Convergence of Quantified Constraint Solving by Approximate Quantifiers

Stefan Ratschan*

September 6, 2000

Abstract

Exactly solving quantified constraints (i.e., first-order formulas over the real numbers) is in general undecidable, and for special theories highly complex. One method to deal with this, is to compute approximate solutions instead of exact ones. In an earlier paper we presented a method for computing such approximate solutions. This method proceeded by propagating approximate solution sets over the structure of constraints. In this paper we show that the quantifiers of the first-order predicate language are an obstacle to allowing approximations to arbitrary small error bounds. We solve the problem by modifying the first-order language and replacing the classical quantifiers with approximate quantifiers. These also have two additional advantages: First, they are tunable, in the sense that they allow the user to decide on the trade-off between precision and efficiency. Second, they introduce additional expressivity into the first-order language by allowing reasoning over the size of solution sets.

1 Introduction

This paper deals with solving quantified real constraints, that is, formulas in the first-order predicate language over the real numbers. Solving such constraints usually means to find a representation of their solution set that is in some sense simple. For example, the method of quantifier elimination takes a first-order constraint and returns an equivalent, but quantifier-free constraint. Such quantifier elimination methods exist for constraints containing only the predicate symbols $=$, $\leq$ and $<$, the function symbols $+$ and $\cdot$, and rational constants (i.e., the theory of real-closed fields [38]). But as soon as one adds transcendental functions to the theory, the problem becomes

*This work was supported by the Austrian Science Fund FWF in the frame of the project SFB F1303

undecidable. Even when restricting oneselves to the theory of real-closed fields, quantifier elimination is usually extremely complex and complicated.

To deal with this problem, one can either restrict oneselves to even more special problem classes (see e.g. [43, 21, 15]), or relax the problem by allowing approximation up to a user-specified error bound (as proposed by Hong [22, 23]). This paper follows the second approach. Its main contributations are: To show that this relaxed specification is still not usable for approximate computation because the classical quantificiers are an obstacle to attaining an arbitrary small positive error bound; and to introduce a modification of the first-order language — approximate quantifiers — that solves this problem.

These quantifiers have two additional advantages: First, they are tunable, in the sense that they allow the user to decide on the trade-off between precision and efficiency. Second, they introduce additional expressivity into the first-order predicate language, by allowing reasoning over the size of solution sets.

The first step to introduce approximate quantifiers is to allow quantifiers with a positive real annotation q, with the intuitive meaning that a formula $(\exists_q x)\phi(x)$ is true iff the volume of the solution set of ϕ is greater than q. We will see that this does not yet allow the computation of approximate solution sets up to arbitrarily small error bounds.

So we allow quantifiers to be annotated with a real interval $[\underline{q}, \overline{q}]$, with the intuitive meaning that the exact annotation can be any element of $[\underline{q}, \overline{q}]$. This allows an algorithm to choose the most suitable value in $[\underline{q}, \overline{q}]$ and we *don't care* which one. This means that a sentence in the language does not have one distinct truth value but has a set of possible truth values (cmp. with [18] or [20]). We will prove that, from a good enough approximation, we can always compute at least one of them, and thus one can always attain an arbitrarily small error bound for approximate solution sets.

Now the problem arises that sometimes one cannot assign a volume to every solution set, because it might be unmeasurable (see any book on measure theory [16, 5] for details). Furthermore it might be interesting to deliberately use less powerful measures, for example a floating point box measure that models the sets that can be measured by floating point boxes on a computer. Every unmeasurable set has several possible volumes between the inner and outer measure, but we *don't know* which one.

Following the usual approach (see e.g. [42]), one would implement such a logic using sets of truth values (representing a many-valued logic [39, 34]) instead of single truth values. We show that this approach is not suitable here and present a new method that is completely orthogonal to the semantics usually given to formulas when we *don't know* the value of certain predicate and function symbols and thus assign validity to the formula using all possible predicate and function symbol assignments (i.e., interpretations). Although arising from problems over real numbers, the resulting logic is completely domain-independent.

The above situation that there are several possible values for an object, and we either do not know or do not care which one should be taken, is commonly called *don't know* and *don't care nondeterminism*, respectively. Here both forms occur at the same time, which creates various difficulties through their interaction. Our approach gives general insight into such a situation by showing how one can compute with such nondeterministic objects, nevertheless. So our language can be easily extended to take into account nondeterminism coming from other sources. For example uncertain coefficients of occurring polynomials can be either modeled as don't care nondeterminism (the *united* approach) or as don't know nondeterminism (the *robust* approach) (cmp. with [6, 13, 36]).

The structure of the paper is as follows: In section 2, we give the specification of solving quantified constraints approximately up to some specified error bound, and informally show that this is impossible for constraints containing classical quantifiers. In section 3, we develop the necessary formal tools for approximate constraint solving with classical quantification. In section 4, we give a first-order language where the classical quantifiers are replaced by approximate ones. In section 5, we adapt the formal tools for approximate constraint solving to approximate quantification. In section 6, we show how to deal with the resulting two forms of nondetermism. In section 7, we prove that one can solve constraints that contain approximate instead of classical quantifiers up to an arbitrary small error bound. In section 8, we discuss related work, and in section 9, we give a final conclusion.

2 Approximate Quantified Constraint Solving

Quantified constraints are first-order formulas over the real numbers. We want to solve them: Given some constraint, we want to get a simple representation of its solution set. However, since we cannot represent arbitrary real numbers on computers, we also cannot represent arbitrary solution sets. Methods for solving quantified real constraints exactly exist only for special cases (e.g., for the theory of real-closed fields, which allows only the function symbols $+$ and $.$ [9, 38]). But even in these special cases the methods usually have huge complexity [10], and often create output that is too complicated for practical use.

So we can only hope to tackle the general problem, if we relax it. As proposed by Hoon Hong [22, 23], we do this by allowing approximation. This results in the problem specification of figure 1.

How can we find an algorithm that meets this specification? Let us examine the example of a constraint of the form $L(\phi_1, \ldots, \phi_n)$, where L is either a quantifier (in this case $n = 1$) or a connective. Then the solution sets of the sub-constraints $\phi_1, \ldots, \phi_n$, determine the solution set of the whole

GIVEN:	A constraint ϕ, and a positive real error bound ε
FIND:	Sets Y and N, such that all elements of Y are in the solution set of ϕ, all elements of N are out of the solution set of ϕ, and the volume of the elements not in Y or N is smaller than ε

Figure 1: Algorithm Specification

constraint. In a similar way, approximations of the solution sets of the atomic sub-constraints determine an approximation of the solution set of the whole constraint. So, by recursive application of this argument over the structure of constraints, we can compute approximations of solution sets of constraints from approximations of the solution sets of their atomic sub-constraints (see [32] for details).

Now assume that we can compute approximations of the solution sets of atomic constraints up to an arbitrarily small positive error bound. The question remains, whether we can use this to compute an approximation of the solution set of arbitrary constraints up to an arbitrarily small positive error bound. This is not the case for constraints containing quantifiers. We show this here informally and formalize the arguments in the next section. We do all our considerations for existential quantification only. They can be easily adapted to the dual case of universal quantification.

