

where $x_0 = x_{n+1} = 0$. Similarly to [21], we use the precision 10^{-8} for these problems. Table 3 presents a preliminary comparison between **CIRD[ai]** and **Quad**.

The results of **Quad** in Table 3 are copied from [21, 22], except that the ones in the cells filled with "n/a" are not yet available due to our limited access to the code of **Quad**. In Table 3, $\#S$ denotes the number of splits and $\#B$ denotes the number of boxes in the output.

Remark 2 We have also carried out experiments on the naive use of variants of affine arithmetic as a replacement of interval arithmetic in constraint propagation. That is, affine arithmetic is used only to get bounds on subexpression like in interval arithmetic. However, the performance of the obtained techniques (using affine arithmetic) is even (at least two times) worse than their counterparts (using interval arithmetic).

Table 3 A preliminary comparison of **Quad** and **CIRD[ai]**, n is the number of variables

Propagator ▶	**Quad**				**CIRD[ai]**				Time ratio
Problem ▼	$\#S$	$\#B$	Time (s)	CPU speed (GHz)	$\#S$	$\#B$	Time (s)	CPU speed (GHz)	$\frac{\textbf{Quad}}{\textbf{CIRD[ai]}}$
Gough-Steward ($n = 9$)	24	4	183.0	1.0	912	4	2.7	1.7	39.9
Yama196 ($n = 30$)	108	16	31.4	2.66	25	2	3.8	1.7	12.9
Yama196 ($n = 60$)	n/a	n/a	n/a	n/a	18	2	21.0	1.7	n/a
Yama196 ($n = 100$)	n/a	n/a	n/a	n/a	20	2	85.8	1.7	n/a
Yama196 ($n = 200$)	n/a	n/a	n/a	n/a	19	2	560.2	1.7	n/a
Yama196 ($n = 300$)	n/a	n/a	n/a	n/a	20	2	1878.1	1.7	n/a

Finally, we conducted some experiments for comparing **CIRD[ia]** with **FBPD** [45]. The latter is a particular instance of **CIRD** that uses a single inclusion representation (interval arithmetic). Our experiments show that the strengths of **FBPD** and **CIRD[ai]** are complementary. Namely, the **FBPD** algorithm outperforms the **CIRD[ai]** algorithm by an order of magnitude or more in speed when solving NCSPs with continuums of solutions. Conversely, the **CIRD[ai]** algorithm is superior to the **FBPD** algorithm by 1 to 3 orders of magnitude or more in speed when solving numerical CSPs with isolated solutions. Theoretically, it is easy to combine these two algorithms to solve numerical CSPs, in most cases, at the highest speed of both : it is only needed to *heuristically* predict if the constraints are equality, then resort to the **CIRD[ai]** algorithm, otherwise use the **FBPD** algorithm.

Although still preliminary, the experimental results presented before corroborate the intuition that combining multiple inclusion representations may have a strong impact on the efficiency of constraint-based solvers. An in depth analysis of the experimental results through:

– the comparison of the many possible combination strategies;
– and the confrontation with other existing solvers or approaches (such as [3, 11, 20]),

is now needed to give a better insight on the capabilities of **CIRD**. This will also help in identifying the most powerful combination strategies and the conditions for them to perform efficiently.

The main contribution of this paper is to provide a general and flexible scheme, **CIRD**, facilitating the experimental comparisons of many possible combination strategies.

7 Conclusion

In summary, our contribution in this paper is twofold:

1. We propose a novel generic algorithm, called **CIRD**, which enhances numerical constraint propagation by enabling the combination of multiple inclusion techniques during constraint propagation. The scheme potentially allows bringing into the constraint propagation framework the strengths of inclusion techniques coming from different fields. It uses the DAG representations of constraint systems whose flexibility and expressiveness enables devising fine-grained and flexible combination strategies for any factorable constraint system.
2. We devise from the generic scheme *specific combination strategies for numerical constraint propagation*. These strategies are designed to combine interval arithmetic, affine arithmetic, and linear programming into the framework of constraint propagation. Our experiments on a particular strategy, called **CIRD[ai]**, show that the new approach outperforms previously available constraint propagation techniques by 1 to 4 orders of magnitude or more in speed,[3] while still

[3]Our observations show that the gain in speed quickly increases when the hardness of problems increases.

being better in quality measures. It even outperforms some recent techniques specifically designed to solve particular constraint systems. The largest gain is obtained for well-constrained problems.

Moreover, the **CIRD** generic algorithm opens several potential directions for future research among which we can cite:

- Study the replacement of linear programming techniques used in the affine pruning by any domain reduction techniques for linear systems. Linear programming is indeed expensive for this purpose. For example, we can use the Krawczyk iteration for linear equations, the interval Gauss-Seidel iteration, the interval Gauss elimination or the hull method.
- Integrate Kolev generalized affine arithmetic.
- Integrate linear relaxation techniques (e.g., the ones in [14] and [8]).
- Integrate the *quadratic form* proposed in [30] and **Quad** into **CIRD**.
- Investigate the ability to integrate high-order inclusion techniques, such as the convexification techniques proposed in [15] and [41]. Rigorous bounds on polynomials can be obtained by using the technique proposed in [10].
- Study the use of the new divisions for affine forms proposed in [19] and [31] in place of the division $x/y := x * (1/y)$ implemented in **CIRD[ai]**.

Acknowledgements Support for this research was partially provided by the European Commission and the Swiss Federal Education and Science Office (OFES) through the COCONUT project (IST-2000-26063). We would like to thank ILOG for the software licenses of ILOG Solver used in the COCONUT project, and thank the COCONUT team of the University of Nantes for the **HC4** source code.

Appendix

Appendix A: Affine arithmetic

Affine arithmetic [9] is an extension of interval arithmetic that keeps track of correlations between input and computed quantities. Therefore, it is resistant to the catastrophic loss of accuracy often observed in long-running interval computations.

Affine arithmetic is somewhat similar to Hansen's generalized interval arithmetic [12], but differs in several important details. For example, in Hansen's model the internal approximation errors are combined with the input uncertainties, whereas in affine arithmetic they are represented separately, which makes it possible for the approximation error introduced at one step to be canceled out at a later step. Furthermore, in Hansen's model, but not in affine arithmetic, the joint range of two variables may be a nonconvex region. The ranges of functions obtained with affine arithmetic may be substantially more accurate than those obtained with interval arithmetic. However, the operations of affine arithmetic are often more expensive than those of interval arithmetic.

Some comparisons on interval methods and affine arithmetic methods can be found in [30, 40], and [25].

A.1 Affine form

In particular, a real-valued quantity x is represented by an *affine form* as follows

$$\hat{x} \equiv x_0 + x_1\epsilon_1 + \cdots + x_n\epsilon_n, \tag{26}$$

where $x_0, \ldots, x_n$ are real *coefficients* and $\epsilon_1, \ldots, \epsilon_n$ are *noise variables* taking value in $[-1, 1]$. Affine arithmetic allows us to use rounded floating-point arithmetic to construct *rigorous enclosures* for the ranges of operations and functions [40]. In long-running computations, the number of noise variables may be very high, but their distribution is often sparse. Therefore, we only need to store the *nonzero coefficients* and the indices of the respective noise variables of the considered affine form (26).

A.2 Affine operations

In affine arithmetic, a general *affine operation* of the form $\alpha\hat{x} + \beta\hat{y} + \gamma$ ($\alpha, \beta, \gamma \in \mathbb{R}$) can be obtained exactly, except the rounding errors, by the following formula:

$$\alpha\hat{x} + \beta\hat{y} + \gamma \equiv (\alpha x_0 + \beta y_0 + \gamma) + \sum_{i=1}^{n}(\alpha x_i + \beta y_i)\epsilon_i. \tag{27}$$

In the computations on floating-point arithmetic, if the rounding error is enclosed by $[-c, c]$, a new term $z_{\text{new}}\epsilon_{\text{new}}$ is added to represent this error, where $z_{\text{new}} = c$. The rigorous result is

$$\alpha\hat{x} + \beta\hat{y} + \gamma \equiv (\alpha x_0 + \beta y_0 + \gamma) + \sum_{i=1}^{n}(\alpha x_i + \beta y_i)\epsilon_i + z_{\text{new}}\epsilon_{\text{new}}. \tag{28}$$

Note that the *length* of the result (28) is increased by one.

A.3 Non-affine operations

Unlike the affine operations, non-affine operations such as $f(\hat{x}, \hat{y})$ can only be computed by approximations. In general, the exact result of a non-affine operation has the form $f^*(\epsilon_1, \ldots, \epsilon_n)$, where f^* is a *nonlinear function* corresponding to f. In general, this result is then approximated by an *affine function* $f^{\text{a}}(\epsilon_1, \ldots, \epsilon_n) = z_0 + z_1\epsilon_1 + \cdots + z_n\epsilon_n$. A new term $z_{\text{new}}\epsilon_{\text{new}}$ is used to represent the difference between f^* and f^{a}, namely

$$z_{\text{new}}\epsilon_{\text{new}} = f^*(\epsilon_1, \ldots, \epsilon_n) - f^{\text{a}}(\epsilon_1, \ldots, \epsilon_n). \tag{29}$$

Hence, the result is an affine form

$$z \equiv z_0 + z_1\epsilon_1 + \cdots + z_n\epsilon_n + z_{\text{new}}\epsilon_{\text{new}}, \tag{30}$$

where $\epsilon_{\text{new}} \in [-1, 1]$ and z_{new} must not be less than the *maximum absolute error*; that is,

$$z_{\text{new}} \geq \sup\left\{|f^*(\epsilon_1, \ldots, \epsilon_n) - f^{\text{a}}(\epsilon_1, \ldots, \epsilon_n)| : \forall(\epsilon_1, \ldots, \epsilon_n) \in [-1, 1]^n\right\}.$$

An important goal is to find f^{a} such that the maximum absolute error is as small as possible or can be bounded by a value z_{new} that is as small as possible. This is a subject of *Chebyshev approximation theory*, which is a well-developed field with a vast literature. In fact, a sub-theory of affine approximations is enough for affine arithmetic because we only need to construct the elementary operations in affine arithmetic. Factorable expressions/functions can be recursively composed of these elementary operations.

The reader can find in [40] some detailed rigorous procedures for constructing elementary operations, such as $1/\hat{x}$, $\hat{x}/\hat{y}$, $\hat{x}^2$, $\sqrt{\hat{x}}$, $e^{\hat{x}}$, and $\ln \hat{x}$, in affine arithmetic. Hereafter, we recall briefly the two most special operations: the multiplication and the division.

