Novel Approaches to Numerical Software with Result Verification

Laurent Granvilliers[1], Vladik Kreinovich[2], and Norbert Müller[3]

[1] IRIN, University of Nantes, France
granvilliers@irin.univ-nantes.fr
[2] Department of Computer Science
University of Texas at El Paso, El Paso, TX 79968, USA
vladik@cs.utep.edu
[3] Abteilung Informatik, Universität Trier
D-54286 Trier, Germany, mueller@uni-trier.de

Abstract. Traditional design of numerical software with result verification is based on the assumption that we know the algorithm $f(x_1, \ldots, x_n)$ that transforms inputs $x_1, \ldots, x_n$ into the output $y = f(x_1, \ldots, x_n)$, and we know the intervals of possible values of the inputs. Many real-life problems go beyond this paradigm. In some cases, we do not have an algorithm f, we only know some relation (constraints) between x_i and y. In other cases, in addition to knowing the intervals $\mathbf{x}_i$, we may know some relations between x_i; we may have some information about the probabilities of different values of x_i, and we may know the exact values of some of the inputs (e.g., we may know that $x_1 = \pi/2$).
In this paper, we describe the approaches for solving these real-life problems. In Section 2, we describe interval consistency techniques related to handling constraints; in Section 3, we describe techniques that take probabilistic information into consideration, and in Section 4, we overview techniques for processing exact real numbers.

1 Introduction

Why data processing? In many real-life situations, we are interested in the value of a physical quantity y that is difficult or impossible to measure directly. Examples of such quantities are the distance to a star and the amount of oil in a given well. Since we cannot measure y directly, a natural idea is to measure y *indirectly*. Specifically, we find some easier-to-measure quantities $x_1, \ldots, x_n$ which are related to y by a known relation $y = f(x_1, \ldots, x_n)$; this relation may be a simple functional transformation, or complex algorithm (e.g., for the amount of oil, numerical solution to an inverse problem). Then, to estimate y, we first measure the values of the quantities $x_1, \ldots, x_n$, and then we use the results $\widetilde{x}_1, \ldots, \widetilde{x}_n$ of these measurements to compute an estimate $\widetilde{y}$ for y as $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$.

For example, to find the resistance R, we measure current I and voltage V, and then use the known relation $R = V/I$ to estimate resistance as $\widetilde{R} = \widetilde{V}/\widetilde{I}$.

Computing an estimate for y based on the results of direct measurements is called *data processing*; data processing is the main reason why computers were invented in

the first place, and data processing is still one of the main uses of computers as number crunching devices.

Traditional approach to numerical software with result verification: from computing with numbers to probabilities to intervals. Measurement are never 100% accurate, so in reality, the actual value x_i of i-th measured quantity can differ from the measurement result $\widetilde{x}_i$. Because of these *measurement errors* $\Delta x_i \stackrel{\text{def}}{=} \widetilde{x}_i - x_i$, the result $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ of data processing is, in general, different from the actual value $y = f(x_1, \ldots, x_n)$ of the desired quantity y [53].

It is desirable to describe the error $\Delta y \stackrel{\text{def}}{=} \widetilde{y} - y$ of the result of data processing. To do that, we must have some information about the errors of direct measurements.

What do we know about the errors Δx_i of direct measurements? First, the manufacturer of the measuring instrument must supply us with an upper bound Δ_i on the measurement error. (If no such upper bound was supplied, this would mean that no accuracy is guaranteed, and the corresponding "measuring instrument" would be practically useless.) Since the upper bound Δ_i is supplied, once we performed a measurement and got a measurement result $\widetilde{x}_i$, we know that the actual (unknown) value x_i of the measured quantity belongs to the interval $\mathbf{x}_i = [\underline{x}_i, \overline{x}_i]$, where $\underline{x}_i = \widetilde{x}_i - \Delta_i$ and $\overline{x}_i = \widetilde{x}_i + \Delta_i$.

In many practical situations, we not only know the interval $[-\Delta_i, \Delta_i]$ of possible values of the measurement error; we also know the probability of different values Δx_i within this interval. This knowledge underlies the traditional engineering approach to estimating the error of indirect measurement, in which we assume that we know the probability distributions for measurement errors Δx_i.

In practice, we can determine the desired probabilities of different values of Δx_i by comparing the results of measuring with this instrument with the results of measuring the same quantity by a standard (much more accurate) measuring instrument. Since the standard measuring instrument is much more accurate than the one used, the difference between these two measurement results is practically equal to the measurement error; thus, the empirical distribution of this difference is close to the desired probability distribution for measurement error. There are two cases, however, when this determination is not done:

- First is the case of cutting-edge measurements, e.g., measurements in fundamental science. When a Hubble telescope detects the light from a distant galaxy, there is no "standard" (much more accurate) telescope floating nearby that we can use to calibrate the Hubble: the Hubble telescope is the best we have.
- The second case is the case of measurements on the shop floor. In this case, in principle, every sensor can be thoroughly calibrated, but sensor calibration is so costly – usually costing ten times more than the sensor itself – that manufacturers rarely do it.

In both cases, we have no information about the probabilities of Δx_i; the only information we have is the upper bound on the measurement error.

In this case, after we performed a measurement and got a measurement result $\widetilde{x}_i$, the only information that we have about the actual value x_i of the measured quantity is that it belongs to the interval $\mathbf{x}_i = [\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$. In such situations, the only

information that we have about the (unknown) actual value of $y = f(x_1, \ldots, x_n)$ is that y belongs to the range $\mathbf{y} = [\underline{y}, \overline{y}]$ of the function f over the box $\mathbf{x}_1 \times \ldots \times \mathbf{x}_n$:

$$\mathbf{y} = [\underline{y}, \overline{y}] = \{f(x_1, \ldots, x_n) \,|\, x_1 \in \mathbf{x}_1, \ldots, x_n \in \mathbf{x}_n\}.$$

The process of computing this interval range based on the input intervals $\mathbf{x}_i$ is called *interval computations*; see, e.g., [30–32, 44].

Limitations of the traditional approach. Traditional design of numerical software with result verification is based on the assumption that we know the algorithm $f(x_1, \ldots, x_n)$ that transforms input $x_1, \ldots, x_n$ into the output $y = f(x_1, \ldots, x_n)$, and we know the intervals $\mathbf{x}_1, \ldots, \mathbf{x}_n$ of possible values of the inputs. Many real-life problems go beyond this paradigm:

- In some cases, we do not have an algorithm f, we only know some relation (constraints) between x_i and y.
- In other cases, in addition to knowing the intervals $\mathbf{x}_i$ of possible values of x_i, we may have some additional information:
  - we may have known some relation *between* different quantities x_i;
  - we may also have some additional information about each of these quantities:
    - we may have some information about the *probabilities* of different values of x_i;
    - in some cases, we may even know the *exact* values of some of the inputs (e.g., we may know that $x_1 = \pi/2$).

What we need. In view of the above limitations, in addition to traditional interval techniques, we must also have:

- techniques that translate known constraints between x_i and y into an algorithm that inputs the values $x_1, \ldots, x_n$ and computes the value(s) y that satisfies all the given constraints (for given values of x_i);
- techniques that use the algorithm f, the ranges $\mathbf{x}_i$, *and* additional constraints between x_i and y to get a better estimate for the range of possible values of y;
- techniques that use the additional information about probabilities of different values of $x_i \in \mathbf{x}_i$ to come up with the information about the probabilities of possible values of $y = f(x_1, \ldots, x_n)$; and
- techniques that would enable us to deal with *exact* real numbers in addition to the numbers known with interval uncertainty.

What we are planning to do. In this paper, we describe the approaches for solving these real-life problems:

- In Section 2, written by L. Granvilliers, we describe interval consistency techniques related to handling constraints–both constraints that are known *instead* of the algorithm f and *in addition* to the algorithm f.
- In Section 3, written by V. Kreinovich, we describe techniques that take probabilistic information into consideration.
- Finally, in Section 4, written by N. Müller, we overview techniques for processing exact real numbers.

Why we decided to get together. At first glance, this paper may seem quite inhomogeneous. As we have mentioned, this paper consists of three separate parts, treating three different topics. Each of these three parts concerns an important practical problem, but, as a reader may notice, as of now, there is little interaction between the three parts.

A reader may ask: why did we decide to make it a joint paper as opposed to three separate papers on three topics? The main reason for this is that we have a joint vision – which we describe in this Introduction. According to this vision, to get more practical numerical software with result verification, we must overcome the above described limitations and thus, we must move from the traditional interval techniques to the techniques that incorporate interval consistency and constraint techniques, interval probabilistic techniques, and techniques for handling exact real numbers.

Ideally, we would like all these techniques to be incorporated in a single tool. Yes, at present, there is very little interaction, little integration between these three techniques. By presenting different techniques within a single paper, we want to emphasize the need for integration and to encourage the readers to think not only about the further development of each of these techniques but also about their possible integration.

Let us work together towards this ambitious but noble goal!

2 Interval Consistency Techniques

Mathematical modeling is heavily used to simulate real-life phenomena from engineering, biology or economics. This is a mean for analysis of behavior, optimization or simulation of extreme situations. In many applications the problem is to solve (in)equality or differential equation systems, which may be parametric. Moreover observed data are often uncertain. Uncertainty can be efficiently handled by interval methods [43, 47, 46], implementing set computations over real numbers to derive global information on systems. Recently constraint propagation using consistency techniques [5, 60, 20] have been shown to enhance pure interval methods.

Consistency techniques over real numbers originate from two concurrent works, the introduction of continuous domains in the constraint satisfaction framework [19] and the use of interval arithmetic in constraint logic programming [16] in order to define a logical meaning of arithmetic. In Cleary's work constraint systems are processed by hull-consistency, a consistency property exploiting the convex hull of (in)equalities over the reals. These pioneering ideas have been extended in several ways, *e.g.*, for heterogeneous constraint processing [6] or implementing strong consistencies [40]. `CLP(BNR)` [50] was the premier CLP system using interval consistency techniques.

The next revolution was the design of box-consistency, a local consistency property implemented in `Newton` [5]. For the first time constraint solving smoothly combines Newton-like iterative methods from interval analysis and constraint propagation. These techniques have been further developed and implemented in `Numerica` [60]. Furthermore box-consistency has been shown to drastically improve hull-consistency for a large set of problems in [59].

Recent works have been interested in advanced propagation techniques [41], relaxations for specific problems [64, 38], solver cooperation [28], processing of differential equations [20], quantified formulas [54] or applications [58, 18, 24, 27, 30, 15]. The

combination of box-consistency and hull-consistency has been shown to be efficient in [4]. In the following we review the main lines of interval consistency techniques.

2.1 Constraint Satisfaction Problems

A numeric constraint satisfaction problem (NCSP) P is a triple $\langle \mathcal{C}, \mathcal{V}, \mathcal{D} \rangle$ where $\mathcal{C} = \{c_1, \ldots, c_m\}$ is a set of (in)equalities over a set $\mathcal{V} = \{x_1, \ldots, x_n\}$ of real-valued variables to which is associated a domain $\mathcal{D} = \boldsymbol{d}_1 \times \cdots \times \boldsymbol{d}_n$ called a box.

By inequalities, we mean, as usual, inequalities of the type $f = g$, $f \geq g$, or $f \leq g$, where f and g are made of real numbers, variables, arithmetic operations, and elementary functions.

Each $\boldsymbol{d}_i$ is an interval bounded by floating-point numbers that represents the set of possible values of x_i. A solution to P is an assignment of variables $(a_1, \ldots, a_n) \in \mathcal{D}$ such that all the constraints are satisfied. Let $\mathsf{Sol}\,(P)$ denote the solution set of P.