Take an input constraint of the form $(\exists x)\phi(x)$ without free variables, where $\phi(x)$ is an atomic constraint with an empty solution set. Determine for all x, except for a set of arbitrarily small but positive volume, whether x is in the solution set of $\phi(x)$. From this information we cannot deduce that $(\exists x)\phi(x)$ is false, because some of the remaining x might be in the solution set of $\phi(x)$, in which case $(\exists x)\phi(x)$ would be true.

One could suspect that the reason for this problem is, that a classical quantifier has to take into account arbitrarily small solution sets of the quantified sub-constraint. So we introduce an additional quantifier $\exists_q$ into the first-order predicate language, where q is a non-negative real number. For a constraint ϕ in n free variables this *existential volume quantifier* is true iff the volume of the x, for which $\phi(x)$ is true, is greater than q. But even for this quantifier, we have the same problem: Take an input constraint of the form $(\exists_q x)\phi(x)$. Assume that the solution set of ϕ has exactly the volume q. Determine for all x, except for a set of arbitrarily small but positive volume, whether x is in the solution set of $\phi(x)$. Again, from this information we cannot deduce the truth-value of $(\exists_q x)\phi(x)$. Note that one can easily find examples of constraints *with* free variables, that show the same behaviour; in this case we can deduce for no element of the free-variable space whether it is an element of the solution set.

Speaking in the language of numerical analysis, the problem of finding approximations of solution sets of quantified constraints is *ill-posed* for certain inputs. In these cases small perturbations of the input result in arbitrarily big changes of the output. We can only work on such inputs if we *regularize* the problem [11]. We do this here by introducing different quantifiers, for which the problem does not occur.

3 Approximate Constraint Solving and Classical Quantification

Before solving the problems introduced in the last section, we formalize them. Readers who want to see the solution immediately, without being interested in formal proofs, can skip this section and can directly jump to section 4.

Observe that approximate information about a solution set means, that there are several possible solution sets, and we do not know which one is correct. We represent this don't know nondeterminism by using sets (analogously to interval arithmetic [28, 2, 33, 30, 17, 31, 37]).

Definition 1 *For any constraint ϕ in n free variables, $\hat{d} \subseteq 2^{\mathbb{R}^n}$ is an* approximate solution set *of ϕ iff it contains the solution set of ϕ.*

Of course an approximate solution set with just one element corresponds to an exact solution set. Note that constraints without free variables have 0-dimensional solution sets. In this case the set containing the empty tuple represents truth (i.e., the truth value $\mathbf{T}$) and the empty set represents falsehood (i.e., the truth value $\mathbf{F}$). From now on we call subsets of $\mathbb{R}^n$ n-dimensional solution sets, and denote them by $\mathbb{S}_n$. We call subsets of $\mathbb{S}_n$ n-dimensional approximate solution sets and denote them by $\hat{\mathbb{S}}_n$. Observe that it can also occur that there are several possible functions, and we do not know which one is correct. Again we represent this don't know nondetermism by a set of functions, which we call *approximate function*.

We can deduce the information we want to find for our specification (figure 1) as follows:

Definition 2 *An $x \in \mathbb{R}^n$ is* determined to true *by an n-dimensional approximate solution set $\hat{d}$ iff for all $d \in \hat{d}$, $x \in d$. It is* determined to false *by $\hat{d}$ iff for all $d \in \hat{d}$, $x \notin d$. It is* determined *by $\hat{d}$ iff it is either determined to true or determined to false by $\hat{d}$.*

Elements for which we cannot deduce such information, constitute the error:

Definition 3 *For an n-dimensional approximate solution set $\hat{d}$, the* error *of $\hat{d}$ is the volume of the set of all $x \in \mathbb{R}^n$ for which x is not determined by $\hat{d}$.*

We adopt the convention that in $\mathbb{R}^0$ the volume of the empty set is zero and the volume of the set containing the empty tuple is infinite.

We have seen that for a constraint of the form $L(\phi_1, \ldots, \phi_n)$, where L is either a quantifier (in this case $n = 1$) or a connective, the solution set of the total constraint is a function of the solution sets of the sub-constraints. In the following we restrict ourselves to the case where L is an existential quantifier. In this case we have the following function on solution sets (see [32] for the other cases).

Definition 4 *For a nonnegative integer n, the* n-dimensional projection operator *is a function $P : \mathbb{S}_{n+1} \rightarrow \mathbb{S}_n$ such that $(x_1, \ldots, x_n) \in P(d)$ iff there is a y s.t. $(x_1, \ldots, x_n, y) \in d$.*

An approximate function determines the following function on approximate solution sets:

Definition 5 *For an approximate function $\hat{f} \subseteq \mathbb{S}_k \rightarrow \mathbb{S}_l$, for an approximate solution set $\hat{d}$,*

$$Ext_{\hat{f}}(\hat{d}) := \{f(d) | f \in \hat{f}, d \in \hat{d}\}$$

We call $Ext_{\hat{f}}$ the extension of $\hat{f}$.

So, for a constraint $(\exists x)\phi$, we can compute an approximate solution set of the total constraint from an approximate solution set of ϕ, by applying $\mathrm{Ext}_{\{P\}}$, where P is the projection operator, to the approximate solution set of ϕ.

Now we formalize the property needed for a function on approximate solution sets to be useable for computing approximate information up to arbitrary small error.

Definition 6 *A function $\hat{f} : \hat{\mathbb{S}}_k \rightarrow \hat{\mathbb{S}}_l$ is* contracting *iff for all $\varepsilon \in \mathbb{R}^+$ there is an $\varepsilon' \in \mathbb{R}^+$ such that for all $\hat{d} \in \hat{\mathbb{S}}_k$ such that the error of $\hat{d}$ is less or equal ε', the error of $\hat{f}(\hat{d})$ is less or equal ε.*

Using the argument from section 2 it is easy to show that the extension of the approximate function that contains the projection operator is not contracting. Thus we cannot fulfill our specification (figure 1) for constraints containing quantifiers. So we have to find a similar, but easier problem for which we can fulfill the specification. We do this by introducing quantifiers that approximate classical quantification, but result in a contracting projection operator.

4 First-Order Predicate Logic with Approximate Quantifiers

Since we do not confine ourselves to the domain of real numbers, in this section we speak of formulas and sentences instead of constraints. Before we go into details, recall that the notion of volume is modeled in mathematics by measure spaces (see any textbook on measure theory, for example [16] or [5], for details). For any measure v and set A, the inner measure $\underline{v}(A)$ is the supremum of the measures of all measurable subsets of A, and the outer measure $\overline{v}(A)$ is the infimum of the measures of all measurable supersets of A. For any set A, $\underline{v}(A) \leq \overline{v}(A)$, and if A is measurable, then $v(A) = \underline{v}(A) = \overline{v}(A)$. Furthermore, we call a function v *possible measure* iff for any set A, $v(A) \in [\underline{v}(A), \overline{v}(A)]$.

Existential quantifiers and existential volume quantifiers do not allow approximation up to an arbitrarily small error bound because they discontinuously change from false to true. If the size of the solution set of the quantified constraint is exactly at this point of discontinuous change, then we cannot decide between true and false using approximate computation.