Multiplication In affine arithmetic, the multiplication of two affine forms $\hat{x} = x_0 + \sum_{i=1}^n x_i\epsilon_i$ and $\hat{y} = y_0 + \sum_{i=1}^n y_i\epsilon_i$ is another affine form $\hat{z} = z_0 + \sum_{i=1}^n z_i\epsilon_i + z_{\text{new}}\epsilon_{\text{new}}$ defined as

$$z_0 \equiv x_0 y_0, \tag{31a}$$

$$z_i \equiv x_0 y_i + y_0 x_i \quad (\text{for } i = 1, \ldots, n), \tag{31b}$$

$$z_{\text{new}} \equiv \left(\sum_{i=1}^n |x_i|\right)\left(\sum_{i=1}^n |y_i|\right). \tag{31c}$$

The number of real additions in (31) is $3n - 2$. The number of real multiplications (31) is $2n + 2$. Hence, the total number of real operations for the multiplication defined by (31) is $5n$. The multiplication defined by (31) is however not tight. Therefore, we can use the following tighter one at higher cost (it can be viewed as a special case of (38)):

$$z_0 \equiv x_0 y_0 + \frac{1}{2}\sum_{i=1}^n x_i y_i, \tag{32a}$$

$$z_i \equiv x_0 y_i + y_0 x_i \quad (\text{for } i = 1, \ldots, n), \tag{32b}$$

$$z_{\text{new}} \equiv \frac{1}{2}\sum_{i=1}^n |x_i y_i| + \sum_{1 \le i, j \le n;\ i \ne j} |x_i y_j|. \tag{32c}$$

The number of *real additions* in (32) is $n^2 + 2n - 1$. The number of *real multiplications* in (32) is $n^2 + 2n + 3$. Hence, the total number of real operations for the multiplication defined by (32) is $2(n + 1)^2$. This cost is much higher than the cost, $5n$, of (31). Proving the inclusion property of the multiplications (31) and (32) is easy, and hence is omitted.

Miyajima [32, p. 22] proposed to replace the multiplication (32) by the following:

$$z_0 \equiv x_0 y_0 + \frac{1}{2}\sum_{i=1}^n x_i y_i, \tag{33a}$$

$$z_i \equiv x_0 y_i + y_0 x_i \quad (\text{for } i = 1, \ldots, n), \tag{33b}$$

$$z_{\text{new}} \equiv \frac{1}{2}\sum_{i=1}^n |x_i y_i| + \sum_{1 \le i < j \le n} |x_i y_j + x_j y_i|. \tag{33c}$$

The multiplication (33) provides a tighter enclosure than the multiplication (32) does. They both have a number of similar real operations. However, the cost of each term $|x_i y_j + x_j y_i|$ is more expensive than the cost of $|x_i y_j| + |x_j y_i|$ when they need to be rounded upwards.

Division The division $\hat{x}/\hat{y}$ can be written as $\hat{x} * (1/\hat{y})$; hence it can be computed by one reciprocal and one multiplication. Kolev [19] proposed an improvement for computing the reciprocal $1/\hat{y}$, hence for computing $\hat{x}/\hat{y} := \hat{x} * (1/\hat{y})$. This has the interesting property that $\hat{x}/\hat{x} = 1$, which does not hold for interval arithmetic. Miyajima et al. [31] also proposed new methods to compute $\hat{x}/\hat{y}$. However, these methods are too complicated to be presented here. Hence, the reader should find the details in [19, 31].

A.4 Variants of affine arithmetic

Kolev [17] showed that, under some assumptions, it is possible to enclose a (piecewise) continuously differentiable separable function $f : \mathbf{x} \in \mathbb{I}^n \to \mathbb{R}^m$ in a linear enclosure. Namely, let

$$f(x) = \sum_{j=1}^{n} f_j(x_j),$$

where $x \equiv (x_1, \ldots, x_n)^{\mathrm{T}}$ is a vector of n real variables. It is possible to compute a linear enclosure of f of the form

$$f(x) \in \sum_{j=1}^{n} a_j x_j + \mathbf{d}_j, \tag{34}$$

where $\mathbf{d}_j \in \mathbb{I}^m$ and $a_j \in \mathbb{R}^m$. Probably inspired by the similarity between (34) and Hansen's *generalized interval* [12, 18] proposed a modified form of Hansen interval. However, Kolev's arithmetic, which is defined on Kolev's forms, is much similar to affine arithmetic than to *Hansen interval arithmetic*. We recall here the formal definition of Kolev's form.

Definition 15 (Kolev Affine Form, Kolev Interval Form) A *Kolev affine form* is a *semi-affine function* on *n noise variables* $\kappa_1, \ldots, \kappa_n$ of the form

$$\tilde{x} = c_x + \sum_{i=1}^{n} x_i \kappa_i + \mathbf{v}_x, \quad \kappa_i \in \mathbf{v}_i, \tag{35}$$

where $\mathbf{v}_i \equiv [-v_i, v_i]$ (for $i = 1, \ldots, n$) and $\mathbf{v}_x \equiv [-v_x, v_x]$ are symmetric intervals; $x_1, \ldots, x_n$ are real coefficients; and $c_x \in \mathbb{R}$. It can also be written in the interval form:

$$\tilde{\mathbf{x}} = c_x + \sum_{i=1}^{n} x_i \mathbf{v}_i + \mathbf{v}_x, \tag{36}$$

which is called a *Kolev generalized interval* associated with the above Kolev affine form.

A Kolev affine form (and also its associated generalized interval) was originally called a *generalized interval* or a *G interval* by [18]. The arithmetic that is defined on Kolev generalized intervals follows the spirit of affine arithmetic rather than the spirit of interval arithmetic; thus, we call it *Kolev generalized affine arithmetic*. Indeed, one can see hereafter that it can be viewed as a generalization of (standard) affine arithmetic.

Kolev generalized affine arithmetic An *affine operation* $\alpha x + \beta y + \gamma$, where α, β, $\gamma \in \mathbb{R}$, of two Kolev affine forms, $\tilde{x} \equiv c_x + \sum_{i=1}^{n} x_i \kappa_i + \mathbf{v}_x$ and $\tilde{y} \equiv c_y + \sum_{i=1}^{n} y_i \kappa_i + \mathbf{v}_y$, is another Kolev affine form $\tilde{z} \equiv c_z + \sum_{i=1}^{n} z_i \kappa_i + \mathbf{v}_z$, where

$$c_z \equiv \alpha c_x + \beta c_y + \gamma, \tag{37a}$$

$$z_i \equiv \alpha x_i + \beta y_i \quad (\text{for } i = 1, \ldots, n), \tag{37b}$$

$$v_z \equiv |\alpha| v_x + |\beta| v_y. \tag{37c}$$

The product $\tilde{z}$ of two Kolev affine forms, $\tilde{x} \equiv c_x + \sum_{i=1}^{n} x_i \kappa_i + \mathbf{v}_x$ and $\tilde{y} \equiv c_y + \sum_{i=1}^{n} y_i \kappa_i + \mathbf{v}_y$, is another Kolev affine form $\tilde{z} \equiv c_z + \sum_{i=1}^{n} z_i \kappa_i + \mathbf{v}_z$ defined as follows:

$$c_z \equiv c_x c_y + \frac{1}{2} \sum_{i=1}^{n} x_i y_i v_i^2, \tag{38a}$$

$$z_i \equiv c_x y_i + c_y x_i \quad (\text{for } i = 1, \ldots, n), \tag{38b}$$

$$v_z \equiv v_x v_y + |c_x| v_y + |c_y| v_x + \sum_{1 \le i, j \le n;\ i \ne j} |x_i y_j| v_i v_j + v_x \sum_{i=1}^{n} |y_i| v_i + v_y \sum_{i=1}^{n} |x_i| v_i + \frac{1}{2} \sum_{i=1}^{n} |x_i y_i| v_i^2. \tag{38c}$$

The number of *real additions* in (38) is $n^2 + 4n + 2$. The number of *real multiplications* in these formulas is $3n^2 + 4n + 8$. In present computers, the cost of a floating-point addition is quite the same as that of a floating-point multiplication, then the complexity of (38) is $4n^2 + 8n + 10$ real operations.

Kolev [18] also extended the above arithmetic for continuously differentiable elementary operations $\psi : D \subseteq \mathbb{R} \to \mathbb{R}$ by using the linear relaxation techniques. In particular, given an interval $\mathbf{x} \subseteq D$, for all $x \in \mathbf{x}$ we have

$$\psi(x) \in ax + \mathbf{d},$$

where $a \in \mathbb{R}$, $\mathbf{d} \in \mathbb{I}$. Now let us consider a Kolev affine form $\tilde{x} \equiv c_x + \sum_{i=1}^{n} x_i \kappa_i + \mathbf{v}_x$ such that $\tilde{x} \subseteq \mathbf{x}$, we then have

$$\psi(\tilde{x}) \subseteq a \left(c_x + \sum_{i=1}^{n} x_i \kappa_i + \mathbf{v}_x \right) + \mathbf{d}.$$

Hence, a Kolev affine form $\tilde{z} \equiv c_z + \sum_{i=1}^{n} z_i \kappa_i + \mathbf{v}_z$ can be obtained for $\psi(\tilde{x})$, w.r.t. the inclusion property, by defining that

$$\mathbf{d}' \equiv \mathbf{d} + a \left(\sum_{i=1}^{n} \mathbf{v}_x \right), \tag{39a}$$

$$c_z \equiv a c_x + \mathrm{mid}(\mathbf{d}'), \tag{39b}$$

$$z_i \equiv a x_i \quad (\text{for } i = 1, \ldots, n), \tag{39c}$$

$$v_z \equiv \mathrm{rad}(\mathbf{d}'). \tag{39d}$$

We can obtain a Kolev affine form (and its associated Kolev generalized interval) for any *factorable expression/function*, with respect to the inclusion property, by composing the expression/function using the above-defined elementary operations. Since most elementary operations are continuously differentiable, a Kolev affine form can be obtained for any factorable expression built on these elementary operations.