The purpose of consistency techniques is to compute a NCSP $P' = \langle \mathcal{C}, \mathcal{V}, \mathcal{D}' \rangle$ from P such that the following properties hold:

– completeness: $\mathsf{Sol}\,(P') = \mathsf{Sol}\,(P)$
– contractance: $\mathcal{D}' \subseteq \mathcal{D}$

To this end, given an integer $i \in \{1, \ldots, n\}$, define the i-th projection of a relation $\rho \subseteq \mathbb{R}^n$ as the set

$$\pi_i(\rho) = \{a_i \in \mathbb{R} \mid \exists (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n) \in \mathbb{R}^{n-1} : (a_1, \ldots, a_n) \in \rho\}.$$

An inconsistent value is an element $a_i \in \boldsymbol{d}_i$ that does not belong to the i-th projection of $\mathsf{Sol}\,(P)$. It directly follows that a_i can be removed from $\boldsymbol{d}_i$ while preserving $\mathsf{Sol}\,(P)$. In practice the main problem is to characterize the set of inconsistent values for each variable. Unfortunately, the whole set $\mathsf{Sol}\,(P)$ cannot be computed due to the limitations of machine arithmetic. As a consequence weaker techniques using constraint projections and interval enclosures have been developed.

2.2 Local Consistency Techniques

Each constraint c_j from P defines a relation ρ_j composed of all the assignments of variables $(a_1, \ldots, a_n) \in \mathcal{D}$ satisfying c_j. A constraint projection is just a projection of the associated relation. Hull-consistency is a consistency property based on the convex hull of constraint projections. The convex hull of a set of real numbers is the smallest enclosing interval.

Definition 1. *Given a constraint c_j and an integer $i \in \{1, \ldots, n\}$ $\boldsymbol{d}_i$ is hull-consistent wrt. c_j and $\mathcal{D}$ iff*

$$\boldsymbol{d}_i = \mathsf{hull}\,(\pi_i(\rho_j)).$$

The meaning of the definition is that no value is declared to be inconsistent if the hull of the i-th projection of c_j is equal to $\boldsymbol{d}_i$. In the contrary, if the consistency property is not verified, inconsistent values can be removed at bounds of $\boldsymbol{d}_i$. More precisely a

new domain is computed as $\boldsymbol{d}_i \cap \mathsf{hull}\,(\pi_i(\rho_j))$. However the computation of the hull of constraint projections cannot be achieved in general due to rounding errors of numerical computations. This problem has led to the definition of box-consistency using interval functions, *i.e.*, computable objects.

An interval form of a function $f : D_f \to \mathbb{R}$ with $D_f \subseteq \mathbb{R}^n$ is an interval function $\boldsymbol{f} : \mathbb{I}^n \to \mathbb{I}$ such that for every box $\mathcal{D} = \boldsymbol{d}_1 \times \cdots \times \boldsymbol{d}_n$ the following property holds.

$$\boldsymbol{f}(\boldsymbol{d}_1, \cdots, \boldsymbol{d}_n) \supseteq \{f(a_1, \ldots, a_n) \mid (a_1 \ldots, a_n) \in \mathcal{D} \cap D_f\}$$

In other words the evaluation of $\boldsymbol{f}$ encloses the range of f. The natural form is the syntactic extension of a given expression of f to the intervals. Interval forms can be used to implement a proof by refutation for constraint satisfaction. Given two expressions $f(x_1, \ldots, x_n)$ and $g(x_1, \ldots, x_n)$, an interval form $\boldsymbol{f}$ of f, an interval form $\boldsymbol{g}$ of g and a box $\mathcal{D}$ such that $\boldsymbol{f}(\boldsymbol{d}_1, \cdots, \boldsymbol{d}_n) = [u, v]$ and $\boldsymbol{g}(\boldsymbol{d}_1, \cdots, \boldsymbol{d}_n) = [a, b]$ the interval reasonings are as follows.

$$\begin{array}{ll} (i) & [u, v] \cap [a, b] = \varnothing \implies f = g \text{ has no solution in } \mathcal{D} \\ (ii) & v < a \implies f \geqslant g \text{ has no solution in } \mathcal{D} \end{array}$$

The constraint is proved to be violated when the interval test succeeds (i.e., when either we have an equality constraint $f = g$ and the intersection of $[u, v]$ and $[a, b]$ is empty, or we have an inequality constraint $f \leq g$ and $v < a$). This method is based on the possibly interpretation of constraint relation symbols over the intervals. Given a constraint c_j, a box $\mathcal{D}$, an integer $i \in \{1, \ldots, n\}$ and a real number $a_i \in \boldsymbol{d}_i$ let $\boldsymbol{c}_j^i(a_i, \mathcal{D})$ denote a failure of the interval test over the box $\boldsymbol{d}_1 \times \cdots \times \boldsymbol{d}_{i-1} \times \mathsf{hull}\,(\{a_i\}) \times \boldsymbol{d}_{i+1} \times \cdots \times \boldsymbol{d}_n$.

Definition 2. *Given a constraint $c_j(x_1, \ldots, x_n)$, a box $\mathcal{D}$, an interval test for c_j and an integer $i \in \{1, \ldots, n\}$ $\boldsymbol{d}_i$ is box-consistent wrt. c_j and $\mathcal{D}$ iff*

$$\boldsymbol{d}_i = \mathsf{hull}\,(\{a_i \in \boldsymbol{d}_i \mid \boldsymbol{c}_j^i(a_i, \mathcal{D})\}).$$

Each real number in $\boldsymbol{d}_i$ is characterized using the interval test, *i.e.*, the bounds of $\boldsymbol{d}_i$ cannot be declared inconsistent using the interval test. We see that the projection appearing in the definition of hull-consistency is just replaced with interval tests.

There are two kinds of algorithms implementing box-consistency: constraint inversion and dichotomous search using interval tests. Constraint inversion[4] uses inverse real operations, as illustrated in Fig. 1. The initial box $[u, v] \times [a, b]$ is reduced to $[u, \lceil \log(b) \rceil] \times [\lfloor \exp(u) \rfloor, b]$ using rounded interval arithmetic (operations $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$ are machine rounding operations). A numerical constraint inversion algorithm for processing complex constraints has been introduced in [4]. This method is efficient when variables occur once in constraints, because in this case, straightforward interval computations lead to the exact range (see, e.g., [29]); in more general situations, due to the dependency problem of interval arithmetic, this method may not always be so efficient.

The dichotomous search procedure enforces the interval test over sub-domains from $\boldsymbol{d}_i$. The aim is to compute the leftmost real number $a \in \boldsymbol{d}_i$ and the rightmost real

[4] Note that constraint inversion has often been presented as a computational technique for hull-consistency but this notion is not computable, as shown in [4].

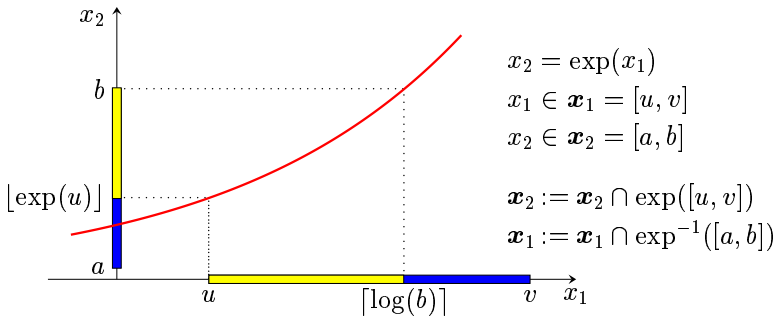

Fig. 1. Inversion of $x_2 = \exp(x_1)$ for computing domain reductions.

number $b \in \boldsymbol{d}_i$ such that the interval tests $\boldsymbol{c}_j^i(a, \mathcal{D})$ and $\boldsymbol{c}_j^i(b, \mathcal{D})$ fail (the constraint is not violated). Doing so the new domain is $[a, b]$. The method is in three steps: if the whole domain $\boldsymbol{d}_i$ is rejected using the interval test then stop and return the empty set; otherwise try to reduce the left bound of $\boldsymbol{d}_i$ and then the right bound. The search of outermost points is just implemented by a dichotomous search using the interval tests.

Example 1. Consider the constraint $(x_1 - 1)^2 = 0$, the interval $\boldsymbol{d}_1 = [-10, 10]$ and the interval test based on natural forms. Domain of x_1 is not box-consistent since the interval test succeeds for the right bound ($(10 - 1)^2 \neq 0$). As a consequence $\boldsymbol{d}_1$ can be reduced using *e.g.* constraint inversion, as follows.

$$\boldsymbol{d}_1 := \boldsymbol{d}_1 \cap \left(square^{-1}(0) + 1 \right) = [1, 1]$$

Now suppose that the constraint is given in a developed form $x_1^2 - 2x_1 + 1 = 0$. The problem of constraint inversion is to choose one occurrence of x_1. Suppose this is the one in the quadratic term. Then try to reduce $\boldsymbol{d}_1$.

$$\boldsymbol{d}_1 := \boldsymbol{d}_1 \cap square^{-1}(2\boldsymbol{d}_1 - 1) \approx [-\sqrt{19}, \sqrt{19}]$$

The approximation is weak, which is due to the dependency problem of interval arithmetic (two occurrences of x_1 considered as different variables during interval computations). A solution is to implement a dichotomous search. In this case a tight interval enclosing 1 is computed since each small sub-domain can be rejected using the interval test. For instance the real number $2 \in \boldsymbol{d}_1$ can be rejected since $(2^2 - 2 \times 2 + 1 \neq 0)$. Using the search procedure the dependency problem over the considered variable vanishes. However, in this case, a second problem happens, a slow convergence, since there is a multiple root of function $(x_1 - 1)^2$. This phenomenon has to be controlled and the search stopped if necessary. The reader is referred to [26] for more details.

2.3 Constraint Propagation and Strong Consistency Techniques

Given a NCSP constraint projections have to be processed in sequence in order to reach the consistency of the whole problem. The fixed-point algorithm implementing such a sequence of computations is called constraint propagation. Given a constraint

$c(x_1, \ldots, x_n)$ and an integer $i \in \{1, \ldots, n\}$ let revise_{ij} denote the function from boxes to boxes such that for each box $\mathcal{D}$

$$\mathsf{revise}_{ij}(\mathcal{D}) = \mathcal{D}' = \boldsymbol{d}_1 \times \cdots \times \boldsymbol{d}_{i-1} \times \boldsymbol{d}'_i \times \boldsymbol{d}_{i+1} \times \cdots \times \boldsymbol{d}_n$$

and $\boldsymbol{d}'_i$ is the largest interval included in $\boldsymbol{d}_i$ that is box-consistent wrt. c_j and $\mathcal{D}'$. The revise function is implemented by constraint inversion or dichotomous search. A generic constraint propagation algorithm implementing box-consistency over NCSPs is presented in Table 1. Each step consists in an application of a revise function for reducing the current box. The loop invariant states that at the beginning of each step of the loop, each function θ not in $\mathcal{S}$ is such that $\theta(\mathcal{D}) = \mathcal{D}$. Then every revise function associated to a constraint c_q that contains a variable x_i which domain has been modified is added in $\mathcal{S}$.

It is worth mentioning that the algorithm presented in Table 1 is actually intended for a more general situation that the situation that we have described. In our particular case, according to the definition of revise_{ij}, the function revise_{ij} changes only the i-th component $\boldsymbol{d}_i$ to $\boldsymbol{d}'_i$. Therefore, there is only value k which should be considered in the line foreach $k \in \{1, \ldots, n\}$ such that $\boldsymbol{d}'_k \neq \boldsymbol{d}_k$ do – the value $k = i$. In a more general situation, however, we do need a foreach-loop.