We avoid this discontinuous change by using existential volume quantifiers $\exists_q$ for which we do not specify the annotation q exactly, but only require it to be within a certain interval $[\underline{q}, \overline{q}]$. Then a constraint solver can choose an element of this interval element for which it can safely decide on the resulting truth value using the available approximate information. This means that for sentences containing such an approximate quantifier both true and false are possible truth values, if the size of a quantified set is within the interval $[\underline{q}, \overline{q}]$ (see figure 2).

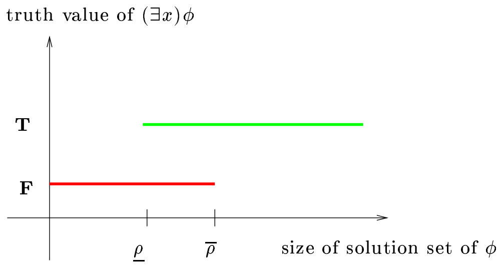

Figure 2: Approximate Quantifiers

Recall that an approximate solution set consists of several possible solution sets, and we do not know which of them is the correct one — we have don't know nondeterminism. Here we are in exactly the dual situation: We allow several equally possible truth values, and we do not care, which of them is chosen — thus we have don't care nondeterminism.

Since our quantifiers depend on the size of solution sets, we also have to deal with the situation when such a solution set is not measurable. Fortunately, this again is don't know nondeterminism. If a solution set is not measurable, then we only know that its volume is between the inner and the outer measure, but we do not know which of these.

A naive approach to modeling such a situation, where a logical formula can have more than one truth-value, would propagate sets of truth values by applying the logical symbols element-wise. For example, if in a formula $\phi_1 \wedge \phi_2$ the truth-value set of ϕ_1 is $\{\mathbf{T}\}$ and the truth-value set of ϕ_2 is $\{\mathbf{T}, \mathbf{F}\}$ then the combination of all these elements yields a truth-value set $\{\mathbf{T}, \mathbf{F}\}$ for the whole formula. This results in a many-valued logic [39, 34]. In order to show that this approach is not feasible, we first demonstrate, that this would need a more complicated many-valued logic, and then give a reason, why to avoid many-valued logics altogether.

One would need a more complicated many-valued logic, because of the need to consider the interaction between the don't care nondeterminism resulting from the approximate quantifiers and the don't know nondeterminism resulting from unmeasurable sets. Approximate quantifiers can result in a truth-value set $\{\mathbf{T}, \mathbf{F}\}$. An unmeasurable solution set can result in an unknown truth value of a formula. Typically one would model this by an empty truth-value set. But one wants to be able to assign the set $\{\mathbf{F}\}$ to the whole formula $\phi_1 \wedge \phi_2$ if ϕ_1 has the truth-value set $\{\mathbf{F}\}$ and ϕ_2 has the empty truth-value set, although element-wise combination yields the empty set in this case. This problem arises because a non-existing truth value for ϕ_2 means that it can have two possible truth values. We don't know which one, but we have modeled only the don't care form of nondeterminism.

One could construct a many-valued logic that solves this problem (see section 6), but there is another problem, that makes us avoid such an approach altogether: Many-valued logics are not truth-functional [39, 34]: For example, consider $\phi \wedge \neg\phi$, where ϕ has the truth-value set $\{\mathbf{T}, \mathbf{F}\}$. Then element-wise combination yields the truth-value set $\{\mathbf{T}, \mathbf{F}\}$, although we want this formula to be false in any case. The reason is, that a many-valued logic forgets the information about the equality of ϕ in both branches of $\wedge$. This also makes it impossible to define $\leftrightarrow$ (equivalence) as an abbreviation. A similar problem also occurs in interval mathematics [28, 2, 33, 30, 17, 31, 37], where the information about equal terms is lost.

Before going into the details of our solution we fix the syntax. It is the usual one of the first-order predicate language (using the connectives $\neg$, $\vee$ and $\wedge$), with the only exception that instead of classical quantifiers, approximate quantifiers are used. Consider the example $(\forall^1_{[0.1,0.2]} x)(\exists^2_{[0.0,0.1]} y)[\ x > 0 \wedge y = 0\ ]$. The quantifiers have a subscript consisting of a nonnegative real interval (the *annotation*). Furthermore they have a positive integer superscript (the *tag*). Tag equality indicates equal (nondeterministic) behavior of the according quantifiers. This will allow us to make $A \wedge \neg A$ false in any case, and makes the definition of $A \leftrightarrow B$ as an abbreviation for $(A \vee \neg B) \wedge (\neg A \vee B)$ possible. We require that, within a formula, quantifiers that have the same tag, also have the same annotation.

If the annotation of a quantifier is a one-elementary interval then we say that the quantifier is deterministic. A formula where all the quantifier tags of quantifiers that are not deterministic, are different, is called *free*. Often we do not explicitly write down the tags but assume an arbitrary tagging such that a formula is free.

For assigning semantics to such formulas, we need some analogies to the usual definitions of the first-order predicate language:

Definition 7 *A* structure *consists of a measure space $\mathcal{D}$, and functions and relations on $\mathcal{D}$. $\mathcal{D}$ is called the* domain *of the structure.*

Definition 8 *An interpretation $\mathcal{I}$ in a structure $\mathcal{S}$ assigns to every n-ary predicate symbol an n-ary relation in $\mathcal{S}$, and to every n-ary function symbol an n-ary function in $\mathcal{S}$.*

Terms can be interpreted as usual in structures. For fixing the semantics of approximate quantification we use a method that is completely orthogonal to the semantics usually given to formulas when we *don't know* the value of certain predicate and function symbols and thus assign validity to the formula using all possible predicate and function symbol assignments (i.e., interpretations).

It is straightforward to give semantics to a formula if we already know for each occuring quantifier with annotation $[\underline{q}, \overline{q}]$, which volume quantifier $\exists_q$ or $\forall_q$, where $q \in [\underline{q}, \overline{q}]$, and which possible measure to use for determining the volume of solution sets. So, in analogy to the notion of interpretation, which assigns unknown information to predicate and function symbols, we assign information to quantifiers as follows:

Definition 9 *Given a formula ϕ, a function $q^* : \mathbb{N} \to \mathbb{R}$ is a* quantifier choice for *ϕ iff for every $t \in \mathbb{N}$ that occurs as a quantifier tag in ϕ, $q^*(t) \in [\underline{q}, \overline{q}]$, where $[\underline{q}, \overline{q}]$ is the annotation of the quantifiers occurring in ϕ that are tagged by t.*

Definition 10 *Given a structure $\mathcal{S}$ containing a domain $\mathcal{D}$ with measure v, v^* is a* measure guess in *$\mathcal{S}$ iff*

$$v^* : \mathbb{N} \to (\bigcup_{n \geq 1} 2^{\mathcal{D}^n} \to [0, \infty]),$$

such that for all $t \in \mathbb{N}$, $v^*(t)$ is a possible measure.

It is an easy exercise to define the truth value of a sentence and solution set of a formula, given an interpretation $\mathcal{I}$ in a structure $\mathcal{S}$, a quantifier choice q^* for ϕ, and a measure-guess v^* in $\mathcal{S}$. Now we model that we *don't care* for the quantifier choices and *don't know* about the right measure guesses.

