Solving Existentially Quantified Constraints with One Equality and Arbitrarily Many Inequalities

Stefan Ratschan

Max-Planck-Institut für Informatik, Saarbrücken, Germany
stefan.ratschan@mpi-sb.mpg.de

Abstract. This paper contains the first algorithm that can solve disjunctions of constraints of the form $\exists \boldsymbol{y} \in B\ [f = 0\ \wedge\ g_1 \geq 0 \wedge \ldots \wedge g_k \geq 0]$ in free variables $\boldsymbol{x}$, terminating for all cases when this results in a numerically well-posed problem. The only assumption on the terms $f, g_1, \ldots, g_n$ is the existence of a pruning function, as given by the usual constraint propagation algorithms or by interval evaluation. The paper discusses the application of an implementation of the resulting algorithm on problems from control engineering, parameter estimation, and computational geometry.

1 Introduction

Dealing with uncertainty is an important challenge for constraint programming. An important way of modeling bounded (in contrast to stochastic [33]) uncertainty uses the logical quantifiers $\forall$ and $\exists$—as illustrated by an over-60-paper-bibliography on applications of solving constraints with quantifiers [25]. However, the problem of solving real-number constraints with quantifiers is undecidable in general [32], and very hard for special cases [35, 9].

This paper is part of a research program on solving real-number constraints with quantifiers, with only the two restrictions of numerical well-posedness and existence of a pruning algorithm for the individual (atomic) constraints. The case we consider in this paper are disjunctions of constraints of the form $\exists \boldsymbol{y} \in B\ [f = 0 \wedge g_1 \geq 0 \wedge \ldots \wedge g_n \geq 0]$ in free variables $\boldsymbol{x}$. For example, this case is very important in parameter estimation [17]. For disproving such constraints or computing elements that are not in the solution set, one can use a method for solving constraint with quantifiers and no equality predicate symbols [29, 28]. However, proving such constraints or computing elements in its solution set introduces significant additional difficulties: First, the non-empty solution set of an equality might not contain any rational (or even real algebraic) number, as in the example $\exists x\ [\sin x = 0 \wedge x \geq 3 \wedge x \leq 4]$. Second, the branching step of the mentioned algorithms [29, 28] does not necessarily decrease the difficulty for such constraints. On the contrary—it can even produce a new, numerically ill-posed problem!

For the design of the algorithm introduced in this paper we use the following main objectives: First, it should terminate for all cases that are numerically well-posed. And second, computed positive and negative information should be mutually used within the algorithm.

We proceed by giving a criterion characterizing the numerical well-posedness of such constraints, extending the case [29] of just inequalities. Using this criterion we design an algorithm that is based on the usual branch-and-prune scheme extended with an additional checking step for proving the truth of the input constraint on a part of the free-variable space. Here branching not only splits bounds on the free-variables, but can also split an existential quantifier into a disjunction of two existential quantifiers. If desired, the algorithm can also return witnesses in the case of a positive result. We have implemented the algorithm and applied it to problems in control engineering and parameter estimation.

The structure of the paper is as follows: In Section 2 we introduce some basic notions; in Section 3 we present the main branch-prune-and-check algorithm; in Section 4 we give a characterization of the stability of such constraint; in Section 5 we discuss the pruning step, in Section 6 the branching step, in Section 7, the checking step; in Section 8 we apply the results to the main algorithm; in Section 9 we discuss an implementation of the algorithm and we apply it to simple application examples; in Section 10 we discuss related work; and in Section 11 we conclude the paper.

2 Preliminaries

In this paper we concentrate on a certain type of constraints:

Definition 1. *An* E-constraint *is a constraint of the form*

$$\begin{gathered}\exists \boldsymbol{y} \in B_1\ [f_1 = 0 \wedge g_{1,1} \geq 0 \wedge \ldots \wedge g_{1,k_1} \geq 0] \\ \vee \ldots \vee \\ \exists \boldsymbol{y} \in B_n\ [f_n = 0 \wedge g_{n,1} \geq 0 \wedge \ldots \wedge g_{n,k_n} \geq 0]\end{gathered}$$

where

- $B_1, \ldots, B_n$ *are boxes of the same dimension as the length of the variable vector* $\boldsymbol{y}$*, and*
- $f_1, g_{1,1}, \ldots, g_{1,k_1}, \ldots, f_n, g_{n,1}, \ldots, g_{n,k_n}$ *are terms built from a fixed set of function symbols (e.g.,* $+$*,* $.$*,* $\sin$*,* $\cos$*,* $\exp$*) to which we give their usual meaning over the real numbers.*

Within this paper we call an E-constraint or any sub-constraint of an E-constraint a *constraint*. A bounded constraint is a pair consisting of a constraint ϕ in n free variables, and a box $B \subseteq \mathbb{R}^n$ (the *free-variable-bound*). A bounded constraint is true/false iff it is true/false for all elements of the bound. As usual in mathematics we will use the same notation to write down a term and the function it denotes.

We will use a few basic notions from analysis such as continuity, Bolzano intermediate value theorem, convergence of sequences. Furthermore, we say that a sequence of sets over $\mathbb{R}^n$ *converges* to a real vector $r \in \mathbb{R}^n$ iff all element sequences converge to r. In order to be able to use the convergence notion also for $n = 0$, we measure distance in $\mathbb{R}^0$ according to $d(x, y) = 0$ if $x = y$ and ∞, otherwise. We also say that a sequence $x_1, \ldots$ *eventually* fulfills a property P iff there is a k such that for all $i \geq k$, $P(x_i)$ holds. Finally, we define the *width of a box* B *(*$w(B)$*)* to be the maximum of the width of its component intervals.

3 Overall Algorithm

One can extend numerical constraint satisfaction methods [10, 7, 3] to constraints that contain quantifiers [26, 28]. This uses a branch-and-prune framework, where pruning tries to prove or disprove (parts of) the input constraints and if this fails, branching tries to decrease the difficulty by splitting one of the quantifiers into subproblems.

However, for constraints that contain equalities or disequalities this fails, because equalities have solution sets without volume, and disequalities solution set complements without volume. For the example $\exists x\ [\sin x = 0 \wedge x \geq 3 \wedge x \leq 4]$ pruning will never compute a real number x fulfilling $\sin x = 0$ and so the method will never prove the whole constraint. In general, for E-constraints, pruning will disprove a false constraint (compute false elements) without problems. However, it fails in proving a true constraint (computing true elements).