Messine affine arithmetic Historically, [28] proposed a simpler version of Kolev affine form/arithmetic before [18] did propose the above generalized version. Affine forms of [28–30] nearly resemble to Kolev affine forms in Definition 15 when fixing $v_i = 1$ for all i, and the idea of Messine affine arithmetic is similar to that of Kolev affine arithmetic in (37) and (38). In particular, a *Messine affine form* has the form

$$\check{x} \equiv x_0 + \sum_{i=1}^{n} x_i \epsilon_i + x_{n+1}[-1, 1] + x_{n+2}[0, 1] + x_{n+3}[-1, 0], \tag{40}$$

where $\epsilon_i, \ldots, \epsilon_n$ are the noise variables taking values in the interval $[-1, 1]$, as in affine forms, and the coefficients x_{n+1}, x_{n+2} and x_{n+3} are nonnegative. Suppose the Messine affine form of $\check{y}$ written similarly to that of $\check{x}$. For all real numbers $\alpha = -\beta \geq 0$ and γ, *Messine affine arithmetic* defines that

$$\begin{aligned}\check{x} + \check{y} \equiv{} & (x_0 + y_0) + \sum_{i=1}^{n}(x_i + y_i)\epsilon_i + (x_{n+1} + y_{n+1})[-1, 1] \\ & + (x_{n+2} + y_{n+2})[0, 1] + (x_{n+3} + y_{n+3})[-1, 0];\end{aligned} \tag{41}$$

$$\begin{aligned}\check{x} - \check{y} \equiv{} & (x_0 - y_0) + \sum_{i=1}^{n}(x_i - y_i)\epsilon_i + (x_{n+1} + y_{n+1})[-1, 1] \\ & + (x_{n+2} + y_{n+3})[0, 1] + (x_{n+3} + y_{n+2})[-1, 0];\end{aligned} \tag{42}$$

$$\alpha * \check{x} \equiv \alpha x_0 + \sum_{i=1}^{n} \alpha x_i \epsilon_i + \alpha x_{n+1}[-1, 1] + \alpha x_{n+2}[0, 1] + \alpha x_{n+3}[-1, 0]; \tag{43}$$

$$\beta * \check{x} \equiv \beta x_0 + \sum_{i=1}^{n} \beta x_i \epsilon_i + \alpha x_{n+1}[-1, 1] + \alpha x_{n+3}[0, 1] + \alpha x_{n+2}[-1, 0]; \tag{44}$$

$$\gamma + \check{x} \equiv (\gamma + x_0) + \sum_{i=1}^{n} x_i \epsilon_i + x_{n+1}[-1, 1] + x_{n+2}[0, 1] + x_{n+3}[-1, 0]. \tag{45}$$

The multiplication in Messine affine arithmetic is defined as follows:

$$\check{x} * \check{y} \;\equiv\; x_0 y_0 + \sum_{i=1}^{n}(x_0 y_i + x_i y_0)\epsilon_i + K_1[-1, 1] + K_2[0, 1] + K_3[-1, 0], \tag{46}$$

where K_1, K_2, and K_3 are computed by:[4]

$$K_1 \equiv |x_0|y_{n+1} + |y_0|x_{n+1} + x_{n+1}y_{n+1} + \sum_{1\le i,j\le n+3;\ i\ne j} |x_i y_j|, \tag{47a}$$

$$K_2 \equiv K_2^{\langle 0\rangle} + x_{n+2}y_{n+2} + x_{n+3}y_{n+3} + \sum_{i=1;\ x_i y_i>0}^{n} x_i y_i, \tag{47b}$$

$$K_3 \equiv K_3^{\langle 0\rangle} + \sum_{i=1;\ x_i y_i<0}^{n} |x_i y_i|, \tag{47c}$$

where $K_2^{\langle 0\rangle}$ and $K_3^{\langle 0\rangle}$ are, in turn, defined as follows:

$$K_2^{\langle 0\rangle} = \begin{cases} x_0 y_{n+2} + y_0 x_{n+2} & \text{if } x_0 \ge 0 \text{ and } y_0 \ge 0, \\ x_0 y_{n+2} - y_0 x_{n+3} & \text{if } x_0 \ge 0 \text{ and } y_0 < 0, \\ -x_0 y_{n+3} + y_0 x_{n+2} & \text{if } x_0 < 0 \text{ and } y_0 \ge 0, \\ -x_0 y_{n+3} - y_0 x_{n+3} & \text{if } x_0 < 0 \text{ and } y_0 < 0; \end{cases} \tag{48a}$$

$$K_3^{\langle 0\rangle} = \begin{cases} x_0 y_{n+3} + y_0 x_{n+3} & \text{if } x_0 \ge 0 \text{ and } y_0 \ge 0, \\ x_0 y_{n+3} - y_0 x_{n+2} & \text{if } x_0 \ge 0 \text{ and } y_0 < 0, \\ -x_0 y_{n+2} + y_0 x_{n+3} & \text{if } x_0 < 0 \text{ and } y_0 \ge 0, \\ -x_0 y_{n+2} - y_0 x_{n+2} & \text{if } x_0 < 0 \text{ and } y_0 < 0. \end{cases} \tag{48b}$$

The numbers of real additions and multiplications are $n^2 + 7n + 13$ and $n^2 + 8n + 16$, respectively. Hence, the total is $2n^2 + 15n + 29$. One can easily see that Messine's multiplication is a special case of Kolev's definition in (38). Notice that this multiplication is not as tight as the one in (33) when adapted to Messine affine forms or vice versa.

Kolev affine arithmetic Fortunately, Kolev [19] improved the multiplication in affine arithmetic by revising the formulas in (38) and (46) for computing an affine form of the product.

$$\hat{z} \equiv c_z + \sum_{i=1}^{n} z_i \kappa_i + z_{\text{new}} \kappa_{\text{new}}, \tag{49}$$

where κ_{new} is a new noise variable taking its value in $[-1, 1]$, and all the other noise variables are also in $[-1, 1]$. The new formulas are:

$$S_x \equiv \sum_{i=1}^{n} |x_i|, \quad S_y \equiv \sum_{i=1}^{n} |y_i|, \quad P \equiv \frac{1}{2}\sum_{i=1}^{n} x_i y_i, \tag{50a}$$

$$c_z \equiv c_x c_y + P, \tag{50b}$$

$$z_i \equiv c_x y_i + c_y x_i \quad (\text{for } i = 1, \ldots, n), \tag{50c}$$

$$z_{\text{new}} \equiv v_x v_y + v_y(|c_x| + S_x) + v_x(|c_y| + S_y) + S_x S_y - |P|. \tag{50d}$$

[4]There is a minor error in the multiplication of Messine affine arithmetic in [28–30]. To be correct, a term $x_{n+1}y_{n+1}$ must be added to K_1 and removed from K_2 therein. We correct this error in the version presented here.

In computations with *rigorous rounding controls*, I propose to replace P in (50a) with $\langle P \rangle \pm e_P$, where $\langle Z \rangle \pm z$ denotes some floating-point number such that $\langle Z \rangle - z \leq Z \leq \langle Z \rangle + z$ and $z \in \mathbb{F}$, and then replace (50d) with $z_{\text{new}} \equiv \lceil v_x v_y + v_y(|c_x| + S_x) + v_x(|c_y| + S_y) + S_x S_y + 2e_P - \langle P \rangle \rceil$. By this way, we avoid computing P and $|P|$ as if they were completely different expressions in order to reduce the computation cost. The other parts of (50) are rounded in the same way. In (50), the number of real additions is $4n + 5$ and the number of real multiplications is $3n + 7$. The total number of real operations is $7n + 12$. Therefore, the new multiplication (50) is much faster than the previous multiplication in (32) and (38), when n is big.

Note 2 By substituting the term $z_{\text{new}}\kappa_{\text{new}}$ in the (standard) affine form $\hat{z}$ in (49) with $\mathbf{v}_z \equiv [-z_{\text{new}}, z_{\text{new}}]$, we get a Kolev affine form $\tilde{z}' \equiv c_z + \sum_{i=1}^{n} z_i \kappa_i + \mathbf{v}_z$, which is equivalent to $\hat{z}$. However, it is not tighter than the Kolev affine form $\tilde{z}$ obtained by (38), namely,

$$\tilde{z}' \equiv \left\{ \hat{z} \mid \kappa_{\text{new}} \in [-1, 1] \right\} \supseteq \tilde{z}. \tag{51}$$

Remark 3 All affine forms presented here are easily converted to (standard) affine form by replacing each interval of them with a new noise variable. All the above improvements to variants of affine arithmetic can also be easily adapted to (standard) affine arithmetic.

Appendix B: Revised affine arithmetic

B.1 Revised affine form

Inspired by the ideas of Messine affine arithmetic and Kolev generalized affine arithmetic (see Section A.4), we propose a kind of affine form, called *revised affine form*, which is similar to the affine form (30); namely, it is similar to

$$\hat{z} \equiv z_0 + z_1\epsilon_1 + \cdots + z_n\epsilon_n + z_{\text{new}}\epsilon_{\text{new}} \tag{52}$$

but the new term $z_{\text{new}}\epsilon_{\text{new}}$ is replaced with a symmetric interval $e_z[-1, 1]$, called the *accumulative error*, that bounds the maximum error of non-affine operations. Namely, a *revised affine form* of a real-valued quantity/variable x is defined as

$$\widehat{x} = x_0 + x_1\epsilon_1 + \cdots + x_n\epsilon_n + e_x[-1, 1]. \tag{53}$$

This form consists of two separated parts: an affine part of length n and an interval part; thus, it is said to be of length n. This form is similar to, but more concise than, the Messine affine form (40). The magnitude of the accumulative error, $e_x \geq 0$, is identified by the interval part. We write $x \in \widehat{x}$ if, for each real value x of the quantity $\widehat{x}$, there exist $\epsilon_x \in [-1, 1]$ and $\epsilon_i \in [-1, 1]$ (for $i = 1, \ldots, n$) such that $x = x_0 + x_1\epsilon_1 + \cdots + x_n\epsilon_n + e_x\epsilon_x$.

In fact, revised affine form (53) is a special case of Kolev affine/interval form (see Definition 15). The two parts of a revised affine form are computed separately in an affine operation. For example, an affine operation on two revised affine forms is now defined as

$$\widehat{z} \equiv \alpha\widehat{x} + \beta\widehat{y} + \gamma = (\alpha x_0 + \beta y_0 + \gamma) + \sum_{i=1}^{n}(\alpha x_i + \beta y_i)\epsilon_i + (|\alpha|e_x + |\beta|e_y)[-1, 1]. \tag{54}$$

Therefore, during long-running computations the lengths of revised affine forms will not exceed the number of noise symbols at the beginning (which equals the number of variables of the input constraint system). Note that we should implement special affine operations separately to gain in speed. In rigorous computing, e_z is used to accumulate the rounding errors in floating-point arithmetic; namely, (54) can be interpreted as follows:

$$z_0 = \langle \alpha x_0 + \beta y_0 + \gamma \rangle \pm e_0, \tag{55a}$$

$$z_i = \langle \alpha x_i + \beta y_i \rangle \pm e_i, \tag{55b}$$

$$e_z = \left\lceil |\alpha|e_x + |\beta|e_y + \sum_{i=0}^{n} e_i \right\rceil \tag{55c}$$

Theorem 10 *The affine operation defined by* (54) *or* (55) *is an interval form of its counterpart (i.e., the real-valued operation).*

Proof This theorem is obvious; hence, the proof is omitted. □

We propose to associate each quantity $\widehat{x}$ with a data field $x_\infty \in \{-1, 0, +1\}$ to represent the half-lines of the form $[-\infty, a]$ and $[a, +\infty]$. The *revised affine form* is then interpreted as follows:[5]

$$\widehat{x} \equiv \begin{cases} [-\infty, +\infty] & \text{if } e_x = +\infty, \\ [-\infty, x_0] & \text{else if } x_\infty = -1, \\ [x_0, +\infty] & \text{else if } x_\infty = +1, \\ x_0 + x_1\epsilon_1 + \cdots + x_n\epsilon_n + e_x[-1, 1] & \text{otherwise.} \end{cases} \tag{56}$$

Remark 4 In an operation, if the domain of a variable is unbounded (i.e., in the first three cases of (56)), the other variables are converted into interval forms for that operation performed in interval arithmetic, then the result is converted back to revised affine form. Therefore, we only need to discuss about the last case of (56) in the rest of this thesis.