Definition 11 *Given a sentence ϕ, a structure $\mathcal{S}$, and an interpretation $\mathcal{I}$ in $\mathcal{S}$, t is a* truth value *of ϕ under $\mathcal{I}$ in $\mathcal{S}$ iff there is a quantifier-choice q^* for ϕ such that for all measure-guesses v^* in $\mathcal{S}$ the truth value of ϕ under $\mathcal{S}, \mathcal{I}, q^*, v^*$ is t.*

As an example assume a sentence $(\exists_{[\underline{q},\overline{q}]} x)\phi(x)$, where $\phi(x)$ has a measurable solution set d. If the volume of d is less or equal $\underline{q}$, then only **F** is a truth value of $(\exists_{[\underline{q},\overline{q}]} x)\phi(x)$; if the volume of d is greater than $\overline{q}$, then only **T** is a truth value of $(\exists_{[\underline{q},\overline{q}]} x)\phi(x)$; but if the volume of d is within $[\underline{q}, \overline{q}]$, then there is a quantifier choice for which $(\exists_{[\underline{q},\overline{q}]} x)\phi(x)$ is **F** and also a quantifier choice for which it is **T**. So we have exactly the behaviour sketched in figure 2.

Since we are interested in solving formulas instead of only sentences we generalize definition 11 as follows:

Definition 12 *Given a formula ϕ, a structure $\mathcal{S}$, and an interpretation $\mathcal{I}$ in $\mathcal{S}$, x is* in (out) of a solution set *of ϕ under $\mathcal{I}$ in $\mathcal{S}$ iff there is a quantifier-choice q^* for ϕ such that for all measure-guesses v^* in $\mathcal{S}$, x is in (out) of the solution set of ϕ under $\mathcal{S}, \mathcal{I}, q^*, v^*$.*

This is exactly the information we ask for in our specification (figure 1).

5 Approximate Constraint Solving and Approximate Quantification

The question remains, whether we can compute approximate solution sets of formulas that contain approximate instead of classical quantifiers up to an arbitrarily small error bound. For this we go back to (real) constraint solving by fixing an according structure $\mathcal{S}$ and interpretation $\mathcal{I}$, and adapt the development of section 3 to approximate quantification.

There we represented the uncertainty about a solution set by the notion of approximate solution set, which is a set of possible solution sets. But in addition to this non-determinism, the semantics of our language, as given in definition 12, already takes into account the non-determinism resulting from approximate quantifiers and unmeasurable sets. In order to deal with all these appearances of non-determinism, we have to combine them within one definition. We do this by again starting from the definition of solution set under quantifier choice and measure guess, this time taking into account approximation.

Definition 13 *Given a constraint ϕ in n free variables, $\hat{d} \in \hat{\mathbb{S}}_n$ is an* approximate solution set of ϕ under a quantifier choice q^* *iff it contains all solution sets of ϕ under q^* and measure guesses v^*. Furthermore, $\tilde{d} \subseteq \hat{\mathbb{S}}_n$ is a* biapproximate solution set of ϕ *iff it is a set of approximate solution sets of ϕ.*

So for approximating don't know non-determinism one has to take a superset of all possible objects, but for approximating don't care non-determinism one has to take a subset of all possible objects. Note that if $n = 0$ then we speak of (bi)approximate truth values instead of solution sets. We call the set of all subsets of $\hat{\mathbb{S}}_n$ the set of all n-dimensional biapproximate solution sets and denote it by $\tilde{\mathbb{S}}_n$. Also for functions we can have these two forms of non-determinism: On the one hand, we can have several functions where we *don't know* which is the right one. One the other hand, we can have several functions where we *don't care* which one is chosen. Analogously to solution sets, we model the combination of both by sets of sets of functions, which we call *biapproximate functions.*

Again we can deduce the information that we want to find for our specification (figure 1) as follows:

Definition 14 *An $x \in \mathbb{R}^n$ is* determined to true *by an n-dimensional biapproximate solution set $\tilde{d}$ iff there is a $\hat{d} \in \tilde{d}$ such that x is determined to true by $\hat{d}$. It is* determined to false *by $\tilde{d}$ iff there is a $\hat{d} \in \tilde{d}$ such x is determined to false by $d \in \hat{d}$. It is* determined *by $\tilde{d}$ iff it is either determined to true or determined to false by $\tilde{d}$.*

For a constraint ϕ, the elements determined to true by the biapproximate solution set of ϕ are a subset of the elements that are in a solution set of ϕ (see def. 12), and the elements determined to false are a subset of the elements that are out of a solution set of ϕ. Again, elements for which we cannot deduce such information, constitute the error:

Definition 15 *For an n-dimensional biapproximate solution set $\tilde{d} \in \tilde{\mathbb{S}}_n$, the* error *of $\tilde{d}$ is the outer measure of the set of all $x \in \mathbb{R}^n$ for which x is not determined by $\tilde{d}$.*

All the above definitions are compatible with their counterparts in section 3 in the sense that a definition on a one-elementary biapproximate set is equivalent to the corresponding old definition on the set element.

As for classical quantification also approximate quantification results in a function on solution sets.

Definition 16 *For a nonnegative integer n, nonnegative real number q, and possible measure v, the* n-dimensional (q, v)-projection operator *is a function $P_{q,v} : \mathbb{S}_{n+1} \to \mathbb{S}_n$ such that $(x_1, \ldots, x_n) \in P_{q,v}(d)$ iff $v(A) > q$, where A is the set of all the y such that $(x_1, \ldots, x_n, y) \in d$.*

Given a fixed q, we call the set of all n-dimensional (q, v)-projection operators *n-dimensional approximate q-projection operator*. This is an approximate function. Given an interval $[\underline{q}, \overline{q}]$ we call the set of all n-dimensional approximate q-projection operators, where $q \in [\underline{q}, \overline{q}]$, *$n$-dimensional biapproximate $[\underline{q}, \overline{q}]$-projection operator.* This is a biapproximate function.

Also a biapproximate function results in a function on biapproximate solution sets:

Definition 17 *For a biapproximate function $\tilde{f} \subseteq 2^{\mathbb{S}_k \rightarrow \mathbb{S}_l}$, for any biapproximate solution sets $\tilde{d}$,*

$$Ext_{\tilde{f}}(\tilde{d}) := \{Ext_{\hat{f}}(\hat{d}) | \hat{f} \in \tilde{f}, \hat{d} \in \tilde{d}\}$$

We call $Ext_{\tilde{f}}$ the extension of *$\tilde{f}$.*

Now we compute biapproximate solution sets by applying the extension of the functions that correspond to the logical symbols. However, this is only possible for free constraints:

Lemma 1 *Let $(\exists^t_{[\underline{q},\overline{q}]})\phi$ be a constraint, and $\hat{d}_\phi$ an approximate solution set of ϕ under a quantifier choice q^* of $(\exists^t_{[\underline{q},\overline{q}]})\phi$. Let $\hat{P}$ be the n-dimensional approximate $q^*(t)$-projection operator. Then $Ext_{\hat{P}}(\hat{d}_\phi)$ is an approximate solution set of $(\exists^t_{[\underline{q},\overline{q}]})\phi$ under q^*.*