In order to remedy this situation, we modify the according branch-and-prune algorithm, resulting in Algorithm 1. The main change consists of an additional checking step that tries to prove constraints. By letting pruning (for computing negative information) and checking (for computing positive information) work on the same constraints, one can take advantage of the results of the other.

Algorithm 1 Solver

Input: (ϕ, B): a bounded E-constraint, $\varepsilon \in \mathbb{R}^+$
Output: T, F: a set of boxes such that

- $T \subseteq B$, $F \subseteq B$,
- ϕ is true on all elements of T,
- ϕ is false on all elements of F, and
- the volume of $B \setminus T \setminus F$ is smaller than ε

$C \leftarrow \{(\phi, B)\}$
$C' \leftarrow C$; $C \leftarrow \mathrm{Prune}(C)$; $F \leftarrow F \cup \mathrm{Diff}(C', C)$
$C' \leftarrow C$; $C \leftarrow \mathrm{Check}(C)$; $T \leftarrow T \cup \mathrm{Diff}(C', C)$
while the volume of $\{B \mid (\phi, B) \in C\}$ is greater or equal ε **do**
  $C \leftarrow \mathrm{Branch}(C)$
  $C' \leftarrow C$; $C \leftarrow \mathrm{Prune}(C)$; $F \leftarrow F \cup \mathrm{Diff}(C', C)$
  $C' \leftarrow C$; $C \leftarrow \mathrm{Check}(C)$; $T \leftarrow T \cup \mathrm{Diff}(C', C)$
end while

Here we have the following sub-algorithm specifications:

Diff: Takes two sets of bounded constraints and returns a set of boxes such that their union is equal to the closure of the difference of the free-variable-bounds of the inputs

Prune: Returns a set of bounded constraints whose free-variable bounds still contain all the solutions of the input.

Check: Returns the input except for some true bounded constraints.

Branch: Splits a sub-constraint of the form $\exists x \in B\ \phi$ into $\exists x \in B_1\ \phi \ \vee\ \exists x \in B_2\ \phi$ such that the union of B_1 and B_2 is B, and their intersection has zero volume, or splits a bounded constraint (ϕ, B) into (ϕ, B_1) and (ϕ, B_2), such that the union of B_1 and B_2 is B, and their intersection has zero volume.

Here we assume that in $\mathbb{R}^0$ the volume of the empty box is 0 and the volume of the non-empty box is ∞. In the algorithm, for closed inputs, the set C will never contain more than one element, and the algorithm terminates as soon as C becomes empty.

The reason, why such algorithms use a pruning step (instead of checking for false constraints also), is that often one can deduce information by pruning even when a checking step would fail. This allows us to keep the size of the problem small by avoiding branching. Now it is is easy to prove:

Theorem 1. *Algorithm 1 is correct.*

However, it is not clear when it will terminate. In the rest of the paper we will implement the sub-algorithms in such a way that we can prove termination in all cases that are numericall well-posed (in a sense that will we shortly define).

4 Stability of Constraints

Algorithms that involve rounding or approximation can only succeed on problems where this does not change the result in an essential way. Studying this phenomenon is one of the main tasks of the field of numerical analysis. In this section we undertake a similar endeavor for E-constraints. Readers who are interested only in algorithms for solving E-constraints and not in their detailed properties can skip this section.

Definition 2. *A constraint* ϕ' is a result of an ε-perturbation of a constraint ϕ *iff it results from* ϕ *by replacing the right-hand-side zeros of atomic constraints by terms denoting a continuous function whose co-domain is* $[-\varepsilon, \varepsilon]$.

Definition 3. *A closed* E*-constraint is* stable *iff there is a real number* $\varepsilon > 0$ *such that all results of* ε*-perturbations have the same truth value.*

For example, the constraint $\exists x \in [-2, 2]\ x^2 = 0$ is unstable: although it is true, it becomes false under small perturbations. On the other hand, the constraint $\exists x \in [-2, 2]\ x^2 - 1 = 0$ is stable.

In the previous definition, perturbations by constants—as for the case of inequality constraints [29]—do not suffice for capturing the E-constraints that are solvable by numerical methods. For example, for functions f and g as in Figure 1, the con-

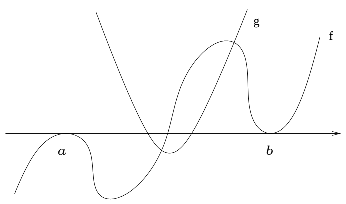


Fig. 1. Constant Perturbations Fail

straint $\exists x\ [f(x) = 0 \wedge g(x) \geq 0]$ would be stable and true, but the zero proving the existential quantifier jumps discontinuously between the zeros a and b. This obstructs the use of methods that involve rounding or approximation for such a problem.

Now, as in the case of inequality constraints [29], we introduce a number that replaces the discrete notion of truth of a constraint by a continuous one that is negative for false constraints, positive for true constraints, and such that the ease of proving this is proportional to its distance to zero. Similar to the notion of *condition number* in numerical analysis, this will allow us to study the difficulty of a problem (the essential difference being that, by using derivatives, conditions numbers concentrate on local information, while we want to capture the problem globally).

The idea is, that for proving a constraint of the form $\exists \boldsymbol{y} \in B\ [f = 0 \wedge g_1 \geq 0 \wedge \ldots \wedge g_k \geq 0]$, by Bolzano's intermediate value theorem, it suffices to prove that there is a path between two points within the quantification bound such that the inequalities $g_1 \geq 0 \wedge \ldots \wedge g_k \geq 0$ hold on the whole path, and the function f is non-positive at the beginning and non-negative at the end. We assume that the ease of proving an inequality between two real numbers is proportional to the distance of the two numbers; the ease of proving an inequality on all elements of a path is the minimal ease of proving the inequality on the path elements; and the ease of proving the whole E-constraint is proportional to the ease of performing the above on the easiest path:

Definition 4. *The* degree of truth of a closed constraint $\exists \boldsymbol{y} \in B\ [f = 0\ \wedge\ g_1 \geq 0 \wedge \ldots \wedge g_n \geq 0]$ on a path $P \subseteq B$ between two points $p_1 \in B$ and $p_2 \in B$ *is*

$$\min\{f(p_1), -f(p_2), \min_P g_1, \ldots, \min_P g_n\}$$

The degree of truth $\tau(\phi)$ of a closed E-constraint of the form $\exists \boldsymbol{y} \in B_1\ \phi_1 \vee \ldots \vee \exists \boldsymbol{y} \in B_n\ \phi_n$ *is the maximum of the degree of truth of a sub-constraint* $\exists \boldsymbol{y} \in B_i\ \phi_i$ *over all paths in* B_i *and all* $i \in \{1, \ldots, n\}$.