Table 1. Constraint propagation algorithm for box-consistency.

```
propagate (C = {c_1, ..., c_m}, V = {x_1, ..., x_n}, D = d_1 × ··· × d_n): box
begin
    S := {revise_ij | i ∈ {1, ..., n}, j ∈ {1, ..., m}}
    repeat
        revise_ij := pop (S)
        D' := revise_ij(D)
        foreach k ∈ {1, ..., n} such that d'_k ≠ d_k do
            S := push ({revise_pq | 1 ⩽ p ⩽ n, 1 ⩽ q ⩽ m, x_i occurs in c_q})
        endfor
        D := D'
    until S = ∅ or D = ∅
    return D
end
```

The propagate algorithm terminates in finite time since every step is contracting and the computational domain is finite. It is complete, *i.e.*, no solution is lost, since every revise function is complete. It converges and computes the greatest common fixed-point of the revise functions. As a consequence the order of applications of the revise functions is not relevant. The proofs of these properties come from domain theory [2].

Example 2. Consider the NCSP $\langle \{x_2 = x_1^2, x_1^2 + x_2^2 = 2\}, \{x_1, x_2\}, [-10, 10]^2 \rangle$ which solutions are $(1, 1)$ and $(-1, 1)$. Applying revise_{21} reduces the domain of x_2 to $[0, 10]$. Applying revise_{12} reduces the domain of x_1 to $[-\sqrt{2}, \sqrt{2}]$, and so on. The result of

propagate is the box $[-1.19, 1.19] \times [0.76, 1.42]$. Unfortunately this process does not compute the enclosure of the solution set, namely $[-1, 1] \times [1, 1]$. This weakness is due to the locality problem.

The locality problem originates from the way to reduce domains, since each constraint projection is used independently. However we may say that "the intersection of projections is weaker than the projection of the intersection". In Fig. 2 the NCSP represents an intersection of two curves c_1 and c_2. In this case the box cannot be reliably reduced using c_1 because this would loose solutions of c_1 (idem for c_2). This problem has led to the definition of stronger consistency techniques, namely kB-consistencies [40]. The main idea is shown in Fig. 2: prove the inconsistency of a sub-box using the constraint propagation algorithm. In this case the leftmost sub-box is discarded and the rightmost sub-box is reduced. If this process is iterated then a tight enclosure of the solution can be computed. Unfortunately it has been shown in [52] that strong consistency techniques do not have a good practical complexity. A formal comparison of consistency techniques can be found in [17].

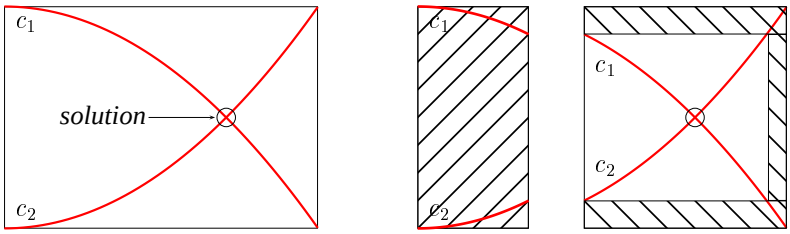

Fig. 2. Locality problem and strong consistency technique.

2.4 Use of Consistency Techniques

Mathematical models are often heterogeneous, involving integer or real numbers, differential equations, inequalities, objectives or quantified constraints. Actually consistency techniques can be implemented as components in more general solving processes, *e.g.*, constrained optimization, ODE solving or decomposition of quantified inequalities. They are used to reduce domains of possible values of the unknowns or to prove the inconsistency of a given problem.

As an example we describe the processing of universally quantified inequalities [3]. Consider the formula $\phi : \forall y \in \boldsymbol{y}\ f(y, x_1, \ldots, x_n) \leqslant 0$ over the box $\mathcal{D}$. A solution to ϕ is a tuple $(a_1, \ldots, a_n) \in \mathcal{D}$ such that for all $b \in \boldsymbol{y}$, $f(b, a_1, \ldots, a_n) \leqslant 0$ holds. Now suppose there exists a box $\boldsymbol{y} \times \mathcal{D}'$ such that every of its elements violates $f(y, x_1, \ldots, x_n) > 0$. As a consequence every element of $\mathcal{D}'$ is a solution to ϕ, since $f > 0$ is the negation of $f \leqslant 0$. Consistency techniques are used to compute such a box: (1) starting from the box $\boldsymbol{y} \times \mathcal{D}$ apply a consistency technique over the negation of the constraint; (2) for each eliminated box $\boldsymbol{y}' \times \mathcal{D}'$ return $\mathcal{D}'$ if $\boldsymbol{y}' = \boldsymbol{y}$. In practice strict inequalities resulting from negations are relaxed. Note that a box that is inconsistent

wrt. $f \geqslant 0$ is clearly inconsistent wrt. $f > 0$. In [54] a general framework for solving quantified constraints using consistency techniques has been proposed.

2.5 Perspectives

Consistency techniques have been developed for solving NCSPs, namely conjunctions of (in)equalities. Only recently they have been extended to tackle quantifiers, conditional constraints, mixed problems, differential equations or optimization problems. We believe that the main perspective is to develop a suite of tools to be used in real-world applications and to integrate these tools in development frameworks. The main advantages that should be highlighted are that the use of intervals lead to robust decisions and that heterogeneous systems are handled without additional work.

A particularly interesting field of applications is the so-called robust design. Solutions in automatic control are proposed in [30]. For instance, state estimation problems are efficiently solved by constraint propagation alone since many redundant constraints, provided by redundant captors, are available. On the contrary problems from image synthesis [15] are often under-constrained. Consistency techniques have to be combined with local search and optimization methods. In conceptual design [51, 24] the aim is to derive all possible concepts from specifications, and possibly to take a decision. Consistency techniques embedding decomposition techniques of NCSPs [8] may be directly applied. However further research has to be done in several ways such as uncertainty quantification given approximate models and sensitivity analysis.

3 Interval-Probability Techniques

What we are planning to do in this section. As we have mentioned in the Introduction, in many practical situations, in addition to the interval information, we also have some information about the probabilities.

In this section, first, we analyze a specific interval computations problem – when we use traditional statistical data processing algorithms $f(x_1, \ldots, x_n)$ to process the results of direct measurements.

Then, we extend our analysis to the case when for each input x_i, in addition to the interval $\mathbf{x}_i = [\underline{x}_i, \overline{x}_i]$ of possible values, we have partial information about the probabilities: specifically, we know its mean E_i (or an interval $\mathbf{E}_i$ of possible values of the mean).

3.1 First Step Beyond Intervals: Error Estimation for Traditional Statistical Data Processing Algorithms under Interval Uncertainty

When we have n results $x_1, \ldots, x_n$ of repeated measurement of the same quantity (at different points, or at different moments of time), traditional statistical approach usually starts with computing their sample average $E = (x_1 + \ldots + x_n)/n$ and their (sample) variance

$$V = \frac{(x_1 - E)^2 + \ldots + (x_n - E)^2}{n} \tag{1}$$

(or, equivalently, the sample standard deviation $\sigma = \sqrt{V}$); see, e.g., [53].

In this section, we consider situations when we do not know the exact values of the quantities $x_1, \ldots, x_n$, we only know the intervals $\mathbf{x}_1, \ldots, \mathbf{x}_n$ of possible values of x_i. In such situations, for different possible values $x_i \in \mathbf{x}_i$, we get different values of E and V. The question is: what are the intervals $\mathbf{E}$ and $\mathbf{V}$ of possible values of E and V?

The practical importance of this question was emphasized, e.g., in [48, 49] on the example of processing geophysical data.

For E, the straightforward interval computations [30–32, 44] leads to the exact range:

$$\mathbf{E} = \frac{\mathbf{x}_1 + \ldots + \mathbf{x}_n}{n}, \text{ i.e., } \underline{E} = \frac{\underline{x}_1 + \ldots + \underline{x}_n}{n}, \text{ and } \overline{E} = \frac{\overline{x}_1 + \ldots + \overline{x}_n}{n}.$$

For V, straightforward interval computations lead to an excess width. For example, for $\mathbf{x}_1 = \mathbf{x}_2 = [0, 1]$, the variance is $V = (x_1 - x_2)^2/4$ and hence, the actual range $\mathbf{V} = [0, 0.25]$. On the other hand, $\mathbf{E} = [0, 1]$, hence

$$\frac{(\mathbf{x}_1 - \mathbf{E})^2 + (\mathbf{x}_2 - \mathbf{E})^2}{2} = [0, 1] \supset [0, 0.25].$$

More sophisticated methods of interval computations also sometimes lead to an excess width. Reason: in the formula for the average E, each variable only occurs once, and it is known that for such formulas, straightforward interval computations lead to the exact range (see, e.g., [29]). In the expression for variance, each variable x_i occurs several times: explicitly, in $(x_i - E)^2$, and explicitly, in the expression for E. In such cases, often, dependence between intermediate computation results leads to excess width of the results of straightforward interval computations. Not surprisingly, we do get excess width when applying straightforward interval computations to the formula (1).

For variance, we can actually prove that the corresponding optimization problem is difficult:

Theorem 1. [23] *Computing $\overline{V}$ is NP-hard.*

The very fact that computing the range of a quadratic function is NP-hard was first proven by Vavasis [61] (see also [35]). We have shown that this difficulty happens even for very simple quadratic functions frequently used in data processing.

A natural question is: maybe the difficulty comes from the requirement that the range be computed exactly? In practice, it is often sufficient to compute, in a reasonable amount of time, a usefully accurate estimate $\widetilde{\overline{V}}$ for $\overline{V}$, i.e., an estimate $\widetilde{\overline{V}}$ which is accurate with a given accuracy $\varepsilon > 0$: $\left|\widetilde{\overline{V}} - \overline{V}\right| \leq \varepsilon$. Alas, for any ε, such computations are also NP-hard:

Theorem 2. [23] *For every $\varepsilon > 0$, the problem of computing $\overline{V}$ with accuracy ε is NP-hard.*

It is worth mentioning that $\overline{V}$ can be computed exactly in exponential time $O(2^n)$:

Theorem 3. [23] *There exists an algorithm that computes $\overline{V}$ in exponential time.*

This algorithm consists of testing all 2^n possible combinations of values $x_i = \underline{x}_i$ and $x_i = \overline{x}_i$. For small number of variables, this is quite doable, but when we have, say, $n = 10^3$ observations, we cannot do that anymore. In the following text, we will describe what we can do in such cases.

For computing $\underline{V}$, there exists feasible algorithms. In [23], we described a quadratic-time algorithm for computing $\underline{V}$. It turns out that we can actually compute $\underline{V}$ in $O(n \cdot \log(n))$ computational steps (arithmetic operations or comparisons) for n interval data points $\mathbf{x}_i = [\underline{x}_i, \overline{x}_i]$.