Theorem 1 *Let $(\exists^t_{[\underline{q},\overline{q}]})\phi$ be a free constraint, and $\tilde{d}_\phi$ a biapproximate solution set of ϕ. Let $\tilde{P}$ be the n-dimensional biapproximate $[\underline{q}, \overline{q}]$-projection operator. Then $Ext_{\tilde{P}}(\tilde{d}_\phi)$ is a biapproximate solution set of $(\exists^t_{[\underline{q},\overline{q}]})\phi$.*

Proof. Let $\hat{d}$ be an arbitrary but fixed element of $\mathrm{Ext}_{\tilde{P}}(\tilde{d})$. We have to prove that $\hat{d}$ is an approximate solution set of $(\exists^t_{[\underline{q},\overline{q}]})\phi$. By definition of extension we know that there is a $\hat{d}_\phi \in \tilde{d}_\phi$ and an approximate q-projection operator $\hat{P}$ with $q \in [\underline{q}, \overline{q}]$, such that $\hat{d} = \mathrm{Ext}_{\hat{P}}(\hat{d}_\phi)$. By definition of biapproximate solution set, $\hat{d}_\phi$ is an approximate solution set of ϕ under a quantifier choice q^*_ϕ of ϕ. Let q^* be such that it is equal to q^*_ϕ on all tags occuring in ϕ, and let it assign q to t, which is possible because $(\exists^t_{[\underline{q},\overline{q}]})\phi$ is free. Then, by the previous lemma, $\mathrm{Ext}_{\hat{P}}(\hat{d}_\phi) = \hat{d}$ is an approximate solution set of $(\exists^t_{[\underline{q},\overline{q}]})\phi$ under q^*. ■

For constraints that are not free, the resulting biapproximate solution set can contain elements that are no approximate solution sets of the constraint. The reason for this is, that the extension does not take into account tag equality.

Finally, we adapt the definition of the property needed for a function to be useable for computing approximate information up to arbitrary small error.

Definition 18 *A function $\tilde{f} : \tilde{\mathbb{S}}_k \to \tilde{\mathbb{S}}_l$ is* contracting *iff for all $\varepsilon \in \mathbb{R}^+$ there is an $\varepsilon' \in \mathbb{R}^+$ such that for all $\tilde{d} \in \tilde{\mathbb{S}}_k$ such that the error of $\tilde{d}$ is less or equal ε', the error of $\tilde{f}(\tilde{d})$ is less or equal ε.*

Again this definition is compatible with the corresponding one in section 3. Before studying, whether biapproximate projection is contracting, we first study biapproximate solution sets in more details.

6 Representing Biapproximate Solution Sets

Observe that different biapproximate solution sets can contain exactly the same information we need to find for fulfilling our specification in figure 1. For example, if we have the biapproximate truth value $\{\{\mathbf{T}\}\}$ of a sentence, then we know that the sentence is true. But, if we have the biapproximate truth value $\{\{\mathbf{T}\}, \{\mathbf{T}, \mathbf{F}\}\}$, then we have exactly the same information. So there is some interaction between the two forms of non-determinism that biapproximate solution sets do not take into account, and we can divide the biapproximate solution sets into equivalence classes such that each equivalence class element contains the same information.

Recall that we can compute our specification by treating all logical symbols as functions from (bi)approximate solution sets to (bi)approximate solution sets. In this section we show, that we need to implement these functions just on the above equivalence classes and not on biapproximate solution sets in general. Since this set of equivalence classes has a much lower cardinality, we can then find a representation that is better suited for computer implementation. Furthermore we gain a valuable tool for the subsequent proof that the biapproximate $[\underline{q}, \overline{q}]$-projection operators are contracting, and find interesting insight in the interaction between don't know and don't care non-determinism.

Before studying the general case of biapproximate solution sets, we start with the easier case of approximate solution sets. The information we want to extract are the elements determined to true, and the elements determined to false by an approximate solution set $\hat{d}$. These are exactly the elements $\bigcap \hat{d}$, and all elements not in $\bigcup \hat{d}$. This gives us the equivalence relation $\hat{d}_1 \sim \hat{d}_2$ iff $(\bigcap \hat{d}_1, \bigcup \hat{d}_1) = (\bigcap \hat{d}_2, \bigcup \hat{d}_2)$.

Now let us also define an order $\leq$ on functions such that $f_1 \leq f_2$ iff for all d, $f_1(d) \subseteq f_2(d)$. Also here we can form an equivalence relation on approximate functions by defining $\hat{f}_1 \sim \hat{f}_2$ iff $(\min \hat{f}_1, \max \hat{f}_1) = (\min \hat{f}_2, \max \hat{f}_2)$. A function $f : \mathbb{S}_k \to \mathbb{S}_l$ is *monotonic* iff $d_1 \subseteq d_2$ implies $f(d_1) \subseteq f(d_2)$.

Equally, an approximate or biapproximate function is monotonic iff all its members are monotonic.

For computing with the equivalence classes instead of their members, we need to know whether all members of equivalence classes behave equally for function application. For this we pick a *canonical representative* from each equivalence class and proving that all the other class members behave in the same way as this representative. We can order approximate solution sets by the subset relation, and so we can also use interval notation on them: $[\underline{d}, \overline{d}] := \{d | \underline{d} \subseteq d \subseteq \overline{d}\}$. Observe that for any approximate solution set $\hat{d}$, the interval $r(\hat{d}) := [\bigcap \hat{d}, \bigcup \hat{d}]$ is equivalent to $\hat{d}$, and for any approximate function $\hat{f}$ the interval $r(\hat{f}) := [\min \hat{f}, \max \hat{f}]$ is equivalent to $\hat{f}$. So we take them as the canonical representatives. Now we have the following congruence property:

Lemma 2 *For an approximate solution set $\hat{d}$ and monotonic approximate function $\hat{f}$, $r(Ext_{\hat{f}}(\hat{d})) = [\underline{f}(\underline{d}), \overline{f}(\overline{d})]$, where $[\underline{d}, \overline{d}] = r(\hat{d})$, and $[\underline{f}, \overline{f}] = r(\hat{f})$.*

Proof. By definition, $r(\mathrm{Ext}_{\hat{f}}(\hat{d}))$ is equal to

$$[\bigcap \mathrm{Ext}_{\hat{f}}(\hat{d}), \bigcup \mathrm{Ext}_{\hat{f}}(\hat{d})]$$

Now, by inserting the definition of $\mathrm{Ext}_{\hat{f}}$ and by monotonicity of $\hat{f}$

$$\bigcap \mathrm{Ext}_{\hat{f}}(\hat{d}) = \bigcap \{f(d) | f \in \hat{f}, d \in \hat{d}\} = (\min \hat{f})(\bigcap \hat{d}) = \underline{f}(\underline{d})$$

and

$$\bigcup \mathrm{Ext}_{\hat{f}}(\hat{d}) = \bigcup \{f(d) | f \in \hat{f}, d \in \hat{d}\} = (\max \hat{f})(\bigcup \hat{d}) = \overline{f}(\overline{d}).$$

■

The (q, v)-projection operators are monotonic. Furthermore the representation of each q-projection operator is the interval $[P_{q,\underline{v}}, P_{q,\overline{v}}]$. So we can compute the approximate projection of an approximate solution set by just computing with the interval bounds of the corresponding representations.