Here we can deal with p_1 and p_2 in an asymmetric way, because the set of all paths contains for each path its reverse. Also we can use the maximum (instead of the supremum) over all paths because the degree of truth is a continuous function on the compact set of paths; therefore the supremum of the image of the degree of truth on the set of paths is attained on one path.

The degree of truth determines the truth of a sentence as follows:

Theorem 2. *For a closed E-constraint, positive degree of truth implies truth, a negative degree of truth implies falsehood.*

Proof. Assume that the degree of truth of a closed E-constraint is positive. This means that there is a sub-constraint of the form $\exists \boldsymbol{y} \in B\ [f = 0 \wedge g_1 \geq 0 \wedge \ldots \wedge g_n \geq 0]$ and a path $P \subseteq B$ with endpoints $p_1 \in B$ and $p_2 \in B$ such that $f(p_1)$ is positive and $f(p_2)$ is negative. Furthermore, $g_1, \ldots, g_n$ are positive on P. Therefore, by the Bolzano intermediate value theorem, the sub-constraint is true, and since this sub-constraint is part of a disjunction, the whole constraint is true.

Now assume that the degree of truth is negative and the constraint is true. The truth of the constraint implies that there is a sub-constraint of the form $\exists \boldsymbol{y} \in B\ [f = 0 \ \wedge\ g_1 \geq 0 \wedge \ldots \wedge g_n \geq 0]$ such that f has a zero in B, and $g_1, \ldots, g_n$ are greater or equal zero there. So the degree of truth of the path that just contains this zero is zero, implying that the total degree of truth is greater or equal zero—a contradiction. □

For investigating the connection between degree of truth and stability, we use:

Lemma 1. *For every closed E-constraint ϕ and $\varepsilon > 0$ there is a result of an ε-perturbation of ϕ whose degree of truth is larger/smaller than the one of ϕ.*

Proof.

- Increasing the degree of truth: Let $\exists \boldsymbol{y} \in B\ [f = 0 \ \wedge\ g_1 \geq 0 \wedge \ldots \wedge g_n \geq 0]$ be the sub-constraint and let P be the path on which the maximum is attained. We have to find a perturbation such that $\min\{f(p_1), -f(p_2), \min_P g_1, \ldots, \min_P g_n\}$ increases. This can be easily done by a perturbation that increases the degree of truth of each element of $\{f(p_1), -f(p_2), \min_P g_1, \ldots, \min_P g_n\}$ [1].
- Decreasing the degree of truth: We find a perturbation that decreases the degree of truth of every existentially quantified sub-constraint and every path. Let $\exists \boldsymbol{y} \in B\ [f = 0 \ \wedge\ g_1 \geq 0 \wedge \ldots \wedge g_n \geq 0]$ and let P be an arbitrary, but fixed path. Their degree of truth is $\min\{f(p_1), -f(p_2), \min_P g_1, \ldots, \min_P g_n\}$. For each of its elements one can easily find a perturbation that decreases it.

□

The degree of truth characterizes its stability as follows:

Theorem 3. *A closed E-constraint is stable iff its degree of truth is non-zero.*

Proof. **⇒:** We assume a constraint with zero degree of truth and prove that it is unstable. This certainly holds because, by Lemma 1, a small perturbations will change the degree of truth, and by Theorem 2 this will also change the truth value.

⇐: Assume that the degree of truth is non-zero. Since the degree of truth depends continuously on perturbation, this implies that there is a $\varepsilon > 0$, such that under all perturbations less than ε, the degree of truth does not change its sign. Hence, by Theorem 2, it also does not change its truth value.

□

[1] For perturbing $f(p_1)$ and $f(p_2)$ independently we need perturbation by functions (instead of just constants) here, see Figure 1.

For false E-constraints, one can use a simpler characterization, that is compatible with the case of inequality constraints [26, 28].

Lemma 2. *The degree of truth of a false closed E-constraint of the form*

$$\begin{gathered}\exists \boldsymbol{y} \in B_1\ [f_1 = 0 \wedge g_{1,1} \geq 0 \wedge \ldots \wedge g_{1,k_1} \geq 0] \\ \vee \ldots \vee \\ \exists \boldsymbol{y} \in B_n\ [f_n = 0 \wedge g_{n,1} \geq 0 \wedge \ldots \wedge g_{n,k_n} \geq 0]\end{gathered}$$

is $\max_{i \in \{1 \ldots n\}} \sup_{\boldsymbol{y} \in B} \min\{f_i, -f_i, g_{i,1}, \ldots, g_{i,k_i}\}$.

Proof. We prove that for every $i \in \{1, \ldots, n\}$ the maximal path of the according disjunctive branch consists of just a point. Obviously this implies the lemma.

Since ϕ is false, by Theorem 2, $\tau(\phi)$ is non-positive, and so for every $i \in \{1, \ldots, n\}$ the degree of truth of every disjunctive branch on the maximal path P from p_1 to p_2 is non-positive.

Now there are three cases:

- There is a $j \in \{1, \ldots, k_i\}$ such that $\min\{f_i(p_1), -f_i(p_2), \min_P g_{i,1}, \ldots, \min_P g_{i,k_i}\}$ is $\min_P g_{i,j}$. In this case the minimum is attained on a certain point of P.
- The minimum of $\{f_i(p_1), -f_i(p_2), \min_P g_{i,1}, \ldots, \min_P g_{i,k_i}\}$ is $f_i(p_1)$: In this case, $-f_i(p_1)$ is not smaller than $f_i(p_1)$ and therefore the path just containing p_1 has the same degree of truth and therefore is also maximal.
- The minimum of $\{f_i(p_1), -f_i(p_2), \min_P g_{i,1}, \ldots, \min_P g_{i,k_i}\}$ is $-f_i(p_2)$: In this case, $f_i(p_2)$ is not smaller than $-f_i(p_2)$, and therefore the path just containing p_2 has the same degree of truth and therefore is also maximal.

□

5 Pruning

In this section we develop the pruning step of Algorithm 1. Here we assume a pruning algorithm for atomic bounded constraints (i.e., bounded constraints where the first element is an equality or inequality) and extend it to E-constraints.

We start with introducing certain properties that we assume for atomic pruning. These properties refine the properties postulated for the notion of "narrowing operator" [1, 4]. We will also use these properties later for implementing the branching and checking steps.