The algorithm $\underline{\mathcal{A}}$ is as follows:

- First, we sort all $2n$ values $\underline{x}_i, \overline{x}_i$ into a sequence $x_{(1)} \le x_{(2)} \le \ldots \le x_{(2n)}$.
- Second, we use bisection to find the value k ($1 \le k \le 2n$) for which the following two inequalities hold:

$$\sum_{j:\overline{x}_j \le x_{(k)}} (x_{(k)} - \overline{x}_j) \le \sum_{i:\underline{x}_i \ge x_{(k+1)}} (\underline{x}_i - x_{(k)}); \tag{2}$$

$$\sum_{j:\overline{x}_j \le x_{(k)}} (x_{(k+1)} - \overline{x}_j) \ge \sum_{i:\underline{x}_i \ge x_{(k+1)}} (\underline{x}_i - x_{(k+1)}). \tag{3}$$

  At each iteration of this bisection, we have an interval $[k^-, k^+]$ that is guaranteed to contain k. In the beginning, $k^- = 1$ and $k^+ = 2n$. At each stage, we compute the midpoint $k_{\mathrm{mid}} = \lfloor (k^- + k^+)/2 \rfloor$, and check both inequalities (2) and (3) for $k = k_{\mathrm{mid}}$. Then:
  - If both inequalities (2) and (3) hold for his k, this means that we have found the desired k.
  - If (2) holds but (3) does not hold, this means that the desired value k is larger than k_{mid}, so we keep k^+ and replace k^- with $k_{\mathrm{mid}} + 1$.
  - If (3) holds but (2) does not hold, this means that the desired value k is smaller than k_{mid}, so we keep k^- and replace k^+ with $k_{\mathrm{mid}} - 1$.
- Once k is found, we compute

$$S_k \stackrel{\mathrm{def}}{=} \sum_{i:\underline{x}_i \ge x_{(k+1)}} \underline{x}_i + \sum_{j:\overline{x}_j \le x_{(k)}} \overline{x}_j, \tag{4}$$

  then $r_k = S_k / N_k$, where N_k is the total number of such is and js, and, finally,

$$\underline{V} = \frac{1}{n} \cdot \left(\sum_{j:\overline{x}_j \le x_{(k)}} (\overline{x}_j - r_k)^2 + \sum_{i:\underline{x}_i \ge x_{(k+1)}} (\underline{x}_i - r_k)^2 \right).$$

Theorem 4. (Kreinovich, Longpré) *The algorithm $\underline{\mathcal{A}}$ always computes $\underline{V}$ in time $O(n \cdot \log(n))$.*

Proof. Let us start with simple calculus. Let $f(x_1, \ldots, x_n)$ be a differentiable function on a box $B \stackrel{\text{def}}{=} \mathbf{x}_1 \times \ldots \times \mathbf{x}_n$, and let $x^- = (x_1^-, \ldots, x_n^-) \in B$ be a point at which f attains its smallest value on this box.

Then, for every i, the function $f_i(x_i) \stackrel{\text{def}}{=} f(x_1^-, \ldots, x_{i-1}^-, x_i, x_{i+1}^-, \ldots, x_n^-)$ also attains its minimum on the interval $[\underline{x}_i, \overline{x}_i]$ at the point $x_i = x_i^-$.

According to the basic calculus, this minimum is either attained in the interior of the interval, in which case $df_i/dx_i = 0$ for $x_i = x_i^-$, or the minimum is attained at one of the endpoints of the interval $[\underline{x}_i, \overline{x}_i]$. If the minimum is attained at the left endpoint $\underline{x}_i$, then the function f_i cannot be decreasing at this point, so $df_i/dx_i \geq 0$. Similarly, if the minimum is attained at the right endpoint $\overline{x}_i$, then $df_i/dx_i \leq 0$.

By definition of the function $f_i(x_i)$, the value of the derivative df_i/dx_i for $x_i = x_i^-$ is equal to the value of the partial derivative $\partial f/\partial x_i$ at the point x^-. Thus, for each i, we have one of the following three cases:

- either $\underline{x}_i < x_i^- < \overline{x}_i$ and $\partial f/\partial x_i = 0$;
- or $x_i^- = \underline{x}_i$ and $\partial f/\partial x_i \geq 0$;
- or $x_i^- = \overline{x}_i$ and $\partial f/\partial x_i \leq 0$.

For $f = V$, as one can easily see, $\partial V/\partial x_i = (2/n) \cdot (x_i - E)$, so the sign of this derivative is the same as the sing of the difference $x_i - E$. Therefore, for the point x^- at which the variance V attains its minimum, we have one of the following three situations:

- either $\underline{x}_i < x_i^- < \overline{x}_i$ and $x_i^- = E$;
- or $x_i^- = \underline{x}_i$ and $x_i^- \geq E$;
- or $x_i^- = \overline{x}_i$ and $x_i^- \leq E$.

In the first case, $\underline{x}_i < E < \overline{x}_i$; in the second case, $E \leq \underline{x}_i$, and in the third case, $\overline{x}_i \leq E$.

Let us show that if we know where E is in comparison to the endpoints of all the intervals, i.e., to which "small interval" $[x_{(k)}, x_{(k+1)}]$ the value E belongs, we can uniquely determine the values x_i^- for all i.

Indeed, when $x_{(k+1)} \leq \underline{x}_i$, this means that $E \leq x_{(k+1)} \leq \underline{x}_i$, so $E \leq \underline{x}_i$. Thus, we cannot have the first case (in which $E > \underline{x}_i$), so we must have either the second or the third cases, i.e., we must have $x_i = \underline{x}_i$ or $x_i = \overline{x}_i$. If the interval $\mathbf{x}_i$ is degenerate, then both cases lead to the same result. If the interval is non-degenerate, then we cannot have the third case – in which $\underline{x}_i < \overline{x}_i \leq E$ hence $\underline{x}_i < E$ – and thus, we must have the second case, i.e., $x_i^- = \overline{x}_i$. Thus, $x_{(k+1)} \leq \underline{x}_i$ implies that $x_i^- = \underline{x}_i$.

Similarly, $x_{(k)} \geq \overline{x}_i$ implies that $x_i^- = \overline{x}_i$, and in all other cases, we have $x_i^- = E$.

All that remains is to find the appropriate k. Once k is fixed, we can find the values x_i^- in linear time, and then compute the corresponding value V is linear time. The only condition on k is that the average of the corresponding values x_i^- should be within the corresponding small interval $[x_{(k)}, x_{(k+1)}]$.

In [23], we proposed to find k by exhaustive (linear) search. Since there are $2n$ possible small intervals, we must therefore repeat $O(n)$ computations $2n$ times, which

takes $2n \cdot O(n) = O(n^2)$ time. Together with the original sorting – that takes $O(n \cdot \log(n))$ time – we thus get a quadratic time algorithm, since

$$O(n^2) + O(n \cdot \log(n)) = O(n^2).$$

Let us now show that we can find k faster. We want to satisfy the conditions $x^{(k)} \le E$ and $E \le x^{(k+1)}$. The value E is the average of all the values x_i^-, i.e., we have

$$n \cdot E = S_k + (n - N_k) \cdot E, \tag{5}$$

where S_k is defined by the formula (4) and N_k is defined in the description of the algorithm $\underline{\mathcal{A}}$. By moving all the terms proportional to E to the left-hand side of (5), we conclude that $N_k \cdot E = S_k$, i.e., that $E = S_k/N_k$ $(= r_k)$. The first desired inequality $x^{(k)} \le E$ thus takes the form $S_k/N_k \le x^{(k)}$, i.e., equivalently, $N_k \cdot x^{(k)} \le S_k$, i.e.,

$$N_k \cdot x_{(k)} \le \sum_{i:\underline{x}_i \ge x_{(k+1)}} \underline{x}_i + \sum_{j:\overline{x}_j \le x_{(k)}} \overline{x}_j. \tag{6}$$

In the left-hand side of (6), we have as many terms $x_{(k)}$ as there are terms in the right-hand side. Thus, we can subtract $x_{(k)}$ from each of N_k term in the right-hand side. When $\overline{x}_j \le x_{(k)}$, the difference $\overline{x}_j - x_{(k)}$ is negative, so we can move it to the left-hand side of the inequality, resulting in the inequality (2).

When k increases, the left-hand side of the inequality (2) increases – because each term increases and new terms may appear. Similarly, the right-hand side of this inequality decreases with k. Thus, if this inequality holds for k, it should also hold for all smaller values, i.e., for $k-1$, $k-2$, etc.

Similarly, the second desired inequality $E \le x^{(k+1)}$ takes the equivalent form (3). When k increases, the left-hand side of this inequality increases, while the right-hand side decreases. Thus, if this inequality is true for k, it is also true for $k+1$, $k+2$, ...

If both inequalities (2) and (3) are true for two different values $k < k'$, then they should both be true for all the values intermediate between k and k', i.e., for $k+1, k+2, \ldots, k'-1$. Let us show that both inequalities cannot be true for k and for $k+1$. Indeed, if the inequality (2) is true for $k+1$, this means that

$$\sum_{j:\overline{x}_j \le x_{(k+1)}} (x_{(k+1)} - \overline{x}_j) \le \sum_{i:\underline{x}_i \ge x_{(k+2)}} (\underline{x}_i - x_{(k+1)}). \tag{7}$$

However, the left-hand side of this inequality is not smaller than the left-hand side of (3), while the right-hand side of this inequality is not larger than the right-hand side of (3). Thus, (7) is inconsistent with (3). This inconsistency proves that there is only one k for which both inequalities are true, and this k can be found by the bisection method as described in the above algorithm $\underline{\mathcal{A}}$.

How long does this algorithm take? In the beginning, we only know that k belongs to the interval $[1, 2n]$ of width $O(n)$. At each stage of the bisection step, we divide the interval (containing k) in half. After I iterations, we decrease the width of this interval by a factor of 2^I. Thus, to find the exact value of k, we must have I for which

$O(n)/2^I = 1$, i.e., we need $I = O(\log(n))$ iterations. On each iteration, we need $O(n)$ steps, so we need a total of $O(n \cdot \log(n))$ steps. With $O(n \cdot \log(n))$ steps for sorting, and $O(n)$ for computing the variance, we get a $O(n \cdot \log(n))$ algorithm. The theorem is proven.

NP-hardness of computing $\overline{V}$ means, crudely speaking, that there are no general ways for solving all particular cases of this problem (i.e., computing $\overline{V}$) in reasonable time.

However, there are algorithms for computing $\overline{V}$ for many reasonable situations. In [23], we proposed an efficient algorithm that computes $\overline{V}$ for the case when all the interval midpoints ("measured values") $\widetilde{x}_i = (\underline{x}_i + \overline{x}_i)/2$ are definitely different from each other, in the sense that the "narrowed" intervals $[\widetilde{x}_i - \Delta_i/n, \widetilde{x}_i + \Delta_i/n]$ – where $\Delta_i = (\underline{x}_i - \overline{x}_i)/2$ is the interval's half-width – do not intersect with each other.

This algorithm $\overline{\mathcal{A}}$ is as follows:

- First, we sort all $2n$ endpoints of the narrowed intervals $\widetilde{x}_i - \Delta_i/n$ and $\widetilde{x}_i + \Delta_i/n$ into a sequence $x_{(1)} \le x_{(2)} \le \ldots \le x_{(2n)}$. This enables us to divide the real line into $2n+1$ segments ("small intervals") $[x_{(k)}, x_{(k+1)}]$, where we denoted $x_{(0)} \stackrel{\text{def}}{=} -\infty$ and $x_{(2n+1)} \stackrel{\text{def}}{=} +\infty$.
- Second, we compute $\underline{E}$ and $\overline{E}$ and pick all "small intervals" $[x_{(k)}, x_{(k+1)}]$ that intersect with $[\underline{E}, \overline{E}]$.
- For each of remaining small intervals $[x_{(k)}, x_{(k+1)}]$, for each i from 1 to n, we pick the following value of x_i:
  - if $x_{(k+1)} < \widetilde{x}_i - \Delta_i/n$, then we pick $x_i = \overline{x}_i$;
  - if $x_{(k)} > \widetilde{x}_i + \Delta_i/n$, then we pick $x_i = \underline{x}_i$;
  - for all other i, we consider both possible values $x_i = \overline{x}_i$ and $x_i = \underline{x}_i$.

  As a result, we get one or several sequences of x_i. For each of these sequences, we check whether the average E of the selected values $x_1, \ldots, x_n$ is indeed within this small interval, and if it is, compute the variance by using the formula (1).
- Finally, we return the largest of the computed variances as $\overline{V}$.