Now we study the general case of biapproximate solution sets. In a similar way as approximate solution sets, different biapproximate solution sets can yield the same information. Also here the information we want to extract are the elements determined to true, and the elements determined to false by a biapproximate solution set $\tilde{d}$. These are the elements $\bigcup\{\bigcap \hat{d} | \hat{d} \in \tilde{d}\}$, and all elements not in $\bigcap\{\bigcup \hat{d} | \hat{d} \in \tilde{d}\}$. Again, this gives us an equivalence relation $\tilde{d}_1 \sim \tilde{d}_2$ iff $(\bigcup\{\bigcap \hat{d} | \hat{d} \in \tilde{d}_1\}, \bigcap\{\bigcup \hat{d} | \hat{d} \in \tilde{d}_1\}) = (\bigcup\{\bigcap \hat{d} | \hat{d} \in \tilde{d}_2\}, \bigcap\{\bigcup \hat{d} | \hat{d} \in \tilde{d}_2\})$. In a similar way we get an equivalence relation on biapproximate functions by $\tilde{f}_1 \sim \tilde{f}_2$ iff $(\max\{\min \hat{f} | \hat{f} \in \tilde{f}_1\}, \min\{\max \hat{f} | \hat{f} \in \tilde{f}_1\}) = (\max\{\min \hat{f} | \hat{f} \in \tilde{f}_2, \min\{\max \hat{f} | \hat{f} \in \tilde{f}_2\})$.

Again we pick canonical representatives from the equivalence classes. For any elements $\underline{x}$ and $\overline{x}$ in a domain with a partial order $\leq$, let $[\![\underline{x}, \overline{x}]\!] := \{[y, y \cup \overline{x}] | y \in [\underline{x} \cap \overline{x}, \underline{x}]\}$. We call such an object *biinterval* (algebraically speaking the resulting objects form a bilattice [14, 12]). By forming biintervals we stay within an equivalence class:

Lemma 3 *For every biapproximate object $\tilde{a}$,*

$$[\![\bigcup\{\bigcap \hat{a} | \hat{a} \in \tilde{a}\}, \bigcap\{\bigcup \hat{a} | \hat{a} \in \tilde{a}\}]\!] = \tilde{a}$$

Proof.
We have:

$$\bigcup\{\bigcap \hat{a} | \hat{a} \in [\![\underline{a}, \overline{a}]\!]\} = \bigcup\{\bigcap [a, a \cup \overline{a}] | a \in [\underline{a} \cap \overline{a}, \underline{a}]\} = \bigcup\{a | a \in [\underline{a} \cap \overline{a}, \underline{a}]\} = \underline{a}$$

In in a similar way $\bigcap\{\bigcup \hat{a} | \hat{a} \in [\![\underline{a}, \overline{a}]\!]\} = \overline{a}$. ■

So again, we denote the canonical representative $[\![\bigcup\{\bigcap \hat{d} | \hat{d} \in \tilde{d}\}, \bigcap\{\bigcup \hat{d} | \hat{d} \in \tilde{d}\}]\!]$ of $\tilde{d}$ by $r(\tilde{d})$, and the canonical representative $[\![\max\{\min \hat{f} | \hat{f} \in \tilde{f}\}, \min\{\max \hat{f} | \hat{f} \in \tilde{f}\}]\!]$ of $\tilde{f}$ by $r(\tilde{f})$. Also here we can compute with the representatives because of the following congruence property:

Theorem 2 *For a biapproximate solution set $\tilde{d}$ and a monotonic biapproximate function $\tilde{f}$, $r(Ext_{\tilde{f}}(\tilde{d})) = [\![\underline{f}(\underline{d}), \overline{f}(\overline{d})]\!]$, where $[\![\underline{d}, \overline{d}]\!] = r(\tilde{d})$, and $[\![\underline{f}, \overline{f}]\!] = r(\tilde{f})$.*

Proof. By definition, $r(\mathrm{Ext}_{\tilde{f}}(\tilde{d}))$ is equal to

$$[\![\bigcup\{\bigcap \hat{d} | \hat{d} \in \mathrm{Ext}_{\tilde{f}}(\tilde{d}), \bigcap\{\bigcup \hat{d} | \hat{d} \in \mathrm{Ext}_{\tilde{f}}(\tilde{d})\}]\!]$$

We have:

$$\begin{aligned}
\bigcup\{\bigcap \hat{d} | \hat{d} \in \mathrm{Ext}_{\tilde{f}}(\tilde{d})\} &= && \text{def. of Ext} \\
\bigcup\{\bigcap \mathrm{Ext}_{\hat{f}}(\hat{d}) | \hat{f} \in \tilde{f}, \hat{d} \in \tilde{d}\} &= && \text{def. of Ext.} \\
\bigcup\{\bigcap\{f(d) | f \in \hat{f}, d \in \hat{d}\} | \hat{f} \in \tilde{f}, \hat{d} \in \tilde{d}\} &= && \text{monot.} \\
\bigcup\{(\min \hat{f})(\bigcap \hat{d}) | \hat{f} \in \tilde{f}, \hat{d} \in \tilde{d}\} &= && \text{monot.} \\
(\max\{\min \hat{f} | \hat{f} \in \tilde{f}\})(\bigcup\{\bigcap \hat{d} | \hat{d} \in \tilde{d}\}) &= && \text{def. of r} \\
& \underline{f}(\underline{d})
\end{aligned}$$

For the dual case an analogous argument holds. ■

Now observe that for a biapproximate $[\underline{q}, \overline{q}]$-projection operator $\tilde{P}$, $r(\tilde{P})$ is the biinterval $[\![P_{\underline{q},\underline{v}}, P_{\overline{q},\overline{v}}]\!]$. So for every biapproximate solution set $\tilde{d}$ we can compute the projection of its equivalence class by just working on the bounds $\underline{d}$ and $\overline{d}$ of $r(\tilde{d})$ to get $[\![P_{\underline{q},\underline{v}}(\underline{d}), P_{\overline{q},\overline{v}}(\overline{d})]\!]$.

7 Biapproximate Projection is Contracting

Now we are ready to prove that for approximate quantifiers we can compute approximate solution sets up to arbitrarily small error bounds. Here we have to take the additional assumption, that all occuring sets are measurable (i.e., the inner and outer measures coincide). So we use the term "volume" instead of measure.

Definition 19 *Let $\tilde{d}$ be an $n+1$-dimensional biapproximate solution set. For any $(x_1, \ldots, x_n)$, the* error *of $\tilde{d}$* over *$(x_1, \ldots, x_n)$ is the volume of all y for which $(x_1, \ldots, x_n, y)$ is not determined by $\tilde{d}$.*

Observe that x is not determined by a biapproximate solution set with representation $[\![\underline{d}, \overline{d}]\!]$ iff $x \notin \underline{d}$ and $x \in \overline{d}$. We first prove that for single points in the free variable space we can attain an arbitarily small error.