Note that branching can result in arbitrarily small boxes. So we have to use arbitrary precision arithmetic. However, the usual pruning techniques (for computing box-consistency [3], hull-consistency [7, 10] etc.) are defined for fixed precision. So we add an additional precision parameter to the pruning function (a similar parameter is sometimes used to prevent slow convergence [13]).

The first property we assume is, that atomic pruning should result in a bounded constraint with the same constraint and a smaller bound (still we return a full bounded constraint instead of just a box because, when extending pruning to E-constraints, we will also allow changes of the constraint):

Property 1 (Contractance). For an atomic bounded constraint (ϕ, B) and positive real number p, $Prune_p(\phi, B) = (\phi', B')$ implies that $\phi = \phi'$ and $B' \subseteq B$.

It should only remove elements not in the solution set:

Property 2 (Correctness). For an atomic bounded constraint (ϕ, B) and positive real number p, $Prune_p(\phi, B) = (\phi', B')$ implies that ϕ and ϕ' have the same solution set in B.

Pruning is monotonic in the following sense:

Property 3 (Monotonicity). For an atomic constraint ϕ, boxes B_1 and B_2 such that $B_1 \supseteq B_2$ and and positive real numbers p_1, p_2 such that $p_1 \leq p_2$, $\mathrm{Prune}_{p_1}(\phi, B_1) = (\phi', B_1')$ and $\mathrm{Prune}_{p_2}(\phi, B_2) = (\phi', B_2')$ implies $B_1' \supseteq B_2'$.

Pruning eventually succeeds for all well-posed inputs:

Property 4 (Convergence). For all atomic constraints ϕ and sequences of boxes $B_1, \ldots$ converging to a point at which ϕ is stably false, there is a natural number k and a real number p such that for all $k' \geq k$ and $p' \geq p$, $\mathrm{Prune}_{p'}(\phi, B_{k'})$ has an empty bound.

Pruning results in borders on which it will succeed using the same precision:

Property 5 (Prunable Borders). For atomic ϕ, such that $\mathrm{Prune}_p(\phi, B) = (\phi', B')$, for all new faces D of B' (i.e., faces of B' that are in the interior of B), $\mathrm{Prune}_p(\phi, D) = (\phi, \emptyset)$.

This property has two purposes: First, it will allow the Diff function of Algorithm 1 to include the new borders into the result. And second, it will allow the checking step to compute the necessary information on the new borders using the current precision.

Prune with a certain precision eventually reaches a fixpoint:

Property 6 (Fixed Point). For every positive real number p and infinite sequence $(\phi_1, B_1), \ldots$ of bounded constraints such that for every natural number i, B_{i+1} is the bound of $\mathrm{Prune}_p(\phi_i, B_i)$, there is a k such that $B_k = B_{k+1} = \ldots$.

Now we can extend such a pruning algorithm to E-constraints as required by Algorithm 1. For this we accordingly adapt the case of constraints with quantifiers, as introduced in earlier papers [26, 28]:

- For a set C of E-constraints, $\mathrm{Prune}(C) := \{\mathrm{Prune}(\phi, B) | (\phi, B) \in C\}$
- $\mathrm{Prune}(\phi_1 \vee \ldots \phi_n, B) := (\phi_1' \vee \ldots \vee \phi_n', B_1' \uplus \ldots \uplus B_n')$, where $(\phi_i', B_i') = \mathrm{Prune}(\phi_i, B_i)$
- $\mathrm{Prune}(\exists \boldsymbol{y} \in B^{\boldsymbol{y}}\ \phi, B^{\boldsymbol{x}}) := (\exists \boldsymbol{y} \in B^{\boldsymbol{y}'}\ \phi', B^{\boldsymbol{x}'})$,
  where $(\phi', B^{\boldsymbol{x}'} \times B^{\boldsymbol{y}'}) = \mathrm{Prune}_{\mathrm{prec}(B^{\boldsymbol{x}} \times B^{\boldsymbol{y}})}(\phi, B^{\boldsymbol{x}} \times B^{\boldsymbol{y}})$
- $\mathrm{Prune}_p(\phi_1 \wedge \ldots \wedge \phi_k, B) := fix(\{\mathrm{Prune}_p^i | 1 \leq i \leq k\})(\phi_1 \wedge \ldots \wedge \phi_k, B)$
- $\mathrm{Prune}_p^i(\phi_1 \wedge \ldots \wedge \phi_k, B) := (\phi_1 \wedge \ldots \wedge \phi_i' \wedge \ldots \wedge \phi_k, B')$
  where $(\phi_i', B') := \mathrm{Prune}_p(\phi_i, B)$

Here $\uplus$ denotes the smallest box containing the union of the argument boxes. The operator fix takes a set of functions and applies them to the second argument until a fixed point is reached (this fixed point exists by Property 6). The function prec() takes the Cartesian product of the bounds of the constraint and returns the desired precision. We only assume that this precision goes to infinity as the width of its argument goes to zero.

6 Save Branching

In this section we study the branching step (i.e., either splitting a sub-constraint of the form $\exists x \in B\ \phi$ into $\exists x \in B_1\ \phi \ \vee\ \exists x \in B_2\ \phi$ such that the union of B_1 and B_2 is B, and their intersection has zero volume, or splitting a bounded constraint (ϕ, B) into (ϕ, B_1) and (ϕ, B_2), such that the union of B_1 and B_2 is B, and their intersection has zero volume). Here we have to ensure that this will allow pruning or checking to eventually succeed on the result. How can it happen that this fails? On the one hand, branching can introduce an unstable constraint (e.g., by replacing $\exists y \in [-1, 1]\ x = 0$ by $\exists y \in [-1, 0]\ x = 0 \vee \exists y \in [0, 1]\ x = 0$). On the other hand, it can fail to decrease the sizes of the free-variable and quantification bound appropriately.

In order to prevent the first possibility we ensure:

Definition 5. *Given a bounded constraint (ϕ, B) and a constraint ϕ' created from ϕ by branching at a quantifier, the branching is* save *iff for all $\boldsymbol{x}_0 \in B$, ϕ' is stable at $\boldsymbol{x}_0$ if ϕ is stable at $\boldsymbol{x}_0$.*

One could easily ensure this by only branching the free-variable-bounds. However, then the second problem discussed above arises—the size of the quantification bound does not go to zero, and therefore pruning might never succeed to disprove a false constraint.

So we have to analyze the problem in more detail.