Theorem 5. *The algorithm $\overline{\mathcal{A}}$ computes $\overline{V}$ in quadratic time for all the cases in which the "narrowed" intervals do not intersect with each other.*

This algorithm also works when, for some fixed k, no more than k "narrowed" intervals can have a common point:

Theorem 6. *For every positive integer k, the algorithm $\overline{\mathcal{A}}$ computes $\overline{V}$ in quadratic time for all the cases in which no more than k "narrowed" intervals can have a common point.*

Can we compute $\overline{V}$ faster? In [23], we provided quadratic-time algorithms for computing $\underline{V}$ and $\overline{V}$. For $\underline{V}$, we have shown, in this paper, that we can replace the exhaustive (linear) search over k with an appropriate binary search and thus reduce the computation time from $O(n^2)$ to $O(n \cdot \log(n))$. It turns out [65] that a similar modification can be done for the algorithm $\overline{V}$. Thus (provided that no more than k narrowed intervals have a common point), we can compute $\overline{V}$ in time $O(n \cdot \log(n))$ as well.

3.2 Second Step Beyond Intervals: Extension of Interval Arithmetic to Situations with Partial Information about Probabilities

Practical problem. In some practical situations, in addition to the lower and upper bounds on each random variable x_i, we know the bounds $\mathbf{E}_i = [\underline{E}_i, \overline{E}_i]$ on its mean E_i.

Indeed, in measurement practice (see, e.g., [53]), the overall measurement error Δx is usually represented as a sum of two components:

- a *systematic* error component $\Delta_s x$ which is defined as the expected value $E[\Delta x]$, and
- a *random* error component $\Delta_r x$ which is defined as the difference between the overall measurement error and the systematic error component: $\Delta_r x \stackrel{\text{def}}{=} \Delta x - \Delta_s x$.

In addition to the bound Δ on the overall measurement error, the manufacturers of the measuring instrument often provide an upper bound Δ_s on the systematic error component: $|\Delta_s x| \le \Delta_s$.

This additional information is provided because, with this additional information, we not only get a bound on the accuracy of a single measurement, but we also get an idea of what accuracy we can attain if we use repeated measurements to increase the measurement accuracy. Indeed, the very idea that repeated measurements can improve the measurement accuracy is natural: we measure the same quantity by using the same measurement instrument several (N) times, and then take, e.g., an arithmetic average $\bar{x} = (\widetilde{x}^{(1)} + \ldots + \widetilde{x}^{(N)})/N$ of the corresponding measurement results $\widetilde{x}^{(1)} = x + \Delta x^{(1)}$, $\ldots$, $\widetilde{x}^{(N)} = x + \Delta x^{(N)}$.

- If systematic error is the only error component, then all the measurements lead to exactly the same value $\widetilde{x}^{(1)} = \ldots = \widetilde{x}^{(N)}$, and averaging does not change the value – hence does not improve the accuracy.
- On the other hand, if we know that the systematic error component is 0, i.e., $E[\Delta x] = 0$ and $E[\widetilde{x}] = x$, then, as $N \to \infty$, the arithmetic average tends to the actual value x. In this case, by repeating the measurements sufficiently many times, we can determine the actual value of x with an arbitrary given accuracy.

In general, by repeating measurements sufficiently many times, we can arbitrarily decrease the random error component and thus attain accuracy as close to Δ_s as we want.

When this additional information is given, then, after we performed a measurement and got a measurement result $\widetilde{x}$, then not only we get the information that the actual value x of the measured quantity belongs to the interval $\mathbf{x} = [\widetilde{x} - \Delta, \widetilde{x} + \Delta]$, but we can also conclude that the expected value of $x = \widetilde{x} - \Delta x$ (which is equal to $E[x] = \widetilde{x} - E[\Delta x] = \widetilde{x} - \Delta_s x$) belongs to the interval $\mathbf{E} = [\widetilde{x} - \Delta_s, \widetilde{x} + \Delta_s]$.

If we have this information for every x_i, then, in addition to the interval $\mathbf{y}$ of possible value of y, we would also like to know the interval of possible values of $E[y]$. This additional interval will hopefully provide us with the information on how repeated measurements can improve the accuracy of this indirect measurement. Thus, we arrive at the following problem.

Resulting optimization problem. In more optimization terms, we want to solve the following problem: given an algorithm computing a function $f(x_1, \ldots, x_n)$ from R^n to R; and values $\underline{x}_1, \overline{x}_1, \ldots, \underline{x}_n, \overline{x}_n, \underline{E}_1, \overline{E}_1, \ldots, \underline{E}_n, \overline{E}_n$, we want to find

$$\underline{E} \stackrel{\rm def}{=} \min\{E[f(x_1, \ldots, x_n)] \mid \text{ all distributions of } (x_1, \ldots, x_n) \text{ for which}$$

$$x_1 \in [\underline{x}_1, \overline{x}_1], \ldots, x_n \in [\underline{x}_n, \overline{x}_n], E[x_1] \in [\underline{E}_1, \overline{E}_1], \ldots E[x_n] \in [\underline{E}_n, \overline{E}_n]\};$$

and $\overline{E}$ which is the maximum of $E[f(x_1, \ldots, x_n)]$ for all such distributions.

In addition to considering all possible distributions, we can also consider the case when all the variables x_i are independent.

Analog of straightforward interval computations. The main idea behind straightforward interval computations can be applied here as well. Namely, first, we find out how to solve this problem for the case when $n = 2$ and $f(x_1, x_2)$ is one of the standard arithmetic operations. Then, once we have an arbitrary algorithm $f(x_1, \ldots, x_n)$, we parse it and replace each elementary operation on real numbers with the corresponding operation on quadruples $(\underline{x}, \underline{E}, \overline{E}, \overline{x})$.

To implement this idea, we must therefore know how to solve the above problem for elementary operations.

For *addition*, the answer is simple. Since $E[x_1 + x_2] = E[x_1] + E[x_2]$, if $y = x_1 + x_2$, there is only one possible value for $E = E[y]$: the value $E = E_1 + E_2$. This value does not depend on whether we have correlation or nor, and whether we have any information about the correlation. Thus, $\mathbf{E} = \mathbf{E}_1 + \mathbf{E}_2$.

Similarly, the answer is simple for *subtraction:* if $y = x_1 - x_2$, there is only one possible value for $E = E[y]$: the value $E = E_1 - E_2$. Thus, $\mathbf{E} = \mathbf{E}_1 - \mathbf{E}_2$.

For *multiplication*, if the variables x_1 and x_2 are independent, then $E[x_1 \cdot x_2] = E[x_1] \cdot E[x_2]$. Hence, if $y = x_1 \cdot x_2$ and x_1 and x_2 are independent, there is only one possible value for $E = E[y]$: the value $E = E_1 \cdot E_2$; hence $\mathbf{E} = \mathbf{E}_1 \cdot \mathbf{E}_2$.

The first non-trivial case is the case of multiplication in the presence of possible correlation. When we know the exact values of E_1 and E_2, the solution to the above problem is as follows (see, e.g., [34]:

Theorem 7. *For multiplication $y = x_1 \cdot x_2$, when we have no information about the correlation,*

$$\underline{E} = \max(p_1 + p_2 - 1, 0) \cdot \overline{x}_1 \cdot \overline{x}_2 + \min(p_1, 1 - p_2) \cdot \overline{x}_1 \cdot \underline{x}_2 + \min(1 - p_1, p_2) \cdot \underline{x}_1 \cdot \overline{x}_2 +$$

$$\max(1 - p_1 - p_2, 0) \cdot \underline{x}_1 \cdot \underline{x}_2;$$

and

$$\overline{E} = \min(p_1, p_2) \cdot \overline{x}_1 \cdot \overline{x}_2 + \max(p_1 - p_2, 0) \cdot \overline{x}_1 \cdot \underline{x}_2 + \max(p_2 - p_1, 0) \cdot \underline{x}_1 \cdot \overline{x}_2 +$$

$$\min(1 - p_1, 1 - p_2) \cdot \underline{x}_1 \cdot \underline{x}_2,$$

where $p_i \stackrel{\rm def}{=} (E_i - \underline{x}_i)/(\overline{x}_i - \underline{x}_i)$.

When we only know the intervals $\mathbf{E}_i$ of possible values of E_i, instead of the values p_i, we have the corresponding intervals $\mathbf{p}_i = (\mathbf{E}_i - \underline{x}_i)/(\overline{E}_i - \underline{x}_i)$. In terms of these intervals, we get the following results:

Theorem 8. *For multiplication under no information about dependence, to find $\underline{E}$, it is sufficient to consider the following combinations of p_1 and p_2:*

- $p_1 = \underline{p}_1$ *and* $p_2 = \underline{p}_2$*;* $p_1 = \underline{p}_1$ *and* $p_2 = \overline{p}_2$*;* $p_1 = \overline{p}_1$ *and* $p_2 = \underline{p}_2$*;* $p_1 = \overline{p}_1$ *and* $p_2 = \overline{p}_2$*;*
- $p_1 = \max(\underline{p}_1, 1 - \overline{p}_2)$ *and* $p_2 = 1 - p_1$ *(if* $1 \in \mathbf{p}_1 + \mathbf{p}_2$*); and*
- $p_1 = \min(\overline{p}_1, 1 - \underline{p}_2)$ *and* $p_2 = 1 - p_1$ *(if* $1 \in \mathbf{p}_1 + \mathbf{p}_2$*).*

The smallest value of $\underline{E}$ for all these cases is the desired lower bound $\underline{E}$.

Theorem 9. *For multiplication under no information about dependence, to find $\overline{E}$, it is sufficient to consider the following combinations of p_1 and p_2:*

- $p_1 = \underline{p}_1$ *and* $p_2 = \underline{p}_2$*;* $p_1 = \underline{p}_1$ *and* $p_2 = \overline{p}_2$*;* $p_1 = \overline{p}_1$ *and* $p_2 = \underline{p}_2$*;* $p_1 = \overline{p}_1$ *and* $p_2 = \overline{p}_2$*;*
- $p_1 = p_2 = \max(\underline{p}_1, \underline{p}_2)$ *(if* $\mathbf{p}_1 \cap \mathbf{p}_2 \neq \varnothing$*); and*
- $p_1 = p_2 = \min(\overline{p}_1, \overline{p}_2)$ *(if* $\mathbf{p}_1 \cap \mathbf{p}_2 \neq \varnothing$*).*

The largest value of $\overline{E}$ for all these cases is the desired upper bound $\overline{E}$.

- For the *inverse* $y = 1/x_1$, bounds for E can be deduced from convexity [57]: $\mathbf{E} = [1/E_1, p_1/\overline{x}_1 + (1 - p_1)/\underline{x}_1]$.
- For min and independent x_i, we have $\overline{E} = \min(E_1, E_2)$ and

$$\underline{E} = p_1 \cdot p_2 \cdot \min(\overline{x}_1, \overline{x}_2) + p_1 \cdot (1 - p_2) \cdot \min(\overline{x}_1, \underline{x}_2) + (1 - p_1) \cdot p_2 \cdot \min(\underline{x}_1, \overline{x}_2) +$$
$$(1 - p_1) \cdot (1 - p_2) \cdot \min(\underline{x}_1, \underline{x}_2).$$

- For max and independent x_i, we have $\underline{E} = \max(E_1, E_2)$ and

$$\overline{E} = p_1 \cdot p_2 \cdot \max(\overline{x}_1, \overline{x}_2) + p_1 \cdot (1 - p_2) \cdot \max(\overline{x}_1, \underline{x}_2) + (1 - p_1) \cdot p_2 \cdot \max(\underline{x}_1, \overline{x}_2) +$$
$$(1 - p_1) \cdot (1 - p_2) \cdot \max(\underline{x}_1, \underline{x}_2).$$

- For min in the general case, $\overline{E} = \min(E_1, E_2)$,

$$\underline{E} = \max(p_1 + p_2 - 1, 0) \cdot \min(\overline{x}_1, \overline{x}_2) + \min(p_1, 1 - p_2) \cdot \min(\overline{x}_1, \underline{x}_2) +$$
$$\min(1 - p_1, p_2) \cdot \min(\underline{x}_1, \overline{x}_2) + \max(1 - p_1 - p_2, 0) \cdot \min(\underline{x}_1, \underline{x}_2).$$

- For max in the general case, $\underline{E} = \max(E_1, E_2)$ and

$$\overline{E} = \min(p_1, p_2) \cdot \max(\overline{x}_1, \overline{x}_2) + \max(p_1 - p_2, 0) \cdot \max(\overline{x}_1, \underline{x}_2) +$$
$$\max(p_2 - p_1, 0) \cdot \max(\underline{x}_1, \overline{x}_2) + \min(1 - p_1, 1 - p_2) \cdot \max(\underline{x}_1, \underline{x}_2).$$

- Similar formulas can be produced for the cases when there is a strong correlation between x_i: namely, when x_1 is (non-strictly) increasing or decreasing in x_2.