Lemma 4 *For every $n+1$-dimensional biapproximate solution set $\tilde{d}$, and n-dimensional biapproximate $[\underline{q}, \overline{q}]$-projection operator $\tilde{P}$ with $\underline{q} < \overline{q}$, for all $(x_1, \ldots, x_n)$ such that $(x_1, \ldots, x_n)$ is not determined by $r(Ext_{\tilde{P}}(\tilde{d}))$, the error of $\tilde{d}$ over $(x_1, \ldots, x_n)$ is greater than $\overline{q} - \underline{q}$.*

Proof. Let $r(d) = [\![\underline{d}, \overline{d}]\!]$ and $(x_1, \ldots, x_n)$ such that it is not determined by $r(\mathrm{Ext}_{\tilde{P}}([\![\underline{d}, \overline{d}]\!]))$, which is $[\![P_{\underline{q},\underline{v}}(\underline{d}), P_{\overline{q},\overline{v}}(\underline{d})]\!]$ by theorem 2. So, by definition of projection operator, $\underline{v}(\{y | (x_1, \ldots, x_n) \in \underline{d}\}) \leq \underline{q}$ and $\overline{v}(\{y | (x_1, \ldots, x_n) \in \overline{d}\}) > \overline{q}$. So the volume of the y that are determined to true by $[\underline{d}, \overline{d}]$ is less or equal $\underline{q}$, and the volume of the y that are not determined to false by $[\underline{d}, \overline{d}]$ is greater than $\overline{q}$. Thus the volume of the $(x_1, \ldots, x_n)$ that are not determined by $[\underline{d}, \overline{d}]$ is greater than $\overline{q} - \underline{q}$. ■

Since we have shown in the last section that for computing our specification (figure 2), we can apply the representation of projection operators instead of the projection operators themselves, we now prove:

Theorem 3 *For $\underline{q} < \overline{q}$, and n-dimensional biapproximate $[\underline{q}, \overline{q}]$-projection operator $\tilde{P}$, $r(Ext_{\tilde{P}})$ is contracting.*

Proof. We have to prove that for all $\varepsilon \in \mathbb{R}^+$, there is an $\varepsilon' \in \mathbb{R}^+$ such that for every $n+1$ dimensional biiapproximate solution set $\tilde{d}$ such that the error of $\tilde{d}$ is greater than ε', the error of $r(\mathrm{Ext}_{\tilde{P}})(\tilde{d})$ is greater than ε.

Choose $(\overline{q}-\underline{q})\varepsilon$ for ε', and let $\tilde{d}$ be an arbitrary but fixed biapproximate solution set. We assume that the error of $r(\mathrm{Ext}_{\tilde{P}})(\tilde{d})$ is less or equal ε and prove that the error of $\tilde{d}$ is less or equal $(\overline{q}-\underline{q})\varepsilon$.

By lemma 4, for every $(x_1,\ldots,x_n)$ that is not determined by $r(\mathrm{Ext}_{\tilde{P}})(\tilde{d})$ the error of $\tilde{d}$ over $(x_1,\ldots,x_n)$ is greater than $\overline{q}-\underline{q}$. The volume of these $(x_1,\ldots,x_n)$ is greater than ε. So, by Fubini's theorem, the error of $\tilde{d}$ is greater than $(\overline{q}-\underline{q})\varepsilon$. ■

Note that, if unmeasurable sets occur, then we still can ensure that the projection operator is contracting, by making the interval $[\underline{q},\overline{q}]$ big enough. In general, we can make computation faster by increasing the size of the interval $[\underline{q},\overline{q}]$. On the other hand, one could extend the language to allow biinterval annotations. In this case $\underline{q}$ could be bigger than $\overline{q}$, which means that we deliberately do not derive any information for solution sets whose size is between $\overline{q}$ and $\underline{q}$.

In a similar way as the projection operators for quantifiers, also for the other logical symbols ($\vee$, $\wedge$, $\neg$) there are corresponding functions on solution sets. The functions corresponding to conjunction and disjunction are monotonic, and for the function N corresponding to negation, $d_1 \subseteq d_2$ implies $N(d_1 \supseteq d_2)$, which allows us to prove a similar property as theorem 2. So we can use biinterval representation to propagate information from atomic constraints to arbitary constraints. Furthermore one can easily show that the functions corresponding to $\vee$, $\wedge$ and $\neg$ are contracting. So we can compute approximate solution sets of free constraints from approximate solution sets of their atomic sub-constraints, as described in section 3. Furthermore — as long as no unmeasurable solution sets occur and provided that for all approximate quantifiers the left bound $\underline{q}$ of the annotation is strictly smaller than the right bound $\overline{q}$ — we can attain arbitrarily small error.

8 Related Work

Languages for modeling nondeterminism in various other forms have been introduced within the frame of abstract data types — see for example [42, 41, 19, 24]. There one uses nondeterministic specifications to either model nondeterminism occuring in reality, or to abstract away unnecessary details of the behaviour of a real or desired system; these details might be specified later.

In contrast to that, in our work the deterministic (exact) specification is already given, and we introduce nondeterminism only later, in order to be able to do approximate copmutation. Furthermore the above approaches are always confined to one form of nondeterminism at a time.

The idea to allow several equally valid outputs (i.e., don't care nondeterminism) to make certain problems computable, already has been taken by Brouwer [7, 44], who showed that the computable real numbers need a redundant representation in order to arrive at terminating computation. Similar ideas are nowadays frequently used for being able to do approximate computation for other inherently discontinuous problems (e.g., GCD-computation [35]). In our work, we have the additional difficulty, that two different kinds of nondeterminism need to be recursively propagated.

Modifications of the first-order language that allow reasoning about the size of sets have been studied by Keisler [25] (coming from logic), Bacchus [3, 4] (knowledge representation), and they are a main topic in the area of generalized quantifiers [29, 40]. However, these languages do not allow any nondeterministic choice of the size specification and they circumvent the problem of how to deal with unmeasurable sets by allowing only domains where all resulting solution sets are measurable.

9 Conclusion

We have shown how to compute approximate solution sets of quantified real constraints up to a user-specified error bound. For constraints containing classical quantifiers such an error bound cannot always be attained. We have provided a remedy for this problem by replacing the classical quantifiers in the first-order predicate language by approximate quantifiers.

In addition to enabling approximation algorithms, this gives us both expressive power in reasoning and the possibility of tunable algorithms (algorithms where the user can decide about the tradeoff between speed and precision). We have implemented such an algorithm — a detailed description and analysis of this implementation will be published elsewhere.

The question remains, how big we should choose the intervals of the quantifier annotation. Too small intervals can hamper the efficiency of constraint solving, while too big intervals can disturb the information one is looking for. Regularization theory [11] provides various tools for this choice; thus we will try to apply it here.

Another interesting observation is, that the notion of continuity in analysis is a special case our notion of a contracting (bi)approximate function. Furthermore our representation of biapproximate objects amounts to some form of rounding [1, 26, 27]). Future work will clarify the details.

Most of this research was done as a part of the author's Ph.D. work. The author thanks his Ph.D. advisor, Hoon Hong, for all the guidance that is also reflected in both the structure and content of this paper.