Lemma 3. *Branching a quantification bound in a false E-constraint is save.*

Proof. We prove that the degree of truth decreases by branching—by Theorems 2 and 3 this implies the lemma.

In each branch of the splitted sub-constraint the set of paths is a subset of the set of paths of the original constraint. Therefore the degree of truth of both paths decreases, and so also the degree of truth of the whole formula. □

So we can branch the quantification bound of a false constraint without problems. Still we have to take care when branching true constraints. For now, we defer the problem by simply assuming that checking of true constraint will even succeed if the width of the quantification bound does not go to zero. Therefore, we only have to branch the quantification bound of false constraints, which according to Lemma 3 is no problem.

So we have to following situation:

- For stably false constraints, both the free-variable bound and the quantification bound size should go to zero.
- For stably true constraints, only the free-variable bound size should go to zero.

However, during the algorithm, we do not yet know whether a constraint is true! So we need a condition that can be checked more easily. The problem in the above example was that branching created a new boundary on the (lower-dimensional) solution set of the sub-constraint. We can avoid this as follows:

Theorem 4. *Branching a sub-constraint of the form $\exists \boldsymbol{y} \in B^{\boldsymbol{y}}\ \phi$ of an E-constraint with free-variable bound $B^{\boldsymbol{x}}$ into quantification bounds $B_1^{\boldsymbol{y}}$ and $B_2^{\boldsymbol{y}}$ is save if $(\exists \boldsymbol{y} \in B_1^{\boldsymbol{y}} \cap B_2^{\boldsymbol{y}}\ \phi, B^{\boldsymbol{x}})$ is false.*

Proof. Let $\boldsymbol{x_0} \in B^{\boldsymbol{x}}$ be arbitrary, but fixed. Assume that the input is stable at $\boldsymbol{x_0}$. We have two cases:

- The input is true at $\boldsymbol{x_0}$. This means that the degree of truth is positive at $\boldsymbol{x_0}$. If no path of positive degree of truth passes through the border $B_1^{\boldsymbol{y}} \cap B_2^{\boldsymbol{y}}$ then obviously the theorem holds.
  Now assume an arbitrary, but fixed path of positive degree of truth that passes through the border. This gives rise to a positive path in at least one resulting branch: Assume that the conjunction under the quantifier has the form $f = 0 \wedge g_1 \geq 0 \wedge \ldots \wedge g_n \geq 0$. Since the path has positive degree of truth, $g_1, \ldots, g_n$ are all positive on the path. This means that, on the border, f has to be non-zero, and so has opposite sign to one of the path end-points. Hence the branch containing this end-point has positive degree of truth, and therefore also the whole constraint.
- The input is false at $\boldsymbol{x_0}$: Then, by Lemma 3, it remains false.

□

Now we can use Theorem 4 in an algorithm for save branching. How can we check the necessary condition? In the one-dimensional case we just have to check a single point, and by Property 4 it suffices to call Prune with the quantification bound replaced by the new border $B_1 \cap B_2$. However, in the higher-dimensional case, this will not succeed in branching the quantification bound of stably false constraints, as the width of this quantification bound does not go to zero. So we have to decompose $B^{\boldsymbol{y}}$ into parts such that the overall size goes to zero, and call Prune on all the parts.

Algorithm 2 Branching

Input: C: a set of bounded E-constraints
  $(\phi, B^{\boldsymbol{x}}) \leftarrow$ an element of C with the free-variable bound of highest volume
  $\exists \boldsymbol{y} \in B^{\boldsymbol{y}}\ \phi' \leftarrow$ a sub-constraint of ϕ with quantification bound of highest volume
  $(B_1^{\boldsymbol{y}}, B_2^{\boldsymbol{y}}) \leftarrow$ bisection of $B^{\boldsymbol{y}}$ along the variable of maximal width
  $n \leftarrow \#(\exists \boldsymbol{y} \in B^{\boldsymbol{y}}\ \phi')^{\dim(B^{\boldsymbol{y}})-1}$
  $(B_1^{\boldsymbol{y}}, \ldots, B_n^{\boldsymbol{y}}) \leftarrow$ equal-sized decomposition of $B_1^{\boldsymbol{y}} \cap B_2^{\boldsymbol{y}}$ into n pieces
  if $w(B^y) > w(B^x)$ and for all $i \in \{1, \ldots, n\}$, Prune$(\exists \boldsymbol{y} \in B_i^{\boldsymbol{y}}\ \phi', B^{\boldsymbol{x}}) = \emptyset$ **then**
    return C with $\exists \boldsymbol{y} \in B^{\boldsymbol{y}}\ \phi'$ replaced by the result of branching $B^{\boldsymbol{y}}$
  else
    return C with $(\phi, B^{\boldsymbol{x}})$ replaced by the result of branching $B^{\boldsymbol{x}}$
  end if

The result is Algorithm 2. By $\#(\phi)$ we denote the number of times the algorithm already tried to branch the quantification bound of ϕ in earlier calls, but did not succeed (i.e., it took the else-branch of the if-statement). In the case of closed constraints, branching of the free-variable-bound will leave it unchanged.

By Theorem 4 the algorithm does save branching. Does it also decrease the width of the bounds appropriately? For the free-variable bounds this is the case:

Theorem 5. *By repeatedly applying Algorithm 2 to a set of E-constraints, interleaved with any operation that removes elements from the set or decreases the width of their bounds, the width of all free-variable bounds in the set goes to zero.*

Proof. We prove that it cannot happen that a quantification bound is branched infinitely often without branching the free-variable bound. Certainly this holds, because eventually $w(B^y) > w(B^x)$ does not hold and the else branch is taken. □

Also the quantification bounds are branched as necessary:

Theorem 6. *By repeatedly applying Algorithm 2 to a set of E-constraints, interleaved with any operation that removes elements from the set or decreases the width of their bounds, the width of all quantification bound of stably false constraints goes to zero.*

Proof. Algorithm 2 cannot produce an infinite sequence of branchings of the free-variable bound because the precision of the if-statement-test goes to infinity. Furthermore, it eventually branches each quantification bound, because of the maximal width choice. □

7 Checking

In this section we show how to do the checking step within Algorithm 1. For this we could build upon the rules provided in an earlier paper [27]. However, in order to increase the readability of this paper, we use the formalism developed there only indirectly: Obviously we can check a set of constraints by checking each of its elements. Furthermore we can check the disjunction of an E-constraint by just checking each of its branches.