From Elementary Arithmetic Operations to General Algorithms: First Idea. In general, we have an algorithm $f(x_1, \ldots, x_n)$ from R^n to R, we have n intervals $\mathbf{x}_i$ that contain the actual (unknown) values of x_i, and we have n more intervals $\mathbf{E}_i$ that contain the expected values $E[x_i]$. Our goal is to find the range $\mathbf{y}$ of $y = f(x_1, \ldots, x_n)$ and the range $\mathbf{E}$ of possible values of $E[y]$.

To compute $\mathbf{y}$, we can use known interval computations techniques. How can we compute $\mathbf{E}$? Our first idea is to compute $\mathbf{E}$ along the same lines as straightforward interval computations:

- first, we parse the algorithm $f(x_1, \ldots, x_n)$, i.e., we represent this algorithm as a sequence of arithmetic operations;
- then, we replace each elementary operation on real numbers with the corresponding operation on quadruples $(\underline{x}, \underline{E}, \overline{E}, \overline{x})$.

At the end, as one can easily prove, we get an enclosure for the range $\mathbf{y}$ of y *and* an enclosure for the range $\mathbf{E}$ of $E[y]$.

From Elementary Arithmetic Operations to General Algorithms: Second Idea. When we have a complex algorithm f, then the above step-by-step approach leads to excess width. How can we find the actual range of $E = E[y]$?

At first glance, the exact formulation of this problem requires that we use infinitely many variables, because we must describe all possible probability distributions on the box $\mathbf{x}_1 \times \ldots \times \mathbf{x}_n$ (or, in the independent case, all possible tuples consisting of distributions on all n intervals $\mathbf{x}_1, \ldots, \mathbf{x}_n$). It turns out, however, that we can reformulate these problems in equivalent forms that require only finitely many variables:

Theorem 10. *For a general continuous function $f(x_1, \ldots, x_n)$, $\underline{E}$ is a solution to the following optimization problem:* $\sum_{j=0}^{n} p^{(j)} \cdot f(x_1^{(j)}, \ldots, x_n^{(j)}) \to \min$ *under the conditions*

$$\sum_{k=0}^{n} p^{(k)} = 1;\ \ p^{(j)} \geq 0;\ \ \underline{x}_i \leq x_i^{(j)} \leq \overline{x}_i;\ \ \underline{E}_i \leq \sum_{j=0}^{n} p^{(j)} \cdot x_i^{(j)} \leq \overline{E}_i\ \ (\text{for all } i, j),$$

and $\overline{E}$ is a solution to $\sum_{j=0}^{n} p^{(j)} \cdot f(x_1^{(j)}, \ldots, x_n^{(j)}) \to \max$ *under the same constraints.*

Thus, by using validated tools for solving the corresponding optimization problem, we can find the desired range of $E[y]$.

3.3 Open Problems

So far, we have provided explicit formulas for the elementary arithmetic operations $f(x_1, \ldots, x_n)$ for the case when we know the first order moments. What if, in addition to that, we have some information about second order (and/or higher order) moments of x_i? What will we be then able to conclude about the moments of y? Partial answers to this question are given in [37, 57, 62]; it is desirable to find a general answer.

4 Exact Real Arithmetic

In the last two decades, the theory of computability on (the *full* set of) real numbers developed very fast. Although there still are competing approaches, the Type-2-Theory of Effectivity (TTE) seems to be the most evolved [12, 33, 63]. Several software packages for exact real arithmetic have been written based on the concepts from TTE or equivalent theories. They all allow functional or imperative programming with *atomic* real objects $x \in \mathbb{R}$, while still being fully consistent with real calculus. In this section, we will compare important aspects of these implementations.

4.1 Non-real arithmetic

The starting point for any practical application of arithmetic is hardware-based fixed-size integer or floating point arithmetic (today usually 32 or 64 bit). For integer arithmetic, we have a canonical and rather intuitive semantics. This is not true for floating point numbers: Here the semantics is no longer canonical, but got so complicated with constructs like NaNs, infinities, +0, -0, directed roundings etc. that it was necessary to define the IEEE standards 754/854 to get a reliable behavior of this arithmetic.

Leaving hardware-based size arithmetic, there are the sets of integer or rational numbers, where again we have a canonical, 'mathematical' semantics that does not need to be standardized. At this time, the most prominent open source implementation on these sets surely is GMP [25], written in `C` with hand-optimized assembler parts.

Within the rational numbers, the multiple precision floating point numbers (i.e. generalizations of the hardware floating point numbers) play a special role: Again, there is no 'canonical', pure 'mathematical' semantics. Instead, the result of an operation like the division of 1 by 3 does not only depend on the arguments themselves, but also on additional parameters like the size available for the result, the chosen rounding mode etc. Some software packages, e.g. the important but older `Fortran77` MP package [11], do not even try to explicitly define the result of such operations, they only give a verified bound for the error. Others, namely the MPFR package [66], again try to follow the spirit of the IEEE standards 754/854 as closely as possible.

Extending the set of rational numbers, there is the set of algebraic numbers, where still it is possible to implement a pure mathematical oriented semantics. The disadvantage of corresponding implementations is that the evaluation of deeply nested arithmetic expressions (like the solution of linear systems) becomes almost infeasible. See e.g. [42], page 116: ...*you may have to wait a long time for the answer when the expression is complex.*

4.2 The Border of decidable Equality

For all of the different types of arithmetic above, it was possible to decide whether two data structures x, y depict the same number, i.e. a test on equality $\mathrm{val}(x) = \mathrm{val}(y)$ was decidable with more or less effort! In contrast to this, equality is not a decidable operation in TTE. This is the most obvious difference to the BSS model [9], where equality of real numbers was taken as a *basic* operation!

So an important question concerning the decidability of equality is: How far can equality be implemented in an usable manner beyond the algebraic numbers? To illustrate the problems arising we recall one still unproven attempt, the *Uniformity Conjecture* [56], in a slightly simplified version below. This conjecture tries to extend the decidability to expressions that just allow exponentiation and logarithm in addition to the algebraic operations:

Consider expressions E built as follows:

- basic objects are integers in decimal form
- expressions may be built iteratively using $A+B$, $A-B$, $A*B$, A/B, (A), $-A$, e^A, $\ln A$, $\sqrt[n]{A}$ from given expressions A and B (and integer constants n).

Now the conjecture is: If E does not contain subexpressions $E' = e^{E''}$ where the value $val(E'')$ of E'' is bigger than 1 (i.e. if E does not use exponentiation to create large numbers intermediately), then

$$\text{either} \quad val(E) = 0 \quad \text{or} \quad |val(E)| > 10^{-1.3 \cdot len(E)}$$

where the length $len(E)$ is simply the length of the string denoting E.

If this conjecture turned out to be true, then we would be able to decide whether two values a and b are equal by approximating the value $a - b$ with a sufficient high precision (depending on the length of the expressions A and B yielding a and b). But if we have a closer look on an example, we see that the conjecture would perhaps not be too helpful in practice: Just consider the following nonlinear iteration (sometimes called *logistic equation*) used as an important example e.g. by [36]

$$x_{i+1} = 3.75 \cdot x_i \cdot (1 - x_i) \quad , \quad x_0 = 1/2 \tag{8}$$

If we try the uniformity conjecture on this example, we get a sequence of expressions E_i with $len(E_{i+1}) = 2 \cdot len(E_i) + 10$ (using $3.75 = 15/4$ with length 4), so already testing whether $x_{25} - x_{25}$ is equal to zero gives an expression E with $len(E) \approx 870,000,000$; this would imply that we would need to evaluate E to far more than one billion decimal digits to get a reliable answer. So today, we must face severe problems if we try to implement an arithmetic allowing decidable equality also for non-algebraic numbers.

4.3 Approximate Real Arithmetic

Deciding equality has already been dropped for a large class of computations: Interval arithmetic, either for hardware based on the standards IEEE 754/854 or for software solutions with variable size.

Two recent implementation in this area are `filib++` [39] (allowing IEEE 754/854 floats as interval borders) and `MPFI` [55], a multiple precision interval arithmetic library in `C`, based on `MPFR`. Of course, the use of interval software implies that the user 'thinks' in intervals, i.e. we have the look and feel of interval arithmetic.

A well-known approach by O. Aberth goes beyond this: In his *precise computation software* [1] he implemented an (almost) exact arithmetic, using 'range' arithmetic

based on an own floating point software. This package, written in `C++`, is freely available on the internet (unfortunately, it does not compile cleanly with the recent `gcc3.x` compilers). It contains basic arithmetic, but extended with a calculus on elementary functions allowing $+$, $-$, $*$, $/$, x^y, sin, cos, tan, asin, acos, atan, e^x, $\ln x$, max, min as basic operations as well as e.g. integration and differentiation as higher level operators. Aberth uses a data type representing real numbers (constructible from the operations above, so we have the *look* of an exact real arithmetic). But the user still gets the *feel* of interval arithmetic: The implementation with range arithmetic is still essentially interval based, and these intervals may grow too large during computations leading to failures due to insufficient precision. An implementation of the sequence for the logistic equation (8) may look as follows:

```
long n, prec;
cin >> n; cin >> prec;  set_precision(prec);
real x=1/real(2);  real c=375/real(100);
for (i=1; i <= n; i++) {
  x=c*x*(one-x);
  if (i%10==0)  cout<<i<<" "<<x.str(2,20)<<endl;
}
```

If the second input parameter `prec` is too small for a first parameter n. the program fails. On the other hand, if the parameter is much too large, the computation time is unnecessarily high.

4.4 Implementations for exact real arithmetic

The main part of this section is a comparison of the following packages:

- CRCalc (Constructive Reals Calculator, [10])
- XR (eXact Real arithmetic, [14])
- IC Reals (Imperial College Reals, [22])
- iRRAM (iterative Real RAM, [45])

Of course, there exist a lot more packages, e.g. the 'Manchester Reals' package by David Lester, which unfortunately is not available to the public at the moment. A test of this package together with the IC Reals, iRRAM, and a few others can be found in [7].

Some common basic concepts of exact real arithmetic are the following:

- Real numbers are atomic objects. The arithmetic is able to deal with (almost) arbitrary real numbers, but the usual entrance to $\mathbb{R}$ is $\mathbb{Q}$.
- The implementations try to follow the theory of computability on the real numbers. This implies that computable functions are continuous, so tests of equality of numbers are not possible in general (they usually lead to an infinite loop if the arguments to be tested are equal).
- An important relaxation (called *multi-valued functions*) of the continuity restriction has been introduced by [13] and is implemented only in the iRRAM. A similar but less general concept are *lazy booleans* that first appeared in the IC Reals.