References

[1] R. Albrecht. Roundings and approximations in ordered sets. In G. Alefeld and R. Grigorieff, editors, *Fundamentals of Numberical Computation (Computer-*

Oriented Numerical Analysis), volume 2 of *Computing Supplementum*, pages 17–31. Springer, 1980.

[2] G. Alefeld and J. Herzberger. *Introduction to Interval Computations.* Academic Press, New York, 1983.

[3] F. Bacchus. Lp, a logic for representing and reasoning with statistical knowledge. *Computational Intelligence*, 6:209–231, 1990.

[4] F. Bacchus. *Representing and Reasoning with Probabilistic Knowledge.* The MIT Press, 1990.

[5] H. Bauer. *Mass- und Integrationstheorie.* Walter de Gruyter, 1990.

[6] B. Bouchon-Meunier and V. Kreinovich. From interval computations to modal mathematics: Applications and computational complexity. Technical Report UTEP-CS-96-47, University of Texas at El Paso, Computer Science Department, November 1996.

[7] L. Brouwer. Besitzt jede reelle Zahl eine Dezimalbruchentwicklung? *Math. Ann.*, 83:201–210, 1920.

[8] B. F. Caviness and J. R. Johnson, editors. *Quantifier Elimination and Cylindrical Algebraic Decomposition.* Texts and Monographs in Symbolic Computation. Springer, 1998.

[9] G. E. Collins. Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition. In *Second GI Conf. Automata Theory and Formal Languages*, volume 33 of *Lecture Notes in Computer Science*, pages 134–183. Springer- Verlag, Berlin, 1975. Also in [8].

[10] J. H. Davenport and J. Heintz. Real quantifier elimination is doubly exponential. *Journal of Symbolic Computation*, 5:29–35, 1988.

[11] H. W. Engl. Regularization methods for the stable solution of inverse problems. *Surv. Math. Ind*, 3:71–143, 1993.

[12] M. Fitting. Bilattices and the semantics of logic programming. *Journal of Logic Programming*, 11(2):91–116, 1991.

[13] E. Gardeñes, H. Mielgo, and A. Trepat. Modal intervals: Reason and ground semantics. In K. Nickel, editor, *Interval Mathematics 1985*, volume 212 of *Lecture Notes in Computer Science*, pages 27–35, Berlin, Heidelberg, 1968. Springer- Verlag.

[14] M. L. Ginsberg. Multivalued logics: A uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4:265–316, 1988.

[15] L. Gonzalez-Vega. A combinatorial algorithm solving some quantifier elimination problems. In B. F. Caviness and J. R. Johnson, editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Texts and Monographs in Symbolic Computation, pages 365–375. Springer Verlag, 1996.

[16] P. Halmos. *Measure Theory.* Van Nostrand, 1950.

[17] E. Hansen. *Global Optimization Using Interval Analysis.* Marcel Dekker, Inc, 1992.

[18] E. Hehner. Predicative programming. *Communications of the ACM*, 27:134–151, Feb. 1984.

[19] W. H. Hesselink. A mathematical approach to nondeterminism in data types. *ACM Transactions on Programming Languages and Systems*, 10(1):87–117, 1988.

[20] C. Hoare. Unified theories of programming. Oxford University Computing Laboratory, `http://www.comlab.ox.ac.uk/oucl/users/tony.hoare/publications.html`, July 1994.

[21] H. Hong. Quantifier elimination for formulas constrained by quadratic equations via slope resultants. *The Computer Journal*, 36(5):440–449, 1993. Special issue on computational quantifier elimination.

[22] H. Hong. Symbolic-numeric methods for quantified constraint solving. In *International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics SCAN-95*, 1995. Invited Talk.

[23] H. Hong and A. Neubacher. Approximate quantifier elimination. In *Proceedings of IMACS-ACA'96*, 1996.

[24] H. Hussmann. *Nondeterminism in Algebraic Specifications and Algebraic Programs*. Birkhäuser, 1993.

[25] H. J. Keisler. Probability quantifiers. In J. Barwise and S. Feferman, editors, *Model Theoretic Logics*, chapter XIV. Springer Verlag, New York, 1985.

[26] U. W. Kulisch and W. Miranker. Arithmetic operations in interval spaces. In G. Alefeld and R. Grigorieff, editors, *Fundamentals of Numberical Computation (Computer-Oriented Numerical Analysis)*, volume 2 of *Computing Supplementum*, pages 51–67. Springer, 1980.

[27] U. W. Kulisch and W. L. Miranker. *Computer Arithmetic in Theory and Practice*. Academic Press, 1981.

[28] R. E. Moore. *Interval Analysis*. Prentice Hall, Englewood Cliffs, NJ, 1966.

[29] A. Mostowski. On a generalization of quantifiers. *Fundamenta Mathematicae*, 44:12–36, 1957.

[30] A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge Univ. Press, Cambridge, 1990.

[31] U. K. R. Hammer, M. Hocks and D. Ratz. *Numerical Toolbox for Verified Computing I*, chapter Nonlinear Equations in One Variable. Number 21 in Series in Computational Mathematics. Springer Verlag, 1993.

[32] S. Ratschan. Uncertainty propagation in heterogenous algebras for approximate quantified constraint solving. *Journal of Universal Computer Science*, 2000. to appear.

[33] H. Ratschek and J. Rokne. *Computer Methods for the Range of Functions*. Ellis Horwood Limited, 1984.

[34] N. Rescher. *Many-valued Logic*. McGraw-Hill, New York, 1969.

[35] A. Schönhage. Quasi-gcd computations. *J. Complexity 1*, pages 118–137, 1985.

[36] S. P. Shary. Algebraic approach to the interval linear static identification, tolerance, and control problems, or one more application of kaucher arithmetic. *Reliable Computing*, 2(1):3–33, 1996.

[37] V. Stahl. *Interval Methods for Bounding the Range of Polynomials and Solving Systems of Nonlinear Equations.* PhD thesis, Research Institute for Symbolic Computation, Johannes Kepler University, A-4040 Linz, Austria, 1996.

[38] A. Tarski. *A Decision Method for Elementary Algebra and Geometry.* Univ. of California Press, Berkeley, 1951. Also in [8].

[39] A. Urquhart. Many-valued logic. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic, Vol. III: Alternatives in Classical Logic*, chapter III.2, pages 71–116. D. Reidel Publishing Company, 1986.

[40] J. Väänänen. Generalized quantifiers. *Bulletin of the European Association for Theoretical Computer Science*, 1997.

[41] M. Walicki and S. Meldal. A complete calculus for the multialgebraic and functional semantics of nondeterminism. *ACM Transactions on Programming Languages and Systems*, 17(2), 1995.

[42] M. Walicki and S. Meldal. Algebraic approaches to nondeterminism: An overview. *ACM Computing Surveys*, 29(1), 1997.

[43] V. Weispfenning. Quantifier elimination for real algebra — the cubic case. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation in Oxford*, pages 258–263. ACM Press, July 1994.

[44] E. Wiedmer. Computing with infinite objects. *Theoretical Computer Science*, 10:133–155, 1980.