For proving a sub-constraint of the form $\exists \boldsymbol{y} \in B^{\boldsymbol{y}}\ [f = 0\ \wedge\ g_1 \geq 0 \wedge \ldots \wedge g_k \geq 0]$ where the free variables range over a box $B^{\boldsymbol{x}}$ it suffices to find a subset $D \subseteq B^{\boldsymbol{y}}$ such that that all the inequality constraints hold on $B^{\boldsymbol{x}} \times D$ and the equality constraint has at least one solution in D for each element of the free-variable-bound $B^{\boldsymbol{x}}$ (then the rules of Theorem 1 of that paper [27] show that the whole constraint holds).

We can prove the existence of a solution to an equality $f = 0$ in a set D by finding an element of D for which f is non-negative and an element for which f is non-positive. For this we can use pruning: If $(f < 0, B)$ is pruned to the empty free-variable bound then the sign of f on B is non-negative, and if $(f > 0, B)$ is pruned to the empty free-variable bound then the sign of f on B is non-positive. In a similar way we can use pruning for proving inequalities.

We search for such a set D by computing the sign of f on the Cartesian product of $B^{\boldsymbol{x}}$ with—in each direction—$\#(\exists \boldsymbol{y} \in B^{\boldsymbol{y}}\ [f = 0\ \wedge\ g_1 \geq 0 \wedge \ldots \wedge g_k \geq 0]) + 2$ equally distributed sample points in $B^{\boldsymbol{y}}$ (including samples on the borders), but such that at least on each corner there is a sample (such as done in Figure 2 for $\#(\exists \boldsymbol{y} \in B^{\boldsymbol{y}}\ [f = 0\ \wedge\ g_1 \geq 0 \wedge \ldots \wedge g_k \geq 0]) + 2 = 3)$. For all this we use pruning with precision $\mathrm{prec}(B^{\boldsymbol{x}} \times B^{\boldsymbol{y}})$.

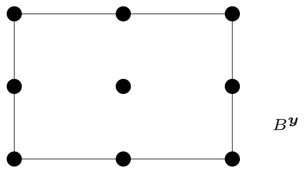


Fig. 2. Samples—coordinates corresponding to $B^{\boldsymbol{y}}$

Furthermore we prove $g_1 \geq 0, \ldots, g_n \geq 0$ on the Cartesian product of $B^{\boldsymbol{x}}$ with boxes that contain a sample point on each corner, and no sample points elsewhere. If we can connect a positive and a negative sample point on a path where $g_1 \geq 0, \ldots, g_n \geq 0$ holds (as in Figure 3), then the constraint is proven. This can be easily done by considering a graph whose vertices are the samples and which has edges between all neighboring samples between which we have proven $g_1 \geq 0, \ldots, g_n \geq 0$. We want to find out whether there is a path between a positive and a negative vertex.

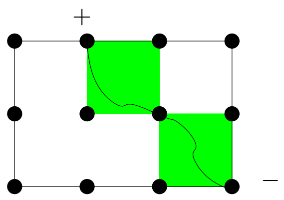


Fig. 3. Successful Check

It is trivial to formalize the above informal algorithm description. Now, using Property 2 of pruning and Bolzano's intermediate value theorem, one can easily prove:

Theorem 7. *Checking is correct.*

Furthermore a witness for this correctness is given by the samples on which f is positive and negative, respectively, and a set of boxes that connects these samples and on which $g_1, \ldots, g_n$ are non-negative.

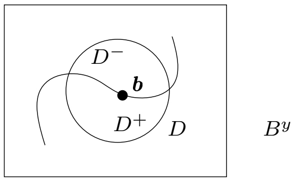
Checking is successful in the following sense:

Theorem 8. *For every sequence of bounded constraints of the form* $(\exists \boldsymbol{y} \in B^{\boldsymbol{y}}\ [f = 0\ \wedge\ g_1 \geq 0 \wedge \ldots \wedge g_k \geq 0], B^{\boldsymbol{x}})$ *that are stable and true for each element of the free-variable bound, and such that each element results from its predecessor as*

- *one branch of branching, and*
- *pruning*

checking eventually succeeds.

Proof. Every element of the sequence is stably true at each element of the free-variable-bound, and so by Theorems 2 and 3 also the degree of truth is positive at each element of the free-variable-bound. So there is an element $\boldsymbol{a}$ of the interior of



the free-variable bound of all sequence elements, an element $\boldsymbol{b}$ of the interior of the quantification bound of all sequence elements, and a neighborhood D of $\boldsymbol{b}$ such that:

- f is zero at $\boldsymbol{a} \times \boldsymbol{b}$,
- $g_1, \ldots, g_n$ are positive on $\boldsymbol{a} \times D$,
- there is an open subset D^+ of D of positive volume such that f is positive on $\boldsymbol{a} \times D^+$, and $\boldsymbol{b}$ is an element of the closure of D^+, and
- there is an open subset D^- of D of positive volume such that f is negative on $\boldsymbol{a} \times D^-$, and $\boldsymbol{b}$ is an element of the closure of D^-, and

Denote by $B_1^{\boldsymbol{x}}, \ldots$ the sequence of free-variable bounds, and by $B_1^{\boldsymbol{y}}, \ldots$ the sequence of quantification bounds. By Theorem 5, the width of the free-variable bound goes to zero ($\lim_{i \to \infty} w(B_i^{\boldsymbol{x}}) = 0$), the width of the quantification bound not necessarily. We prove that checking will eventually succeed.

First we prove the success of finding a sample point on which f is positive. Observe that $D^+ \cap B_i^{\boldsymbol{y}}$ always has positive volume since $\boldsymbol{b}$ is in the interior of $B_i^{\boldsymbol{y}}$ and is an element of the closure of D^+. So checking will eventually try infinitely many samples within this set. It remains to be proven that pruning eventually succeeds on one of them. Here we have two cases:

- The intersection of the border of $B_i^{\boldsymbol{y}}$ with D^+ eventually has positive volume on the border, and so we will eventually check a sample, and by Property 5, since this border has been created by pruning, pruning will be able to compute the sign at the sample using the current precision.
- Otherwise, for all i, D^+ has not more than singular intersection with the border of $B_i^{\boldsymbol{y}}$: In this case, we can construct a sequence $D_1^{\boldsymbol{y}}, \ldots$ of sub-boxes of the sequence of quantification bounds such that this sequence converges to an element of D^+. By Property 4, pruning with precision going to infinity eventually succeeds on the elementwise Cartesian product of that sequence with the corresponding free-variable bounds. Since the samples are equally distributed, for every element of the sequence $D_1^{\boldsymbol{y}}, \ldots$ eventually a sample with higher precision will be created, and by Property 3 pruning will succeed on the Cartesian product of this sample with the corresponding free-variable bound.