Two different basic methods of evaluation can be found in the packages:

- *Explicit computation diagrams*: During any computation, computation diagrams are built and maintained, leading to a quite high memory consumption. These diagrams are evaluated only at need using techniques like *lazy evaluation*, a concept primarily developed for functional programming languages. The evaluation of the diagrams usually is top-down, i.e. a recursive traversal from the root (giving the result) to the leaves (containing the arguments).
- *Implicit computation diagrams*: Instead of explicitly storing the diagrams (containing the full information on their real values), only snapshots of values are maintained. In addition, a small amount of relevant information (called decision history or multi-valued cache) is kept in order to be able to reconstruct better approximations at need. This in general implies that parts of a computation or even a whole computation have to be iterated. In addition, the evaluation of the computations could be called bottom-up, as it necessarily proceeds from the arguments of a computation to its result.

Before comparing the performance of the four packages, we would like to point out some characteristic properties for each of the packages:

- **CRCalc (Constructive Reals Calculator)**: This package by H. Boehm is a `JAVA` implementation. During a computation, it constructs explicit computation diagrams using methods from object oriented programming. At need, these diagrams are evaluated top down representing multiple precision numbers as scaled `BigInteger`. A sample program for the logistic equation sequence looks as follows:

```
CR one = CR.valueOf(1);
CR C = CR.valueOf(375).divide(CR.valueOf(100));
CR X = one.divide(CR.valueOf(2));
for (int i=1;i<param;i++){
   X= C.multiply(X).multiply(one.subtract(X));
   if (i%10==0) {
     { System.out.print(i); System.out.print(" ");
       System.out.println(X.toString(20));  }
}
```

- **XR (eXact Real arithmetic)**: Based on an extension `FC++` of `C++` towards functional languages, Keith Briggs implemented `XR`, where a real number x is represented as a function $\lambda : \mathbb{Z} \to \mathbb{Z}$ in `FC++` via `typedef Fun1<int,Z> lambda`. This representation is defined as $x = \lim_{i\to\infty} \lambda(i) \cdot 2^{-i}$, i.e. $\lambda(i)$ is an approximation to x with absolute error 2^{-i}. Hence it is reasonable to implement the argument type of λ as ordinary 32-bit-integers, but to use arbitrarily long integers as result type. The sample program looks like

```
int i;
XR c=QQ(375,100),x=QQ(1,2);
```

```
cout<<setprecision(20);
for (i=0; i<=param; i++) {
  x=c*x*(1-x);
  if (i%10==0) cout<<i<<" "<<x<<endl;
}
```

– **IC Reals (Imperial College)**: The previous examples were implementations, where the underlying representations for a real number x essentially were normed Cauchy sequences a_i with $|x - a_i| \leqslant 2^{-i}$. Errington, Krznaric, Heckmann, et al. used linear fractional transformations (e.g. [21]) instead to implement a C-package. These LFTs are a generalization of continued fractions, here a real number is represented by a sequence of integers (a_i, b_i, c_i, d_i) (using GMP big integers) with $x_{i+1} = \dfrac{a_i x_i + b_i}{c_i x_i + d_i}$ and $x = \lim x_i$. Again we have explicit computation diagrams with a top-down lazy evaluation. One remarkable point is the use of lazy boolean predicates together with a multi-valued evaluation `realIF` that allows to implement non-continuous overlapping choices, like e.g.

```
y=realIF(2, x<1,a, x>-1,b)
```

to implement the following assignment

$$y := \begin{cases} a, \text{ if } x < 1 \\ b, \text{ if } x > -1 \end{cases}$$

– **iRRAM (iterative Real RAM)**: This C++-package written by Müller is the only package for exact real arithmetic that does not use explicit computation diagrams. Instead, it works with finite precision interval representations of reals numbers similar to Aberth' package. A big difference is a built-in mechanism to repeat computations in case of failing interval operations. Additionally, many concepts from TTE have been implemented, like operators for the evaluation of limits of sequences or explicit multivalued functions as well as lazy boolean similar to the IC reals package. The example function here looks as follows:

```
REAL x = 0.5;  REAL c = 3.75;
for ( int i=1; i<=param; i++ ) {
  x= c*x*(1-x);
  if ( (i%10)==0 ) { rwrite(x,18); rprintf(" %d\n",i); }
}
```

In the following we will compare the different implementations using two examples: the sequence from the logistic equation (8) and the harmonic series.

The maintenance of explicit computation diagrams can be very hard concerning memory consumption, nevertheless building the diagrams for the logistic sequence to e.g. x_n with $n \approx 10000$ should pose no problems here. On the other hand, due to its recursive definition and its chaotic nature, the evaluation of x_n is quite difficult. Ordinary floating point hardware delivers totally wrong values for x_n from about $n \approx 100$. If x_n is computed with interval methods, the sizes of the intervals grow almost

with the involved factor of 3.75. So to get an approximation of x_n with an error of 2^{-k}, we need initial approximations for x_0, x_1 etc. with a precision of about $2^{-k-1.91n}$, i.e. for $n = 10000$ a precision of about 19000 bits is required. For the benchmarks, we tried to compute x_n with about 20 significant decimals for different values of n, using the example programs from above.

A second example, where the loss of precision is much smaller, is the computation of the harmonic series $h(n) := \sum_{i=1}^{n} \frac{1}{i}$ We still have deeply nested operations, so this is a simple example to model e.g. the effects of basic linear algebra. We implemented $h(n)$ in all the packages and measured the time necessary to compute approximations to $h(n)$ with 10 decimals for rather large n.

The following timing results were obtained on a Pentium-3 with 1200 MHz; here "—" indicates that computations took longer than an hour or used more that 500 MB memory, so we canceled them.

package	logistic sequence		harmonic series		
	n=1,000	n=10,000	n=5,000	n=50,000	n=5,000,000
CRCalc	1359 sec	—	325 sec	—	—
XR2.0	423 sec	—	2.48 sec	2027 sec	—
IC-Reals	1600 sec	—	0.85 sec	—	—
Aberth	0.5 sec	1468 sec	<0.1 sec	0.3 sec	1835 sec
iRRAM	<0.1 sec	17 sec	<0.1 sec	0.1 sec	8.5 sec

The timings show that, at least for the two given problems, the advantage of the iterative approach compared to the explicit computation diagrams is so dramatic that the explicit approach seems to be unrealistic. In the package of Aberth, the error propagation seems to be done in an suboptimal way: The precision needed for the logistic sequence at $n = 10000$ was about 58000 bits instead of the sufficient 19000 bits. The same holds for the harmonic series, here a precision of about 14000 bits was necessary to compute $h(n)$ for $n = 5,000,000$. The iRRAM was able to do this using an internal precision of less than 50 bits.

The example of the harmonic series shows that the iRRAM is capable to deliver about 1 MFlops on the given CPU. As a comparison: The interval arithmetic `filib++` [39] based on hardware floats delivers about 8-22 MFlops on a Pentium IV with 2000 MHz. If we take into account the different speeds of the CPUs, then the exact arithmetic is just a factor 5 to 10 slower than a hardware based interval arithmetic, at least for cases where precision is not a critical factor.

To consider the influence of the necessary precision, we additionally used the iRRAM to compute approximations (with maximal error 2^{-50}) of the inverse of the (bad conditioned) Hilbert matrix H_n of size $n \times n$ using Gaussian elimination and compared this to the same computation applied to the well conditioned matrix $H_n + 1_n$ of the same size $n \times n$:

n	50	100	150	200	250	500
inversion of well conditioned matrix $H_n + 1_n$ of size $n \times n$						
bits	100	100	100	100	100	162
time	0.7 s	5.4 s	19 s	45s	91 s	1237 s
inversion of Hilbert matrix H_n of size $n \times n$						
bits	1037	2745	4372	5502	6915	—
time	3.2 s	79 s	457 s	1200 s	3052 s	—

Obviously, the condition of the matrix has big influence on the internal precision that is maintained automatically by the iRRAM package, which explains the big differences in execution time between the two examples.

Similar computations were done using `octave` (a freely available high-level interactive language for numerical computations without interval arithmetic). `octave` is already unable to invert the Hilbert matrix of size 12. On the other hand, the inversion of the well-conditioned matrix $H_n + 1_n$ with $n = 500$ takes only 18.8 s, so here the iRRAM is about a factor of 65 slower.

As a last example, we compared the performance of a few trigonometric functions between `MPFR` (using software arithmetic with non-interval methods but with verified roundings) and an extension to `MPFR` (found in the iRRAM package) that uses subroutines from the iRRAM to compute those functions:

	10 decimals		10000 decimals	
x=$\sqrt{3}$	`MPFR`	`MPFR+iRRAM`	`MPFR`	`MPFR+iRRAM`
e(x)	0.0117 ms	0.0577 ms	201 ms	1080 ms
ln(x)	0.0388 ms	0.0696 ms	105 ms	109 ms
sin(x)	0.0427 ms	0.0745 ms	403 ms	187 ms
cos(x)	0.0270 ms	0.0678 ms	282 ms	184 ms

Again, the overhead due to the much more elaborate exact arithmetic in the iRRAM is remarkably small, in some cases the algorithms using exact arithmetic were even faster.

As a summary, we may say that exact real arithmetic is on its way to be useful either as a reference implementation or as a tool to handle precision critical computations.

5 Conclusions

Traditional design of numerical software with result verification is based on the assumption that we know the algorithm $f(x_1, \ldots, x_n)$ that transforms input $x_1, \ldots, x_n$ into the output $y = f(x_1, \ldots, x_n)$, and we know the intervals $\mathbf{x}_1, \ldots, \mathbf{x}_n$ of possible values of the inputs. Many real-life problems go beyond this paradigm:

- In some cases, we do not have an algorithm f, we only know some relation (constraints) between x_i and y.
- In other cases, in addition to knowing the intervals $\mathbf{x}_i$ of possible values of x_i, we may have some additional information:

- we may have known some relation *between* different quantities x_i;
- we may also have some additional information about each of these quantities:
  - we may have some information about the *probabilities* of different values of x_i;
  - in some cases, we may even know the *exact* values of some of the inputs (e.g., we may know that $x_1 = \pi/2$).

To cover this additional information, in addition to traditional interval techniques, we must also have:

- techniques that translate known constraints between x_i and y into an algorithm that inputs the values $x_1, \ldots, x_n$ and computes the value(s) y that satisfies all the given constraints (for given values of x_i);
- techniques that use the algorithm f, the ranges $\mathbf{x}_i$, *and* additional constraints between x_i and y to get a better estimate for the range of possible values of y;
- techniques that use the additional information about probabilities of different values of $x_i \in \mathbf{x}_i$ to come up with the information about the probabilities of possible values of $y = f(x_1, \ldots, x_n)$; and
- techniques that would enable us to deal with *exact* real numbers in addition to the numbers known with interval uncertainty.

In this paper, we describe the approaches for designing these techniques. The main remaining challenge is to *combine* these techniques into a single working tool.

Acknowledgments

L.G. was supported by PAI-Procope project. V.K. was supported by NASA grant NCC5-209, by the AFOSR grant F49620-00-1-0365, by NSF grants EAR-0112968 and EAR-0225670, by IEEE/ACM SC2001 and SC2002 Minority Serving Institutions Participation Grants, by a research grant from Sandia National Laboratories as part of the Department of Energy Accelerated Strategic Computing Initiative (ASCI), and by Small Business Innovation Research grant 9R44CA81741 to Applied Biomathematics from the National Cancer Institute (NCI), a component of NIH.

The authors are thankful to the organizers of the Dagstuhl meeting for their support and encouragement, and to the anonymous referees for valuable suggestions.