Notation 11 *We denote by* $\hat{\mathbb{A}}$ *the set of all affine forms and by* $\mathbb{A}$ *the set of all revised affine forms of the form* (56).

[5]For simplicity, we allow zero coefficients in the formulae here, however in implementation one should keep only nonzero coefficients and their indices.

B.2 Multiplication

Since every revised affine form of the form (53) is a special case of Kolev affine/interval forms (see Definition 15), we can apply Kolev generalized affine arithmetic to revised affine forms. Indeed, by letting the radius (v_i) of all noise variables in (38) be 1, the multiplication of revised affine forms in Kolev generalized affine arithmetic, $\widehat{z} := \widehat{x} * \widehat{y}$, is defined as follows:

$$z_0 \equiv x_0 y_0 + \frac{1}{2}\sum_{i=1}^{n} x_i y_i, \tag{57a}$$

$$z_i \equiv x_0 y_i + y_0 x_i \quad (\text{for } i = 1, \ldots, n), \tag{57b}$$

$$e_z \equiv e_x e_y + |x_0| e_y + |y_0| e_x + e_y \sum_{i=1}^{n} |x_i| + e_x \sum_{i=1}^{n} |y_i| + \sum_{1 \le i, j \le n;\ i \ne j} |x_i y_j| + \frac{1}{2}\sum_{i=1}^{n} |x_i y_i|. \tag{57c}$$

The number of real additions in (57) is $n^2 + 4n + 2$, which is the same as that in (38). However, the number of real multiplications is $n^2 + 2n + 8$, which is less than that number of (38), $3n^2 + 4n + 8$. The total number of operations is $2n^2 + 6n + 10$; hence, it is about half of that number of (38), $4n^2 + 8n + 10$. Roughly speaking, the multiplication in Kolev generalized affine arithmetic is about two times slower than its simplification (57) for revised affine forms.

Moreover, we propose a faster *multiplication* in which the product of two revised affine forms $\widehat{x} = x_0 + x_1\epsilon_1 + \cdots + x_n\epsilon_n + e_x[-1, 1]$ and $\widehat{y} = y_0 + y_1\epsilon_1 + \cdots + y_n\epsilon_n + e_y[-1, 1]$ is another revised affine form $\widehat{z} = z_0 + z_1\epsilon_1 + \cdots + z_n\epsilon_n + e_z[-1, 1]$ defined as follows:

$$S_x \equiv \sum_{i=1}^{n} |x_i|, \quad S_y \equiv \sum_{i=1}^{n} |y_i|,$$

$$S_1 \equiv 0.5 \sum_{i=1;\ x_i y_i \ge 0}^{n} x_i y_i, \quad S_2 \equiv 0.5 \sum_{i=1;\ x_i y_i < 0}^{n} x_i y_i, \tag{58a}$$

$$z_0 \equiv x_0 y_0 + (S_1 + S_2), \tag{58b}$$

$$z_i \equiv x_0 y_i + y_0 x_i \quad (\text{for } i = 1, \ldots, n), \tag{58c}$$

$$e_z \equiv e_x e_y + e_y(|x_0| + S_x) + e_x(|y_0| + S_y) + S_x S_y - (S_1 - S_2). \tag{58d}$$

The number of real additions in (58) is $4n + 5$, the number of real multiplications in (58) is $3n + 7$. The total number of real operations in (58) is $7n + 12$. This is the same as in Kolev's multiplication (50) (see Section A.4). It however provides tighter enclosures than what is provided by Kolev's multiplication (50) (see Section A.4), because $|P| = 0.5|\sum_{i=1}^{n} x_i y_i| \le 0.5 \sum_{i=1}^{n} |x_i y_i| = (S_1 - S_2)$. Note that in [43], we proposed another version of the multiplication (58) with the same tightness and the cost $8n + 10$, but with simpler formulas. **These improvements can be easily transferred to other variants of affine arithmetic such as Kolev generalized affine arithmetic** in Section A.4. In rigorous computations, we use the following formulas with error rounding controls:

$$S_x \equiv \left\lceil \sum_{i=1}^{n} |x_i| \right\rceil, \quad S_y \equiv \left\lceil \sum_{i=1}^{n} |y_i| \right\rceil, \tag{59a}$$

$$S_1 \equiv \left\langle 0.5 \sum_{i=1;\ x_i y_i \geq 0}^{n} x_i y_i \right\rangle \pm e_{n+1}, \quad S_2 \equiv \left\langle 0.5 \sum_{i=1;\ x_i y_i < 0}^{n} x_i y_i \right\rangle \pm e_{n+2}, \tag{59b}$$

$$z_0 \equiv \langle x_0 y_0 + (S_1 + S_2) \rangle \pm e_0, \tag{59c}$$

$$z_i \equiv \langle x_0 y_i + y_0 x_i \rangle \pm e_i \quad (\text{for } i = 1, \ldots, n), \tag{59d}$$

$$e_z \equiv \left\lceil e_x e_y + e_y(|x_0| + S_x) + e_x(|y_0| + S_y) + S_x S_y + (S_2 - S_1) + \sum_{i=0}^{n+2} e_i \right\rceil. \tag{59e}$$

Theorem 12 *The multiplication defined by (58) (or by (59)) is an interval form of the real multiplication; that is,* $\forall x \in \widehat{x}, \forall y \in \widehat{y} : xy \in \widehat{z} \equiv \widehat{x}\widehat{y}$.

Proof Let $x \in \widehat{x}$ and $y \in \widehat{y}$ be two revised affine forms as defined in (58) or (59). By definition, there exist two real numbers $e_x, e_y \in [-1, 1]$ such that

$$x = x_0 + \sum_{i=1}^{n} x_i \epsilon_i + e_x \epsilon_x, \quad y = y_0 + \sum_{i=1}^{n} y_i \epsilon_i + e_y \epsilon_y.$$

Let $\mathbf{e} \equiv [-1, 1]$. Since $a\epsilon_i^2 \in \frac{1}{2}(a + |a|\mathbf{e})$ for all $a \in \mathbb{R}$ and $i \in \{1, \ldots, n\}$, we have

$$\begin{aligned}
xy &= x_0 y_0 + \sum_{i=1}^{n} (x_0 y_i + x_i y_0)\epsilon_i + x_0 e_y \epsilon_y + y_0 e_x \epsilon_x + e_x e_y \epsilon_x \epsilon_y \\
&\quad + e_y \sum_{i=1}^{n} x_i \epsilon_y \epsilon_i + e_x \sum_{i=1}^{n} y_i \epsilon_x \epsilon_i + \sum_{1 \leq i, j \leq n;\ i \neq j} x_i y_j \epsilon_i \epsilon_i + \sum_{i=1}^{n} x_i y_i \epsilon_i^2 \\
&\in x_0 y_0 + \sum_{i=1}^{n} (x_0 y_i + x_i y_0)\epsilon_i + |x_0| e_y \mathbf{e} + |y_0| e_x \mathbf{e} + e_x e_y \mathbf{e} \\
&\quad + e_y \sum_{i=1}^{n} |x_i| \mathbf{e} + e_x \sum_{i=1}^{n} |y_i| \mathbf{e} + \sum_{1 \leq i, j \leq n;\ i \neq j} |x_i y_j| \mathbf{e} + \frac{1}{2} \sum_{i=1}^{n} (x_i y_i + |x_i y_i| \mathbf{e}) \\
&= \left(x_0 y_0 + \frac{1}{2} \sum_{i=1}^{n} x_i y_i \right) + \sum_{i=1}^{n} (x_0 y_i + x_i y_0)\epsilon_i \\
&\quad + \mathbf{e} \times \left(e_x e_y + e_y \sum_{i=0}^{n} |x_i| + e_x \sum_{i=0}^{n} |y_i| + \sum_{i=1}^{n} |x_i| \sum_{i=1}^{n} |y_i| - \frac{1}{2} \sum_{i=1}^{n} |x_i y_i| \right) \\
&\subseteq z_0 + \sum_{i=1}^{n} z_i \epsilon_i + e_z \mathbf{e} = \widehat{z} \qquad (\text{It follows from (58) or (59)}).
\end{aligned}$$

That is, $\forall x \in \widehat{x}, \forall y \in \widehat{y} : xy \in \widehat{z} \equiv \widehat{x} \times \widehat{y}$. The proof is hence completed. □

B.3 Division

The division $\hat{x}/\hat{y}$ can be written as $\hat{x} * (1/\hat{y})$, hence can be computed by a reciprocal and a multiplication. It is worth mentioning that [19] proposed an improvement for computing the reciprocal $1/\hat{y}$, hence for computing $\hat{x}/\hat{y} := \hat{x} * (1/\hat{y})$. This has the interesting property that $\hat{x}/\hat{x} = 1$, which does not hold for interval arithmetic. Miyajima et al. [31] also proposed new methods to compute $\hat{x}/\hat{y}$. However, these methods are too complicated to be presented here. The reader should find the details in [19, 31].

B.4 Non-affine unary operations

First, we recall the fundamental result in affine arithmetic, which is a result in *Chebyshev approximation theory* (see [40, Theorem 2]).

Theorem 13 *Let f be a bounded and* twice differentiable function *defined on some interval $[a, b]$ of which the second derivative f'' does not change sign inside $[a, b]$, where $a < b$. Let $f^{\mathrm{a}}(x) = \alpha x + \beta$ be its minimax affine approximation in $[a, b]$. Then*

- *The coefficient α is $(f(b) - f(a))/(b - a)$, the slope of the line $l(x)$ that interpolates the points $(a, f(a))$ and $(b, f(b))$;*
- *The maximum absolute error will occur twice (with the same sign) at the endpoints a and b of the range, and once (with the opposite sign) at every interior point c of $[a, b]$, where $f'(c) = \alpha$;*
- *The independent term β is such that $\alpha c + \beta = (f(c) + l(c))/2$ and the maximum absolute error is $\delta = |f(c) - l(c)|/2$.*

Second, we propose the following constructive theorem, which is inspired by Theorem 13 and the related procedures in [40], that serves as a basis to compute affine approximations of elementary univariate functions in a rigorous manner.