In a similar way we can prove the success of finding a sample point on which f is negative.

These sample points are all within D, on which $g_1, \ldots, g_n$ are all positive. So, by Property 4, the positivity check will also eventually succeed. $\square$

8 Application to Overall Algorithm

Now we can use the results developed in the previous three sections in the main algorithm. As a result we get:

Theorem 9. *Using pruning from Section 5, branching from Section 6, and checking from Section 7, Algorithm 1 terminates for inputs for which the volume of the elements of B for which ϕ is not stable is zero.*

Proof.

- For bounded constraints converging to a point with positive degree of truth, by Theorem 8, eventually the check succeeds.
- For bounded constraints converging to a point with negative degree of truth eventually pruning succeeds: We have to prove that for all sequences of bounded constraints converging to a point with negative degree of truth, where one element results from the previous by pruning and branching (taking one element of the produced branches), eventually the free-variable-bound is the empty set. Let $\exists \boldsymbol{y} \in B\ [f = 0 \ \wedge \ g_1 \geq 0 \wedge \ldots \wedge g_n \geq 0]$ be an arbitrary but fixed disjunctive branch of the constraint under consideration. By Lemma 2, its degree of truth is given by $\sup_{\boldsymbol{y} \in B} \min\{f, -f, g_1, \ldots, g_n\}$. Let h be the element of $\{f, -f, g_1, \ldots, g_n\}$ on which the (negative) optimum is attained.
  Now, by Theorem 6 the width of the quantification bound goes to zero, and so by Property 4, Prune will eventually result in the empty set for h, disproving the constraint.

$\square$

9 Implementation and Applications

We have implemented a prototype of the algorithm in the programming language O'Caml (www.ocaml.org). We do not use more precision than available with machine-precision floating-point numbers. This suffices for all our examples. For pruning we use an extremely simple (and thus usually inefficient) algorithm for computing hull consistency based on the interval library smath [15, 14].

As there are no existing algorithms/implementations for the general problem studied in this paper, instead of a comparison, we illustrate the usefulness of our approach by discussing the application of the resulting solver to simple examples from control engineering, parameter estimation, and computational geometry.

In the first example, we consider a problem in control engineering [23, 11]. Here one important tool to describe the behavior of a system is its *characteristic polynomial*. In the design process this polynomial is parametric, and the goal is to find values for these parameters such that the resulting polynomial has certain properties. For example, often one requires that the polynomial has a real root in a certain interval.

Now consider the characteristic polynomial $s(s + 1.71)(s + 100) + 6.63K = 0$ of the positioning system of a radio telescope antenna [23, p. 17ff, p. 295ff]. Here the parameter K denotes the gain of a certain amplifier. It needs to be set in such a way that the total system fulfill certain characteristics. We already know that, for the system to be stable, $0 < K < 2623$. Now we also want that the characteristic polynomial has a real root in $[-2.0, -0.5]$. This results in the constraint

$$\exists s \in [-2.0, -0.5] \ \ s(s + 1.71)(s + 100) + 6.63K = 0$$

which the solver proves to be true for $K \in [0, 9.375]$ and false for $K > 11.25$. Very often, the design goal imposes additional constraints to the system. So we add the additional constraint $se^K \leq -1$. The solver reports the same solution as before.

The second example comes from the field of parameter estimation [34]. Here one has given a model of a system and some information coming from measurements on some of these variables. The goal is to deduce further information on the possible variable values.

We consider the following parameter estimation problem: Given a system model whose explicit solution (obtained from a differential equation) is $f(p_1, p_2, t) = 20e^{-p_1 t} - 8e^{-p_2 t}$ [18, 17, 21]. We have the information that f reaches zero some-when, but we only know that this happens between time 2 and 4. We want more information about the parameters p_1 and p_2. This results in the constraint

$$\exists t \in [2, 4] \ \ 20e^{-p_1 t} - 8e^{-p_2 t} = 0$$

for which the implementation computes a strip of values for p_1 and p_2 as solutions.

The third example comes from computational geometry. Here, very often the situation arises that one wants to visualize high-dimensional non-linear objects. One method for doing this, is to project them into 2-dimensional space, resulting in constraints like

$$\exists z \in [-2, 2] \ \ \left[x^2 + y^2 + z^2 - 1 = 0 \wedge x^2 + y^2 - 0.5 \geq 0\right]$$

to which we applied our solver, resulting in a 2-dimensional ring.

The run-times in all the above examples (for the precision 0.1) were under one second on an average Linux PC. However, for some bigger examples they increase rapidly. The main reason for this is that it often computes lots of sample points although pruning did not yet succeed in isolating a solution sufficiently enough.

10 Related Work

The main alternative to modeling uncertain variables by bounded quantification is stochastic [33]. For quantified constraints, the problem has several special cases that already could be solved:

- In the case without equality constraints, the atomic sub-constraints in general have solution sets with volume. Therefore, one can compute true elements by pruning the negation of the original constraints [29, 28, 2].
- In the case $k = 0$, that is, constraints of the form $\exists \boldsymbol{y} \in B \ f = 0$, according to Bolzano's intermediate value theorem, one can reduce the problem to the previous case by reformulating it to $\exists \boldsymbol{y} \ f \geq 0 \ \wedge \ \exists \boldsymbol{y} \ f \leq 0$. However, this method does not generalize to E-constraints, because here the solution set of the inequalities should connect the positive and negative elements of f.
- In the case $\exists y \in B_y \ [f = y \wedge g_1 \geq 0 \wedge \ldots \wedge g_n \geq 0]$, where f does not contain y, the equality holds a priori. One just has to make sure that it is contained in B_y and that it fulfills the inequalities, which can be checked by the usual interval methods. Note that in some cases one can isolate the variable y in such a way, but in general not! The case where some additional existentially quantified variables (but not the isolated one!) occur in f can be treated by additional splitting [17, p. 156].

– The case where all terms are polynomials, which is a classical research topic in computer algebra [32, 8, 16, 19].
– The case where the quantified variables fulfill certain structural restrictions (e.g., only occurring once) [31, 12, 36], or where primitive pruning operations suffice for solving [5].