References

1. Aberth, O.: Precise Numerical Methods Using C++, Academic Press, Boston (1998)
2. Apt, K.R.: The Role of Commutativity in Constraint Propagation Algorithms. ACM Transactions on Programming Languages and Systems **22**, No. 6 (2000) 1002–1036
3. Benhamou, F., Goualard, F. Universally Quantified Interval Constraints. In: Dechter, R. (ed.): Proceedings of of CP'2000, Principles and Practice of Constraint Programming, Springer Lecture Notes in Computer Science **1894** (2000) 67–82
4. Benhamou, F., Goualard, F., Granvilliers, L., and Puget, J.-F.: Revising Hull and Box Consistency. In de Schreye, D. (ed.): Proceedings of ICLP'99, International Conference on Logic Programming, Las Cruces, USA, MIT Press (1999) 230–244

5. Benhamou, F., McAllester, D., and Van Hentenryck, P.: CLP(Intervals) Revisited. In: Bruynooghe, M. (ed.): Proceedings of ILPS'94, International Logic Programming Symposium, Ithaca, USA, 1994. MIT Press (1994) 124–138.
6. Benhamou, F., Older, W.J.: Applying Interval Arithmetic to Real, Integer and Boolean Constraints. Journal of Logic Programming **32**, No. 1 (1997) 1–24
7. Blanck, J.: Exact Real Arithmetic Systems: Results of Competition, Springer Lecture Notes in Computer Science **2064** (2001) 389–394
8. Bliek, C., Neveu, B., Trombettoni, G.: Using Graph Decomposition for Solving Continuous CSPs. In: Maher, M., J.-F. Puget, J.-F. (eds.): Proceedings of CP'98, Principles and Practice of Constraint Programming, Springer Lecture Notes on Computer Science **1520** (1998) 102–116
9. Blum, L., Shub, M., Smale, S.: On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines, Bulletin of the AMS **21** (July 1989) 1
10. Boehm, H. Constructive Reals Calculator, `http://www.hpl.hp.com/personal/Hans_Boehm/new_crcalc/CRCalc.html`
11. Brent, R.P.: A Fortran multiple precision package, ACM Trans. Math. Software **4** (1978) 57–70
12. Brattka, V.: Recursive characterisation of computable real-valued functions and relations, Theoret. Comput. Sci. **162** (1996) 47–77
13. Brattka, V., Hertling, P.: Continuity and Computability of Relations, Informatik Berichte 164-9/1994, FernUniversität Hagen
14. Briggs, K. XR exact real arithmetic, `http://more.btexact.com/people/briggsk2/XR.html`
15. Christie, M., Languénou, E., Granvilliers, L.: Modeling Camera Control with Constrained Hypertubes. In: Van Hentenryck, P. (ed.): Proceedings of CP'2002, Principles and Practice of Constraint Programming, Springer Lecture Notes on Computer Science **2470** (2002) 618–632
16. Cleary, J.G.: Logical Arithmetic. Future Computing Systems **2**, No. 2 (1987) 125–149
17. Collavizza, H., Delobel, F., Rueher, M.: Comparing Partial Consistencies. Reliable Computing **5**, No. 3 (1999) 213–228
18. Collavizza, H., Delobel, F., Rueher, M.: Extending Consistent Domains of Numeric CSP. In: Proceedings of IJCAI'99, International Joint Conference on Artificial Intelligence, Morgan Kaufmann (1999) 406–413
19. Davis, E.: Constraint Propagation with Interval Labels. Artificial Intelligence **32** (1987) 281–331
20. Deville, Y., Jansen, M., Van Hentenryck, P.: Consistency Techniques in Ordinary Differential Equations. In: Proceedings of CP'1998, Principles and Practice of Constraint Programming, Springer Lecture Notes on Computer Science **1520** (1998) 162–176
21. Edalat, A., Heckmann, R.: Computing with real numbers: (i) LFT approach to real computation, (ii) Domain-theoretic model of computational geometry. In: Barthe, G., Dybjer, P., Pinto, L., and Saraiva, J. (eds): Springer Lecture Notes in Computer Science (2002)
22. Errington, L., Heckmann, R.: Using the IC Reals library, `http://www.doc.ic.ac.uk/~ae/exact-computation/ic-reals-manual.pdf`
23. Ferson, S., Ginzburg, L., Kreinovich, V., Longpré, L., Aviles, M.: Computing Variance for Interval Data is NP-Hard, ACM SIGACT News **33** (2002) 108–118.
24. Fischer, X., Nadeau, J.-P., Sébastian, P., Joyot, P.: Qualitative Constraints in Integrated Design. In: Chedmail, P. Cognet, G., Fortin, C., Mascle, C., Pegna, J. (eds.): Proceedings of ID-

MME'2000, Integrated Design and Manufacturing in Mechanical Engineering Conference, Montréal, Canada, Kluwer Academic Publishers (2002) 35–42
25. Granlund, T.: GMP 4.1, http://www.swox.com/gmp/
26. Granvilliers, L.: On the Combination of Interval Constraint Solvers. Reliable Computing **7**, No. 6 (2001) 467–483
27. Granvilliers, L., Benhamou, F.: Progress in the Solving of a Circuit Design Problem. Journal of Global Optimization **20**, No. 2 (2001) 155–168
28. Granvilliers, L., Monfroy, E., Benhamou, F.: Symbolic-Interval Cooperation in Constraint Programming. In: Proceedings of ISSAC'2001, International Symposium on Symbolic and Algebraic Computations, ACM Press (2001) 150–166
29. Hansen, E.: Sharpness in interval computations, Reliable Computing **3** (1997) 7–29.
30. Jaulin, L. Kieffer, M., Didrit, O., Walter, E.: Applied Interval Analysis: With Examples in Parameter and State Estimation, Robust Control and Robotics, Springer, London (2001)
31. Kearfott, R.B.: Rigorous Global Search: Continuous Problems, Kluwer, Dordrecht (1996)
32. Kearfott, R.B., Kreinovich, V. (eds.): Applications of Interval Computations, Kluwer, Dordrecht (1996)
33. Ko, K.-I.: Complexity Theory of Real Functions, Birkhäuser, Boston (1991)
34. Kreinovich, V.: Probabilities, Intervals, What Next? Optimization Problems Related to Extension of Interval Computations to Situations with Partial Information about Probabilities, Journal of Global Optimization (to appear).
35. Kreinovich, V., Lakeyev, A., Rohn, J., Kahl, P.: Computational Complexity and Feasibility of Data Processing and Interval Computations, Kluwer, Dordrecht (1997)
36. Kulisch, U.: Memorandum über Computer, Arithmetik und Numerik, Universität Karlsruhe, Institut für Angewandte Mathematik (1996)
37. Kuznetsov, V.P.: Interval Statistical Models, Radio i Svyaz, Moscow (1991) in Russian
38. Lebbah, Y., Rueher, M., Michel, C.: A Global Filtering Algorithm for Handling Systems of Quadratic Equations and Inequations. In: Van Hentenryck, P. (ed.): Proceedings of CP'2002, Principles and Practice of Constraint Programming, Ithaca, NY, USA, Springer Lecture Notes in Computer Science **2470** (2002)
39. Lerch, M., Tischler, G., Wolff von Gudenberg, J., Hofschuster, W., Krämer, W.: The Interval Library filib++ 2.0 - Design, Features and Sample Programs, Preprint 2001/4, Universität Wuppertal (2001)
http://www.math.uni-wuppertal.de/wrswt/literatur/lit_wrswt.html
40. Lhomme, O.: Consistency Techniques for Numeric CSPs. In: Wahlster, W. (ed.): Proceedings of IJCAI'93, International Joint Conference of Artificial Intelligence, Chambéry, France, 1993. Morgan Kaufman (1993) 232–238
41. Lhomme, O., Gotlieb, A., Rueher, M.: Dynamic Optimization of Interval Narrowing Algorithms. Journal of Logic Programming **37**, No. 1–2 (1998) 165–183
42. Mehlhorn, K., Näher, S.: LEDA, Cambridge University Press (1999)
43. Moore, R.E.: Interval Analysis, Prentice-Hall, Englewood Cliffs, NJ (1966)
44. Moore, R.E.: Methods and Applications of Interval Analysis, SIAM, Philadelphia (1979)
45. Müller, N.Th.: iRRAM - Exact Arithmetic in C++,
http://www.informatik.uni-trier.de/iRRAM/
46. Nedialkov, N.S.: *Computing Rigorous Bounds on the Solution of an Initial Value Problem for an Ordinary Differential Equation*. PhD thesis, University of Toronto (1999)
47. Neumaier, A.: Interval Methods for Systems of Equations Cambridge University Press (1990)
48. Nivlet, P., Fournier, F., and Royer, J.: A new methodology to account for uncertainties in 4-D seismic interpretation, Proceedings of the 71st Annual International Meeting of the Society

of Exploratory Geophysics SEG'2001, San Antonio, Texas, September 9–14 (2001) 1644–1647.
49. Nivlet, P., Fournier, F., Royer, J.: Propagating interval uncertainties in supervised pattern recognition for reservoir characterization, Proceedings of the 2001 Society of Petroleum Engineers Annual Conference SPE'2001, New Orleans, Louisiana, September 30–October 3 (2001) paper SPE-71327.
50. W. Older, W., Vellino, A.: Constraint Arithmetic on Real Intervals. In: Benhamou, F., Colmerauer, A. (eds.): Constraint Logic Programming: Selected Research, MIT Press (1993)
51. O'Sullivan, B.: Constraint-Aided Conceptual Design, PhD thesis, University College Cork (1999)
52. Puget, J.-F., Van Hentenryck, P.: A Constraint Satisfaction Approach to a Circuit Design Problem, Journal of Global Optimization **13**, No. 1 (1998) 75–93
53. Rabinovich, S.: Measurement Errors: Theory and Practice, American Institute of Physics, New York (1993)
54. Ratschan, S.: Continuous First-Order Constraint Satisfaction. In: Calmet, J., Benhamou, B., Caprotti, O., Henoque, L., Sorge, V. (eds.): Proceedings of AISC'2002, International Conference on Artificial Intelligence and Symbolic Computations, Springer Lecture Notes on Computer Science **2385** (2002) 181–195
55. Revol, N., Rouillier, F.: Motivations for an arbitrary precision interval arithmetic and the MPFI library Research report R 2002-27, LIP, École Normale Supérieure de Lyon (2002)
56. Richardson, D.: The Uniformity Conjecture, Proceedings of CCA2000, Springer Lecture Notes in Computer Science **2064** (2001) 253–272
`http://www.bath.ac.uk/~masdr/unif.dvi`
57. Rowe, N. C.: Absolute bounds on the mean and standard deviation of transformed data for constant-sign-derivative transformations, SIAM Journal of Scientific Statistical Computing **9** (1988) 1098–1113
58. Sam Haroud, D., Faltings, B.: Consistency Techniques for Continuous Constraints. Constraints **1** (1996) 85–118
59. Van Hentenryck, P., Mc Allester, D., Kapur, D.: Solving Polynomial Systems using a Branch-and-Prune Approach. SIAM Journal on Numerical Analysis **34**, No. 2 (1997) 797–827
60. Van Hentenryck, P., Michel, L., Deville, Y.: Numerica: a Modeling Language for Global Optimization. MIT Press (1997)
61. Vavasis, S.A.: Nonlinear Optimization: Complexity Issues, Oxford University Press, N.Y. (1991)
62. Walley, P.: Statistical Reasoning with Imprecise Probabilities, Chapman and Hall, N.Y. (1991)
63. Weihrauch, K.: Computable Analysis. An Introduction, Springer, Berlin (2000)
64. Yamamura, K., Kawata, H., Tokue, A.: Interval Analysis using Linear Programming. BIT **38** (1998) 188–201
65. Xiang, G.: unpublished manuscript.
66. Zimmermann, P.: MPFR: A Library for Multiprecision Floating-Point Arithmetic with Exact Rounding, 4th Conference on Real Numbers and Computers, Dagstuhl (2000) 89–90
`http://www.loria.fr/projets/mpfr/`