Theorem 14 (Chebyshev Affine Approximation) *Let f be a* differentiable function *on $[a, b]$, where a and b are real numbers and $a \le b$; that is, $f \in \mathcal{C}^1([a, b])$. Denote $d_\alpha(x) \equiv f(x) - \alpha x$. Then*

1. (a) *If $\forall x \in [a, b] : \alpha \ge f'(x)$, then $\forall x \in [a, b] : \alpha x + d_\alpha(b) \le f(x) \le \alpha x + d_\alpha(a)$;*
   (b) *If $\forall x \in [a, b] : \alpha \le f'(x)$, then $\forall x \in [a, b] : \alpha x + d_\alpha(a) \le f(x) \le \alpha x + d_\alpha(b)$.*
2. *If f' is* continuous *and* monotone increasing *on $[a, b]$, we have*
   (a) $\forall \alpha \in [f'(a), f'(b)], \exists c \in [a, b] : f'(c) = \alpha;$
   (b) *Let $g : \mathbb{R} \to \mathbb{R}$ be a function such that $g(\alpha) = d_\alpha(c)$, then*

$$\forall x \in [a, b] : \alpha x + g(\alpha) \le f(x) \le \alpha x + \max\{d_\alpha(a), d_\alpha(b)\}.$$

3. *If f' is* continuous *and* monotone decreasing *on* $[a, b]$, *we have*
   (a) $\forall \alpha \in [f'(b), f'(a)], \exists c \in [a, b] : f'(c) = \alpha$;
   (b) *Let* $g : \mathbb{R} \to \mathbb{R}$ *be a function such that* $g(\alpha) = d_\alpha(c)$, *then*

$$\forall x \in [a, b] : \alpha x + \min\{d_\alpha(a), d_\alpha(b)\} \leq f(x) \leq \alpha x + g(\alpha).$$

Proof Hereafter, we prove the results for the cases 1a and 2. The proof is analogous for the other cases. Considering the case 1a, we have

$$\begin{aligned}
&f(x) - (\alpha x + d_\alpha(b)) \\
&\quad = f(x) - (\alpha x + f(b) - \alpha b) \\
&\quad = (f(x) - f(b)) - \alpha(x - b) \\
&\quad = f'(\xi)(x - b) - \alpha(x - b) && \text{for some } \xi \in [x, b] \text{ (by the mean value theorem)} \\
&\quad = (x - b)(f'(\xi) - \alpha) \geq 0 && \text{since } x \leq b,\ f'(\xi) \leq \alpha
\end{aligned}$$

By a similar argument, we also have $f(x) - (\alpha x + d_\alpha(a)) \leq 0$. Hence, the proof of the case 1a is completed.

Because $\alpha \in [f'(b), f'(a)]$ and f' is continuous on $[a, b]$ then there exists $c \in [a, b]$ as required by the case 2a. The proof is then completed for the case 2a. Here we give the proof of the case 2b. For all $x \in [a, b]$ we have

$$\begin{aligned}
&f(x) - (\alpha x + g(\alpha)) \\
&\quad = f(x) - (\alpha x + f(c) - \alpha c) \\
&\quad = (f(x) - f(c)) - \alpha(x - c) \\
&\quad = f'(\xi)(x - c) - \alpha(x - c) && \text{for some } \xi \text{ between } x \text{ and } c \text{ (the mean value theorem)} \\
&\quad = (x - c)(f'(\xi) - f'(c)) && \text{since } \alpha = f'(c) \\
&\quad \geq 0 && \text{since } \xi \text{ is between } x \text{ and } c, \text{ and } f' \text{ is increasing}
\end{aligned}$$

Moreover, if $x \in [a, c]$, we have

$$\begin{aligned}
&f(x) - (\alpha x + d_\alpha(a)) \\
&\quad = f(x) - (\alpha x + f(a) - \alpha a) \\
&\quad = (f(x) - f(a)) - \alpha(x - a) \\
&\quad = f'(\eta)(x - a) - \alpha(x - a) && \text{for some } \eta \in [a, x] \text{ (by the mean value theorem)} \\
&\quad = (x - a)(f'(\eta) - f'(c)) && \text{since } \alpha = f'(c) \\
&\quad \leq 0 && \text{since } \eta \leq c,\ f' \text{ is monotone increasing}
\end{aligned}$$

Table 4 Examples of functions $f \in \mathcal{C}^1([a,b])$ satisfying the conditions of Theorem 14

$f(x)$	$[a,b]$ is a subset of	$f'(x)$	f'	$g(\alpha)$
x^2	$[-\infty,+\infty]$	$2x$	$\nearrow$	$-\alpha^2/4$
$\sqrt{x}$	$[0,+\infty]$	$1/(2\sqrt{x})$	$\searrow$	$1/(4\alpha) : \alpha > 0$
e^x	$[-\infty,+\infty]$	e^x	$\nearrow$	$\alpha(1-\log\alpha) : \alpha > 0$
$\log x$	$(0,+\infty]$	$1/x$	$\searrow$	$-(1+\log\alpha) : \alpha > 0$
$1/x$	$[-\infty,0)$	$-1/x^2$	$\searrow$	$-2\sqrt{-\alpha} : \alpha < 0$
$1/x$	$(0,+\infty]$	$-1/x^2$	$\nearrow$	$2\sqrt{-\alpha} : \alpha < 0$
$x^n : n \geq 2$ is even	$[-\infty,+\infty]$	nx^{n-1}	$\nearrow$	$(1-n)\sqrt[n-1]{(\alpha/n)^n}$
$x^n : n \geq 3$ is odd	$[-\infty,0]$	nx^{n-1}	$\searrow$	$(n-1)\sqrt[n-1]{(\alpha/n)^n} : \alpha \geq 0$
$x^n : n \geq 3$ is odd	$[0,+\infty]$	nx^{n-1}	$\nearrow$	$(1-n)\sqrt[n-1]{(\alpha/n)^n} : \alpha \geq 0$
$1/x^n : n \geq 2$ is even	$[-\infty,0)$; $(0,+\infty]$	$-n/x^{n+1}$	$\nearrow$	$(n+1)\sqrt[n+1]{(-\alpha/n)^n}$
$1/x^n : n \geq 1$ is odd	$[-\infty,0)$	$-n/x^{n+1}$	$\searrow$	$-(n+1)\sqrt[n+1]{(-\alpha/n)^n} : \alpha < 0$
$1/x^n : n \geq 1$ is odd	$(0,+\infty]$	$-n/x^{n+1}$	$\nearrow$	$(n+1)\sqrt[n+1]{(-\alpha/n)^n} : \alpha < 0$
$x^r : r \notin [0,1]$	$(0,+\infty]$	rx^{r-1}	$\nearrow$	$(1-r)(\alpha/r)^{(r/(r-1))} : \alpha r > 0$
$x^r : r \in (0,1)$	$(0,+\infty]$	rx^{r-1}	$\searrow$	$(1-r)(\alpha/r)^{(r/(r-1))} : \alpha > 0$

By a similar argument, we have $f(x) - (\alpha x + d_\alpha(b)) \leq 0$ for all $x \in [c,b]$. Hence, we have $f(x) \leq \alpha x + \max\{d_\alpha(a), d_\alpha(b)\}$ for all $x \in [a,b]$. The proof is then completed. □

To illustrate the usefulness of Theorem 14, we give in Table 4 the *derivative* f' and function g for some elementary operations. In Algorithm 2, we also propose an algorithm, called **SafeChebyshevApprox**$^\uparrow$, to find a safe Chebyshev affine approximation of a function $f \in \mathcal{C}^1([a,b])$ such that f' is monotone, when given the function g satisfying the conditions in Theorem 14. The following theorem guarantees the rigor of the **SafeChebyshevApprox**$^\uparrow$ algorithm, even in the presence of rounding errors.

Algorithm 2: The **SafeChebyshevApprox**$^\uparrow$ algorithm

```
  Input: $\widehat{x} \in \mathbb{A}$, $f \in \mathcal{C}^1([a,b])$, $f'$, $g$ as defined in Theorem 14.
  Output: a revised affine form $\alpha\widehat{x} + \beta + \delta[-1,1]$.
  $f_a := \lfloor f(a) \rfloor$; $f_b := \lceil f(b) \rceil$; $\alpha := \lceil (f_b - f_a)/(b-a) \rceil$;
  if $f'$ is monotone increasing on $[a,b]$ then
      $d_a := \lceil f(a) \rceil - \lfloor \alpha a \rfloor$;
1     if $\alpha > \lceil f'(b) \rceil$ then
          $d_{\min} := \lfloor f(b) \rfloor - \lceil \alpha b \rceil$; $d_{\max} := d_a$;
2     else
          $d_{\min} := \lfloor g(\alpha) \rfloor$; $d_{\max} := \max\{d_a, f_b - \lfloor \alpha b \rfloor\}$;
      end
  else                                                ◀ $f'$ is monotone decreasing on $[a,b]$
      $d_b := \lfloor f(b) \rfloor - \lceil \alpha b \rceil$;
      if $\alpha > \lceil f'(a) \rceil$ then
          $d_{\min} := d_b$; $d_{\max} := \lceil f(a) \rceil - \lfloor \alpha a \rfloor$;
      else
          $d_{\min} := \min\{f_a - \lceil \alpha a \rceil, d_b\}$; $d_{\max} := \lceil g(\alpha) \rceil$;
      end
  end
  $\beta := \mathrm{mid}([d_{\min}, d_{\max}])$; $\delta := \mathrm{rad}([d_{\min}, d_{\max}])$;
```

Theorem 15 *Let $\alpha\widehat{x}+\beta+\delta[-1,1]$ be the revised affine form produced by the* **SafeChebyshevApprox**$^{\uparrow}$ *algorithm in Algorithm 2, where $[a,b]$ is a closed interval containing $\widehat{x}\in\mathbb{A}$. Suppose $f\in\mathcal{C}^1([u,v])$ and f' is monotone on $[u,v]$, where $[u,v]\supseteq[a,b]$, such that $f'(v)\geq\lceil f'(b)\rceil$ if f' is monotone increasing or such that $f'(u)\geq\lceil f'(a)\rceil$ if f' is monotone decreasing. Then*

$$\forall x\in\widehat{x}:\ f(x)\in\alpha\widehat{x}+\beta+\delta[-1,1]. \tag{60}$$

Proof By the mean value theorem, there exists $c^*\in[a,b]$ such that

$$\begin{aligned}\alpha^*&\equiv f'(c^*)=(f(b)-f(a))/(b-a)\\ &\Rightarrow\alpha^*\leq\lceil(\lceil f(b)\rceil-\lfloor f(a)\rfloor)/(b-a)\rceil=\alpha\\ &\Rightarrow\alpha\geq f'(c^*)\geq\min\left\{f'(a),\ f'(b)\right\}\ \text{(since } f' \text{ is monotone).}\end{aligned}$$

Hereafter, we give a proof for the case that f' is monotone increasing. The proof for the case that f' is monotone decreasing is similar, where (a,u) and (b,v) exchange their roles with each other. In case $\alpha>\lceil f'(b)\rceil$, we have $\alpha>f'(x)$ for all $x\in[a,b]$. Hence, according to the case 1a of Theorem 14, for all $x\in[a,b]$, we have

$$\begin{aligned}&\alpha x+(f(b)-\alpha b)=\alpha x+d_\alpha(b)\leq f(x)\leq\alpha x+d_\alpha(a)=\alpha x+(f(a)-\alpha a)\\ &\Rightarrow\alpha x+d_{\min}\leq\alpha x+d_\alpha(b)\leq f(x)\leq\alpha x+d_\alpha(a)\leq\alpha x+d_{\max},\end{aligned}$$

because $d_{\min}=\lfloor f(b)\rfloor-\lceil\alpha b\rceil$ and $d_{\max}=\lceil f(a)\rceil-\lfloor\alpha a\rfloor$ (see Line 1 in Algorithm 2). In case $\alpha\leq f'(b)$, it follows from the case 2 of Theorem 14 and Line 2 in Algorithm 2 that

$$\alpha x+d_{\min}\leq\alpha x+g(\alpha)\leq f(x)\leq\max\{d_\alpha(a),d_\alpha(b)\}\leq\alpha x+d_{\max}.$$