Existence proofs in interval analysis usually are done using variants of the interval Newton method. This fails for zero that are well-posed, but not simple (e.g., the zero of $x^3 = 0$), and for very close zero clusters. These methods do not avoid splitting on solutions. Instead they usually use a method called ε-inflation which seems to succeed for simple zeros in practice, but whose general success is proven only for special cases [20, 30]. As shown by Neumaier [22, Chapter 5] one can alternatively construct a super-box of a presumed zero for which existence holds.

In general, existence proofs for non-simple, but still well-posed, zeros need techniques that do not rely on the derivative. An promising notion here is the topological degree [24, Chapter 6].

11 Conclusion

In this paper we have introduced the first known algorithm for solving a certain type of quantified constraint over the real numbers, and we have applied the algorithm to several application areas. In future work we will consider the following improvements of the method:

– heuristics for the number of computed samples, and the choice of branching,
– making the checking step incremental, by reusing the information computed in earlier steps, and
– replacing the checking operation by a dual pruning operation that removes elements from the free-variable-bound that provably belong to the solution set.

Our final goal is to be able to efficiently solve general, well-posed quantified constraints.

Thanks to Laurent Granvilliers for interesting discussions on the topic and to Varadarajulu Reddy Pyda for help with the implementation of the solver.

References

1. F. Benhamou. Heterogeneous constraint solving. In *Proc. of the Fifth International Conference on Algebraic and Logic Programming*, 1996.
2. F. Benhamou and F. Goualard. Universally quantified interval constraints. In *Proc. of the Sixth Intl. Conf. on Principles and Practice of Constraint Programming (CP'2000)*, number 1894 in LNCS, Singapore, 2000. Springer Verlag.
3. F. Benhamou, D. McAllester, and P. V. Hentenryck. CLP(Intervals) Revisited. In *International Symposium on Logic Programming*, pages 124–138, Ithaca, NY, USA, 1994. MIT Press.
4. F. Benhamou and W. J. Older. Applying interval arithmetic to real, integer and Boolean constraints. *Journal of Logic Programming*, 32(1):1–24, 1997.
5. L. Bordeaux and E. Monfroy. Beyond NP: Arc-consistency for quantified constraints. In P. V. Hentenryck, editor, *Proc. of Principles and Practice of Constraint Programming (CP 2002)*, number 2470 in LNCS. Springer, 2002.
6. B. F. Caviness and J. R. Johnson, editors. *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Springer, Wien, 1998.
7. J. G. Cleary. Logical arithmetic. *Future Computing Systems*, 2(2):125–149, 1987.
8. G. E. Collins. Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition. In Caviness and Johnson [6], pages 134–183.

9. J. H. Davenport and J. Heintz. Real quantifier elimination is doubly exponential. *Journal of Symbolic Computation*, 5:29–35, 1988.
10. E. Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32(3):281–331, 1987.
11. R. C. Dorf and R. M. Bishop. *Modern Control Systems*. Addison-Wesley, Reading, Massachusetts, 1995.
12. E. Gardeñes, M. Á. Sainz, L. Jorba, R. Calm, R. Estela, H. Mielgo, and A. Trepat. Modal intervals. *Reliable Computing*, 7(2):77–111, 2001.
13. L. Granvilliers. On the combination of interval constraint solvers. *Reliable Computing*, 7(6):467–483, 2001.
14. T. J. Hickey. smathlib. http://interval.sourceforge.net/interval/C/smathlib/README.html.
15. T. J. Hickey, Q. Ju, and M. H. van Emden. Interval arithmetic: from principles to implementation. *Journal of the ACM*, 48(5):1038–1068, 2001.
16. H. Hong. *Improvements in CAD-based Quantifier Elimination*. PhD thesis, The Ohio State University, 1990.
17. L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis, with Examples in Parameter and State Estimation, Robust Control and Robotics*. Springer, Berlin, 2001.
18. L. Jaulin and E. Walter. Guaranteed nonlinear parameter estimation from bounded-error data via interval analysis. *Mathematics and Computers in Simulation*, 35(2):123–137, 1993.
19. R. Loos and V. Weispfenning. Applying linear quantifier elimination. *The Computer Journal*, 36(5):450–462, 1993.
20. G. Mayer. Epsilon-inflation in verification algorithms. *Journal of Computational and Applied Mathematics*, 60:147–169, 1994.
21. M. Milanese and A. Vicino. Estimation theory for nonlinear models and set membership uncertainty. *Automatica (Journal of IFAC)*, 27(2):403–408, 1991.
22. A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge Univ. Press, Cambridge, 1990.
23. N. S. Nise. *Control Systems Engineering*. John Wiley & Sons, 3rd edition, 2000.
24. J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations*. Academic Press, 1970.
25. S. Ratschan. Applications of quantified constraint solving over the reals— bibliography. http://www.mpi-sb.mpg.de/ ratschan/appqcs.html, 2001.
26. S. Ratschan. Continuous first-order constraint satisfaction. In J. Calmet, B. Benhamou, O. Caprotti, L. Henocque, and V. Sorge, editors, *Artificial Intelligence, Automated Reasoning, and Symbolic Computation*, number 2385 in LNCS, pages 181–195. Springer, 2002.
27. S. Ratschan. Continuous first-order constraint satisfaction with equality and disequality constraints. In P. van Hentenryck, editor, *Proc. 8th International Conference on Principles and Practice of Constraint Programming*, number 2470 in LNCS, pages 680–685. Springer, 2002.
28. S. Ratschan. Efficient solving of quantified inequality constraints over the real numbers. http://www.mpi-sb.mpg.de/ ratschan/preprints.html, 2002. submitted for publication.
29. S. Ratschan. Quantified constraints under perturbations. *Journal of Symbolic Computation*, 33(4):493–505, 2002.
30. S. M. Rump. A note on epsilon-inflation. *Reliable Computing*, 4:371–375, 1998.
31. S. P. Shary. A new technique in systems analysis under interval uncertainty and ambiguity. *Reliable Computing*, 8:321–418, 2002.
32. A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. Univ. of California Press, Berkeley, 1951. Also in [6].
33. T. Walsh. Stochastic constraint programming. In *Proc. of ECAI*, 2002.
34. E. Walter and L. Pronzato. *Identification of Parametric Models from Experimental Data*. Springer, 1997.
35. V. Weispfenning. The complexity of linear problems in fields. *Journal of Symbolic Computation*, 5(1–2):3–27, 1988.
36. N. Yorke-Smith and C. Gervet. On constraint problems with incomplete or erroneous data. In P. V. Hentenryck, editor, *Principles and Practice of Constraint Programming*, number 2470 in LNCS, pages 732–737, 2002.