In the rest, we consider the case $f'(b)<\alpha\leq\lceil f'(b)\rceil$. According to the case 1a of Theorem 14, for all $x\in[a,b]$, we have $\alpha x+d_\alpha(b)\leq f(x)\leq\alpha x+d_\alpha(a)$. Moreover, applying the case 2 of Theorem 14 to $[b,v]$, we have

$$\begin{aligned}&x\in[b,v]:\alpha x+g(\alpha)\leq f(x)\\ &\Leftrightarrow x\in[b,v]:g(\alpha)\leq f(x)-\alpha x\\ &\Rightarrow g(\alpha)\leq f(b)-\alpha b=d_\alpha(b).\end{aligned}$$

Therefore, for all $x\in[a,b]$, we have

$$\begin{aligned}&\alpha x+g(\alpha)\leq f(x)\leq\alpha x+d_\alpha(a)\leq\max\{\alpha x+d_\alpha(a),\alpha x+d_\alpha(b)\}\\ &\Rightarrow\alpha x+d_{\min}\leq f(x)\leq\alpha x+d_{\max}.\ \text{(See Line 2 in Algorithm 2.)}\end{aligned}$$

As a result, in all cases, we have $\alpha x + d_{\min} \le f(x) \le \alpha x + d_{\max}$ for all $x \in [a, b]$, thus, $\forall x \in \widehat{x} : f(x) \in \alpha\widehat{x} + \beta + \delta[-1, 1]$. The proof is hence completed. □

Algorithm 3: The **SafeChebyshevApprox**$^{\downarrow}$ algorithm

```
    Input: $\widehat{x} \in \mathbb{A}$, $f \in \mathcal{C}^1([a,b])$, $f'$, $g$ as defined in Theorem 14.
    Output: a revised affine form $\alpha\widehat{x} + \beta + \delta[-1,1]$.
    $f_a := \lceil f(a) \rceil$; $f_b := \lfloor f(b) \rfloor$; $\alpha := \lfloor (f_b - f_a)/(b-a) \rfloor$;
    if $f'$ is monotone increasing on $[a,b]$ then
        $d_b := \lceil f(b) \rceil - \lfloor \alpha b \rfloor$;
1       if $\alpha < \lfloor f'(a) \rfloor$ then
            $d_{\min} := \lfloor f(a) \rfloor - \lceil \alpha a \rceil$; $d_{\max} := d_b$;
2       else
            $d_{\min} := \lfloor g(\alpha) \rfloor$; $d_{\max} := \max\{f_a - \lfloor \alpha a \rfloor, d_b\}$;
        end
    else                                              ◀ $f'$ is monotone decreasing on $[a,b]$
        $d_a := \lfloor f(a) \rfloor - \lceil \alpha a \rceil$;
        if $\alpha < \lfloor f'(b) \rfloor$ then
            $d_{\min} := d_a$; $d_{\max} := \lceil f(b) \rceil - \lfloor \alpha b \rfloor$;
        else
            $d_{\min} := \min\{d_a, f_b - \lceil \alpha b \rceil\}$; $d_{\max} := \lceil g(\alpha) \rceil$;
        end
    end
    $\beta := \mathrm{mid}([d_{\min}, d_{\max}])$; $\delta := \mathrm{rad}([d_{\min}, d_{\max}])$;
```

The rigor of the **SafeChebyshevApprox**$^{\uparrow}$ algorithm requires that $f'(v) \ge \lceil f'(b) \rceil$ if f' is monotone increasing, or $f'(u) \ge \lceil f'(a) \rceil$ if f' is monotone decreasing (see Theorem 15). In very special cases, the domain of f may not be extended in the required side, we can use the **SafeChebyshevApprox**$^{\downarrow}$ algorithm in Algorithm 3 as an alternative. These two algorithms are sufficient for the standard elementary operations.

The following theorem states the rigor of the **SafeChebyshevApprox**$^{\downarrow}$ algorithm.

Theorem 16 *Let $\alpha\widehat{x} + \beta + \delta[-1, 1]$ be the revised affine form produced by the* **SafeChebyshevApprox**$^{\downarrow}$ *algorithm in Algorithm 3, where $[a, b]$ is a closed interval containing $\widehat{x} \in \mathbb{A}$. Suppose $f \in \mathcal{C}^1([u, v])$ and f' is monotone on $[u, v]$, where $[u, v] \supseteq [a, b]$, such that $f'(u) \le \lfloor f'(a) \rfloor$ if f' is monotone increasing or such that $f'(v) \le \lfloor f'(b) \rfloor$ if f' is monotone decreasing. Then*

$$\forall x \in \widehat{x} : f(x) \in \alpha\widehat{x} + \beta + \delta[-1, 1]. \tag{61}$$

Proof The proof is similar to the proof of Theorem 15, but (a, u) and (b, v) exchange their roles with each other. For example, the interval $[b, v]$ is replaced with the interval $[u, a]$, in the last part of the proof of Theorem 15. Note that the first result is replaced with $\alpha \le \alpha^* = f'(c^*) \le \max\{f'(a), f'(b)\}$. □

Appendix C: Test cases

C.1 Test case T_1: Problems with isolated solutions

C.1.1 Problem **BIF3**

A bifurcation problem:

$$\begin{cases} 5x^9 - 6x^5y^2 + xy^4 + 2xz = 0; \\ -2x^6y + 2x^2y^3 + 2yz = 0; \\ x^2 + y^2 = 0.265625; \end{cases}$$

where x, y, z in $[-10^8, 10^8]$.

C.1.2 Problem **ECO5**

An economic problem:

$$\begin{cases} (x_1 + x_1x_2 + x_2x_3 + x_3x_4)x_5 - 1 &= 0; \\ (x_2 + x_1x_3 + x_2x_4)x_5 - 2 &= 0; \\ (x_3 + x_1x_4)x_5 - 3 &= 0; \\ x_4x_5 - 4 &= 0; \\ x_1 + x_2 + x_3 + x_4 + 1 &= 0; \end{cases}$$

where $x_1, \ldots, x_5$ in $[-10, 10]$.

C.1.3 Problem **ECO6**

An economic problem:

$$\begin{cases} (x_1 + x_1x_2 + x_2x_3 + x_3x_4 + x_4x_5)x_6 - 1 &= 0; \\ (x_2 + x_1x_3 + x_2x_4 + x_3x_5)x_6 - 2 &= 0; \\ (x_3 + x_1x_4 + x_2x_5)x_6 - 3 &= 0; \\ (x_4 + x_1x_5)x_6 - 4 &= 0; \\ x_5x_6 - 5 &= 0; \\ x_1 + x_2 + x_3 + x_4 + x_5 + 1 &= 0; \end{cases}$$

where $x_1, \ldots, x_6$ in $[-10, 10]$.

C.1.4 Problem **ECO7**

An economic problem:

$$\begin{cases} (x_1 + x_1x_2 + x_2x_3 + x_3x_4 + x_4x_5 + x_5x_6)x_7 - 1 &= 0; \\ (x_2 + x_1x_3 + x_2x_4 + x_3x_5 + x_4x_6)x_7 - 2 &= 0; \\ (x_3 + x_1x_4 + x_2x_5 + x_3x_6)x_7 - 3 &= 0; \\ (x_4 + x_1x_5 + x_2x_6)x_7 - 4 &= 0; \\ (x_5 + x_1x_6)x_7 - 5 &= 0; \\ x_6x_7 - 6 &= 0; \\ x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + 1 &= 0; \end{cases}$$

where $x_1, \ldots, x_7$ in $[-10, 10]$.

C.1.5 Problem **ECO8**

An economic problem:

$$\begin{cases} (x_1 + x_1x_2 + x_2x_3 + x_3x_4 + x_4x_5 + x_5x_6 + x_6x_7)x_8 - 1 = 0; \\ (x_2 + x_1x_3 + x_2x_4 + x_3x_5 + x_4x_6 + x_5x_7)x_8 - 2 = 0; \\ (x_2 + x_1x_3 + x_2x_4 + x_3x_5 + x_4x_6)x_7 - 2 = 0; \\ (x_4 + x_1x_5 + x_2x_6 + x_3x_7)x_8 - 4 = 0; \\ (x_5 + x_1x_6 + x_2x_7)x_8 - 5 = 0; \\ (x_6 + x_1x_7)x_8 - 6 = 0; \\ x_7x_8 - 7 = 0; \\ x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + 1 = 0; \end{cases}$$

where $x_1, \dots, x_8$ in $[-10, 10]$.

C.1.6 Problem **NEU6**

A neurophysiology problem:

$$\begin{cases} x_1^2 + x_3^2 = 1; \\ x_2^2 + x_4^2 = 1; \\ x_5x_1^3 + x_6x_2^3 = 5; \\ x_5x_1x_3^2 + x_6x_4^2x_2 = 4; \\ x_5x_3^3 + x_6x_4^3 = 3; \\ x_5x_1^2x_3 + x_6x_2^2x_4 = 2; \\ x_1 \geq x_2; \\ x_1 \geq 0; \\ x_2 \geq 0; \end{cases}$$

where $x_1, \dots, x_6$ in $[-100, 100]$.

C.1.7 Problem **REI3**

A neurophysiology problem:

$$\begin{cases} x^2 - y^2 + z^2 = 0.5; \\ x^3 - y^3 + z^3 = 0.5; \\ x^4 - y^4 + z^4 = 0.5; \\ 2xy + 6y^2 + 2yz - 2x - 4y - 2z + 1 = 0; \end{cases}$$

where x, y, z in $[-10, 10]$.

C.1.8 Problem **WIN3**

A neurophysiology problem:

$$\begin{cases} 4xz - 4xy^2 - 16x^2 - 1 = 0; \\ 2y^2z + 4x + 1 = 0; \\ 2x^2z + 2y^2 + x = 0; \\ 2xy + 6y^2 + 2yz - 2x - 4y - 2z + 1 = 0; \end{cases}$$

where x, y, z in $[-10^5, 10^5]$.

C.2 Test case T_2: Problems with isolated solutions

C.2.1 Problem **CYC5**

A cyclic problem:

$$\begin{cases} a+b+c+d+e &= 0; \\ ab+bc+cd+de+ea &= 0; \\ abc+bcd+cde+dea+eab &= 0; \\ abcd+bcde+cdea+deab+eabc &= 0; \\ abcde-1 &= 0; \end{cases}$$

where a, b, c, d, e in $[-10, 10]$.

C.2.2 Problem **GS5.1**

A Gough Steward problem:

$$\begin{cases} x_1^2+y_1^2+z_1^2 &= 31, \\ x_2^2+y_2^2+z_2^2 &= 39, \\ x_3^2+y_3^2+z_3^2 &= 29, \\ x_1x_2+y_1y_2+z_1z_2+6x_1-6x_2 &= 51, \\ x_1x_3+y_1y_3+z_1z_3+7x_1-2y_1-7x_3+2y_3 &= 50, \\ x_2x_3+y_2y_3+z_2z_3+x_2-2y_2-x_3+2y_3 &= 34, \\ -12x_1+15y_1-10x_2-25y_2+18x_3+18y_3 &= -32, \\ -14x_1+35y_1-36x_2-45y_2+30x_3+18y_3 &= 8, \\ 2x_1+2y_1-14x_2-2y_2+8x_3-y_3 &= 20, \end{cases}$$

where $x_1 \in [0.00; 5.57]$, $y_1 \in [0.00, 2.70]$, $z_1 \in [0.00, 5.57]$, $x_2 \in [-6.25, 0.00]$, $y_2 \in [-2.00, 0.00]$, $z_2 \in [0.00, 6.25]$, $x_3 \in [-5.39, -1.00]$, $y_3 \in [-5.39, 0.00]$, $z_3 \in [0.00, 5.39]$.

C.2.3 Problem **KOL2**

Kolev's benchmark:

$$\begin{cases} ((4x_3+3x_6)x_3+2x_5)x_3+x_4 &= 0, \\ ((4x_2+3x_6)x_2+2x_5)x_2+x_4 &= 0, \\ ((4x_1+3x_6)x_1+2x_5)x_1+x_4 &= 0, \\ x_4+x_5+x_6+1 &= 0, \\ (((x_2+x_6)x_2+x_5)x_2+x_4)x_2+(((x_3+x_6)x_3+x_5)x_3+x_4)x_3 &= 0, \\ (((x_1+x_6)x_1+x_5)x_1+x_4)x_1+(((x_2+x_6)x_2+x_5)x_2+x_4)x_3 &= 0, \end{cases}$$

where $x_1 \in [0.0333, 0.2173]$, $x_2 \in [0.4000, 0.6000]$, $x_3 \in [0.7826, 0.9666]$, $x_4 \in [-0.3071, -0.1071]$, $x_5 \in [1.1071, 1.3071]$, $x_6 \in [-2.1000, -1.9000]$.

C.2.4 Problem **YAM60**

The Yama160 problem:

$$(n+1)^2x_{i-1}-2(n+1)^2x_i+(n+1)^2x_{i+1}+e^{x_i}=0, \quad (\text{for } i=1,\ldots,n),$$

where $n=60$, $x_0=x_{n+1}=0$, and $x_i \in [-10, 10]$ (for $i=1,\ldots,n$),

C.3 Test case T_3: Problems with isolated solutions

C.3.1 Problem **CAP4**

A Caprasse problem:

$$\begin{cases} y^2z + 2xyt - 2x - z & = 0; \\ -x^3z + 4xy^2z + 4x^2yt + 2y^3t + 4x^2 - 10y^2 + 4xz - 10yt + 2 & = 0; \\ 2yzt + xt^2 - x - 2z & = 0; \\ -xz^3 + 4yz^2t + 4xzt^2 + 2yt^3 + 4xz + 4z^2 - 10yt - 10t^2 + 2 & = 0; \end{cases}$$

where x, y, z, t in $\mathbb{R}$.

C.3.2 Problem **DID9**

A Didrit problem:

$$\begin{cases} x_1^2 + y_1^2 + z_1^2 = 31; \\ x_2^2 + y_2^2 + z_2^2 = 39; \\ x_3^2 + y_3^2 + z_3^2 = 29; \\ x_1x_2 + y_1y_2 + z_1z_2 + 6x_1 - 6x_2 = 51; \\ x_1x_3 + y_1y_3 + z_1z_3 + 7x_1 - 2y_1 - 7x_3 + 2y_3 = 50; \\ x_2x_3 + y_2y_3 + z_2z_3 + x_2 - 2y_2 - x_3 + 2y_3 = 34; \\ -12x_1 + 15y_1 - 10x_2 - 25y_2 + 18x_3 + 18y_3 = -32; \\ -14x_1 + 35y_1 - 36x_2 - 45y_2 + 30x_3 + 18y_3 = 8; \\ 2x_1 + 2y_1 - 14x_2 - 2y_2 + 8x_3 - y_3 = 20; \end{cases}$$

where x_i, y_i, z_i in $[-10, 10]$ for $i = 1, 2, 3$.

C.3.3 Problem **GS5.0**

A Gough Steward problem:

$$\begin{cases} x_1^2 + y_1^2 + z_1^2 & = 31, \\ x_2^2 + y_2^2 + z_2^2 & = 39, \\ x_3^2 + y_3^2 + z_3^2 & = 29, \\ x_1x_2 + y_1y_2 + z_1z_2 + 6x_1 - 6x_2 & = 51, \\ x_1x_3 + y_1y_3 + z_1z_3 + 7x_1 - 2y_1 - 7x_3 + 2y_3 & = 50, \\ x_2x_3 + y_2y_3 + z_2z_3 + x_2 - 2y_2 - x_3 + 2y_3 & = 34, \\ -12x_1 + 15y_1 - 10x_2 - 25y_2 + 18x_3 + 18y_3 & = -32, \\ -14x_1 + 35y_1 - 36x_2 - 45y_2 + 30x_3 + 18y_3 & = 8, \\ 2x_1 + 2y_1 - 14x_2 - 2y_2 + 8x_3 - y_3 & = 20, \end{cases}$$

where $x_1 \in [-2.00; 5.57]$, $y_1 \in [-5.57, 2.70]$, $z_1 \in [0.00, 5.57]$, $x_2 \in [-6.25, 1.30]$, $y_2 \in [-6.25, 2.70]$, $z_2 \in [-2.00, 6.25]$, $x_3 \in [-5.39, 0.70]$, $y_3 \in [-5.39, 3.11]$, $z_3 \in [-3.61, 5.39]$.

C.3.4 Problem **KAT8**

A Katsura problem:

$$\begin{cases} -x_1 + 2x_8^2 + 2x_7^2 + 2x_6^2 + 2x_5^2 + 2x_4^2 + 2x_3^2 + 2x_2^2 + x_1^2 & = 0; \\ -x_2 + 2x_8x_7 + 2x_7x_6 + 2x_6x_5 + 2x_5x_4 + 2x_4x_3 + 2x_3x_2 + 2x_2x_1 & = 0; \\ -x_3 + 2x_8x_6 + 2x_7x_5 + 2x_6x_4 + 2x_5x_3 + 2x_4x_2 + 2x_3x_1 + x_2^2 & = 0; \\ -x_4 + 2x_8x_5 + 2x_7x_4 + 2x_6x_3 + 2x_5x_2 + 2x_4x_1 + 2x_3x_2 & = 0; \\ -x_5 + 2x_8x_4 + 2x_7x_3 + 2x_6x_2 + 2x_5x_1 + 2x_4x_2 + x_3^2 & = 0; \\ -x_6 + 2x_8x_3 + 2x_7x_2 + 2x_6x_1 + 2x_5x_2 + 2x_4x_3 & = 0; \\ -x_7 + 2x_8x_2 + 2x_7x_1 + 2x_6x_2 + 2x_5x_3 + x_4^2 & = 0; \\ -1 + 2x_8 + 2x_7 + 2x_6 + 2x_5 + 2x_4 + 2x_3 + 2x_2 + x_1 & = 0; \end{cases}$$

where $x_1, \ldots, x_8$ in $[-10, 10]$.

C.3.5 Problem **KIN9**

A kinematics problem:

$$\begin{cases} z_1^2 + z_2^2 + z_3^2 - 12z_1 - 68 & = 0; \\ z_4^2 + z_5^2 + z_6^2 - 12z_5 - 68 & = 0; \\ z_7^2 + z_8^2 + z_9^2 - 24z_8 - 12z_9 + 100 & = 0; \\ z_1z_4 + z_2z_5 + z_3z_6 - 6z_1 - 6z_5 - 52 & = 0; \\ z_1z_7 + z_2z_8 + z_3z_9 - 6z_1 - 12z_8 - 6z_9 + 64 & = 0; \\ z_4z_7 + z_5z_8 + z_6z_9 - 6z_5 - 12z_8 - 6z_9 + 32 & = 0; \\ 2z_2 + 2z_3 - z_4 - z_5 - 2z_6 - z_7 - z_9 + 18 & = 0; \\ z_1 + z_2 + 2z_3 + 2z_4 + 2z_6 - 2z_7 + z_8 - z_9 - 38 & = 0; \\ z_1 + z_3 - 2z_4 + z_5 - z_6 + 2z_7 - 2z_8 + 8 & = 0; \end{cases}$$

where $z_1, \ldots, z_9$ in $[-1000, 1000]$.

C.3.6 Problem **REI4**

A Reinmer system:

$$\begin{cases} x^2 - y^2 + z^2 - t^2 = 0.5; \\ x^3 - y^3 + z^3 - t^3 = 0.5; \\ x^4 - y^4 + z^4 - t^4 = 0.5; \\ x^5 - y^5 + z^5 - t^5 = 0.5; \end{cases}$$

where x, y, z, t in $[-10, 10]$.

C.3.7 Problem **REI5**

A Reinmer system:

$$\begin{cases} -1 + 2x_1^2 - 2x_2^2 + 2x_3^2 - 2x_4^2 + 2x_5^2 = 0; \\ -1 + 2x_1^3 - 2x_2^3 + 2x_3^3 - 2x_4^3 + 2x_5^3 = 0; \\ -1 + 2x_1^4 - 2x_2^4 + 2x_3^4 - 2x_4^4 + 2x_5^4 = 0; \\ -1 + 2x_1^5 - 2x_2^5 + 2x_3^5 - 2x_4^5 + 2x_5^5 = 0; \\ -1 + 2x_1^6 - 2x_2^6 + 2x_3^6 - 2x_4^6 + 2x_5^6 = 0; \end{cases}$$

where $x_1, \ldots, x_5$ in $[-1, 1]$.

C.3.8 Problem **REI6**

A Reinmer system:

$$\left\langle -0.5 + \sum_{i=1}^{n} (-1)^{i+1} x_i^k = 0 \; (k = 1, \ldots, n); \;\; n = 6, \;\; x_i \in [-1, 1] \text{ (for } i = 1, \ldots, n) \right\rangle$$

C.4 Test case T_4: Problems with continuums of solutions

C.4.1 Problem **F2.2**

Tricuspoid and Circle:

$$\begin{cases} \left(x^2 + y^2 + 12x + 9\right)^2 \leq 4(2x + 3)^3; \\ x^2 + y^2 \geq 2; \end{cases}$$

where x, y in $[-2, 2]$.

C.4.2 Problem **F2.3**

Foliumd, Circle, and Trifolium:

$$\begin{cases} x^3 + y^3 \geq 3xy; \\ x^2 + y^2 \geq 0.1; \\ \left(x^2 + y^2\right)\left(y^2 + x(x + 1)\right) \leq 4xy^2; \end{cases}$$

where x, y in $[-3, 3]$.

C.4.3 Problem **S04**

Circle:

$$\left\langle x^2 + y^2 \leq 1; \;\; x, y \in [-2, 2] \right\rangle$$

C.4.4 Problem **S05**

$$\left\langle \frac{x}{\sqrt{(y - 5)^2 + 1}} \leq 1; \;\; x, y \in [1, 10] \right\rangle$$

C.4.5 Problem **S06**

$$\left\langle \frac{12y}{\sqrt{(x - 12)^2 + y^2}} \leq 10; \;\; x \in [-50, 50], \;\; y \in [0, 50] \right\rangle$$

C.4.6 Problem **S07**

$$\left\langle x^2 + y^2 \geq 20; \;\; x^2 + y^2 \leq 50; \;\; x \in [-50, 50], \;\; y \in [0, 50] \right\rangle$$

C.4.7 Problem **WP**

A Kinematic Pair (of a wheel and a pawl):

$$\left\langle 20 \le \sqrt{x^2 + y^2} \le 50,\ \frac{12y}{\sqrt{(x-12)^2 + y^2}} \le 10;\ x \in [-50, 50],\ y \in [0, 50] \right\rangle$$

C.5 Test case T_5: Problems with continuums of solutions

C.5.1 Problem **G1.1**

$$\begin{cases} x_1^2 + 0.5x_2 + 2(x_3 - 6) \ge 0; \\ x_1^2 + x_2^2 + x_3^2 \le 25; \end{cases}$$

where x_1, x_2, x_3 in $[-8, 8]$.

C.5.2 Problem **G1.1**

$$\begin{cases} x_1^2 + 0.5x_2 + 2(x_3 - 3) \ge 0; \\ x_1^2 + x_2^2 + x_3^2 \le 25; \end{cases}$$

where x_1, x_2, x_3 in $[-8, 8]$.

C.5.3 Problem **H1.1**

$$\begin{cases} x_1^2 + x_2^2 + x_3^2 \le 9; \\ (x_1 - 0.5)^2 + (x_2 - 1)^2 + x_3^2 \ge 4; \\ x_1^2 + (x_2 - 0.2)^2 \ge x_3; \end{cases}$$

where x_1, x_2, x_3 in $[-4, 4]$.

C.5.4 Problem **P1.4**

$$\begin{cases} x^2 + y^2 + z^2 <= 4; \\ (x-2)^2 + y^2 + z^2 >= 4; \end{cases}$$

where x, y, z in $[-4, 4]$.

C.5.5 Problem **P2**

$$\begin{cases} x^2 \le y, \\ \ln y + 1 \ge z, \\ xz \le 1, \end{cases}$$

where $x \in [0, 15],\ y \in [1, 200],\ z \in [-10, 10]$.

C.5.6 Problem **P3**

$$\begin{cases} x^2 \leq y, \\ \ln y + 1 \geq z, \\ xz \leq 1, \\ x^{3/2} + \ln(1.5z + 1) \leq y + 1, \end{cases}$$

where $x \in [0, 15]$, $y \in [1, 200]$, $z \in [0, 10]$.

References

1. Alefeld, G., Herzberger, J.: Introduction to Interval Computations. Academic, New York (1983)
2. Apt, R.K.: The essence of constraint propagation. Theor. Comp. Sci. **221**(1–2), 179–210 (1999)
3. Benhamou, F.: Heterogeneous constraint solving. In: Proceedings of 5th International Conference on Algebraic and Logic Programming (ALP'96). LNCS, vol. 1139, pp. 62–76, Aachen, 25–27 September 1996
4. Benhamou, F., Goualard, F., Granvilliers, L., Puget, J.-F.: Revising hull and box consistency. In: Proceedings of the International Conference on Logic Programming (ICLP'99), pp. 230–244, Las Cruces, 29 November–4 December 1999
5. Benhamou, F., McAllester, D., Van Hentenryck, P.: CLP(Intervals) revisited. In: Proceedings of the International Logic Programming Symposium, pp. 109–123. MIT, Cambridge (1994)
6. Benhamou, F., Older, W.J.: Applying interval arithmetic to real, integer and Boolean constraints. Technical Report BNR Technical Report, Bell Northern Research, Ontario, Canada (1992)
7. Benhamou, F., Older, W.J.: Applying interval arithmetic to real, integer and Boolean constraints. J. Log. Program. **32**, 1–24 (1997) (Extension of a technical report of Bell Northern Research, Canada, 1992)
8. Borradaile, G., Van Hentenryck, P.: Safe and tight linear estimators for global optimization. Math. Program. **42**, 2076–2097 (2004)
9. Comba, J.L.D., Stolfi, J.: Affine arithmetic and its applications to computer graphics. In: Proceedings of SIBGRAPI'93, Recife, October 1993
10. Garloff, J., Jansson, C., Smith, A.P.: Lower bound functions for polynomials. J. Comput. Appl. Math. **157**(1), 207–225 (2003)
11. Granvilliers, L., Benhamou, F.: Algorithm 852: RealPaver: an interval solver using constraint satisfaction techniques. ACM Trans. Math. Softw. (TOMS) **32**(1), 138–156 (2006)
12. Hansen, E.R.: A generalized interval arithmetic. In: Interval Mathematics. LNCS vol. 29, pp. 7–18. Springer, New York (1975)
13. Hansen, E.R., Walster, G.W.: Global Optimization Using Interval Analysis, 2nd edn. Marcel Dekker, New York (2004)
14. Hongthong, S., Kearfott, R.B.: Rigorous linear overestimators and underestimators. Math. Program. B (2009, in press)
15. Jansson, C.: Convex-concave extensions. BIT Numer. Math. **40**(2), 291–313 (2000)
16. Jaulin, L., Kieffer, M., Didrit, O., Walter, E.: Applied Interval Analysis, 1st edn. Springer, New York (2001)
17. Kolev, L.V.: A new method for global solution of systems of non-linear equations. Reliab. Comput. **4**, 125–146 (1998)
18. Kolev, L.V.: Automatic computation of a linear interval enclosure. Reliab. Comput. **7**, 17–18 (2001)
19. Kolev, L.V.: An improved interval linearization for solving non-linear problems. In: 10th GAMM – IMACS International Symposium on Scientific Computing, Computer Arithmetic, and Validated Numerics (SCAN2002), France, September 2002
20. Lebbah, Y.: ICOS (Interval Constraints Solver). WWW document (2003)
21. Lebbah, Y., Michel, C., Rueher, M.: Global filtering algorithms based on linear relaxations. In: Notes of the 2nd International Workshop on Global Constrained Optimization and Constraint Satisfaction (COCOS 2003), Switzerland, November 2003
22. Lebbah, Y., Rueher, M., Michel, C.: A global filtering algorithm for handling systems of quadratic equations and inequations. In: Proceedings of the 9th International Conference on Principles

and Practice of Constraint Programming (CP 2003). LNCS, vol. 2470, pp. 109–123. Springer, New York (2003)
23. Lhomme, O.: Consistency techniques for numeric CSPs. In: Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93), pp. 232–238 (1993)
24. Mackworth, A.K.: Consistency in networks of relations. Artif. Intell. **8**, 99–118 (1977)
25. Martin, R., Shou, H., Voiculescu, I., Bowyer, A., Wang, G.: Comparison of interval methods for plotting algebraic curves. Comput. Aided Geom. Des. **19**(7), 553–587 (2002)
26. McCormick, G.P.: Computability of global solutions to factorable nonconvex programs: Part I—convex underestimating problems. Math. Program. **10**, 147–175 (1976)
27. McCormick, G.P.: Nonlinear Programming: Theory, Algorithms and Applications. Wiley, New York (1983)
28. Messine, F.: New affine forms in interval branch and bound algorithms. Technical Report R2I 99-02, Université de Pau et des Pays de l'Adour (UPPA), France, October 1999
29. Messine, F.: Extensions of affine arithmetic in interval global optimization algorithms. In: SCAN 2000 and INTERVAL 2000—IMACS/GAMM International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics, Germany, 2000
30. Messine, F.: Extensions of affine arithmetic: Application to unconstrained global optimization. J. Univers. Comput. Sci. **8**(11), 992–1015 (2002)
31. Miyajima, S., Miyata, T., Kashiwagi, M.: A new dividing method in affine arithmetic. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **E86-A**(9), 2192–2196 (2003)
32. Miyajima, S.: On the improvement of the division of the affine arithmetic. Bachelor thesis, Kashiwagi Laboratory, Waseda University, Japan (2000) (It is in Japanese, but easy to guess)
33. Moore, R.E.: Interval Analysis. Prentice Hall, Englewood Cliffs (1966)
34. Moore, R.E.: Methods and Applications of Interval Analysis. SIAM Studies in Applied Mathematics. SIAM, Philadelphia (1979)
35. Neumaier, A.: Complete search in continuous global optimization and constraint satisfaction. Acta Numer. **2004**, 271–369 (2004)
36. Neumaier, A., Shcherbina, O.: Safe bounds in linear and mixed-integer programming. Math. Program. **99**, 283–296 (2004)
37. Neumaier, A.: Interval Methods for Systems of Equations. Cambridge University Press, Cambridge (1990)
38. Schichl, H., Neumaier, A.: Interval analysis on directed acyclic graphs for global optimization. J. Glob. Optim. **33**(4), 541–562 (2005)
39. Schichl, H.: Mathematical modeling and global optimization. Habilitation thesis, Faculty of Mathematics, University of Vienna, Autralia, November 2003
40. Stolfi, J., de Figueiredo, L.H.: Self-validated numerical methods and applications. In: Monograph for 21st Brazilian Mathematics Colloquium (IMPA), Brazil, July 1997
41. Tawarmalani, M., Sahinidis, N.V.: Convexification and global optimization in continuous and mixed-integer nonlinear programming. Nonconvex Optimization and Its Applications. Kluwer, Deventer (2002)
42. Van Hentenryck, P.: Numerica: a modeling language for global optimization. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97) (1997)
43. Vu, X.-H., Sam-Haroud, D., Faltings, B.: Combining multiple inclusion representations in numerical constraint propagation. In: Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004), pp. 458–467. IEEE Computer Society Press, Florida (2004)
44. Vu, X.-H., Sam-Haroud, D., Silaghi, M.-C.: Numerical constraint satisfaction problems with non-isolated solutions. LNCS, vol. 2861, pp. 194–210. Valbonne-Sophia Antipolis, France, Springer, New York (2003)
45. Vu, X.-H., Schichl, H., Sam-Haroud, D.: Interval propagation and search on directed acyclic graphs for numerical constraint solving. J. Glob. Optim. (2009, in press)