Approximate Quantified Constraint Solving By Cylindrical Box Decomposition

Stefan Ratschan*

September 6, 2000

Abstract

This paper applies interval methods to a classical problem in computer algebra. Let a quantified constraint be a first-order formula over the real numbers. As shown by A. Tarski in the 1930's, such constraints, when restricted to the predicate symbols $<$, $=$ and function symbols $+$, $\times$, are in general solvable. However, the problem becomes undecidable, when we add additional function symbols like sin or exp. Furthermore, all exact algorithms known up to now are too slow for big examples, do not provide partial information before computing the total result, cannot satisfactorily deal with interval constants in the input, and often generate huge output. As a remedy we propose an approximation method based on interval arithmetic. It uses a generalization of the notion of cylindrical algebraic decomposition — as introduced by G. Collins. We describe an implementation of the method and demonstrate that it can efficiently give approximate information on problems that are too hard for current exact methods.

1 Introduction

This paper deals with solving quantified constraints (i.e. first-order formulae over the real numbers). This problem has various applications, for example in control theory and stability analysis of differential equations [26, 10, 31].

A. Tarski [49] showed that the theory of real-closed fields allows quantifier elimination, which means that one can solve quantified constraints that contain only the predicate symbols $<$ and $=$, and the function symbols $+$ and $\times$. But as soon as one adds function symbols like sin or exp, the theory becomes undecidable [49, 46] (although, by imposing certain restrictions on the constraint structure one can come up with solvers for special cases [39, 2]).

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

Up to now, Collins' method [7] is the only general quantifier elimination method in the theory of real-closed fields for which a practically useful implementation [20] exists. This implementation is quite usable for many small and a few middle-size examples, however it usually cannot finish big examples. More efficient solvers exist only for special cases [51, 22, 17].

Further problems that occur when trying to solve quantified constraints exactly include: Current methods cannot provide partial information when interrupted before computing the total result [18, 47, 4]; they cannot deal with input that contains constants that come from measurements and are thus not exact but only known to be elements of an interval; the output very often consists of complex algebraic expressions that need further processing in order to be useful.

To deal with these problems, H. Hong [23, 27] proposed the use of algorithms based on interval arithmetic [35, 1, 45, 38, 19, 41], that compute approximate instead of exact solutions, that is, he proposed to do approximate quantified constraint solving. This means that the user provides an error bound, and the computer solves the problem up to this error bound. However, the method he provided (together with A. Neubacher) [27, 37] still was far too slow to be useful, could not guarantee termination, could still not deal with transcendental functions, and was too complicated to allow a thorough study of its properties.

In two earlier papers [44, 43] we gave a general framework for approximate quantified constraint solving, which allowed the proof of guaranteed termination, can deal with transcendental functions, and is elegant and simple enough to allow thorough study of its properties.

The contribution of this paper is an efficient algorithm based on the preceding framework. Like the older method by Hong and Neubacher, it is based on interval arithmetic and is an *incremental approximation algorithm* — an algorithm that improves an initial guess until the specified error bound is attained.

The method structures data using a generalization of the notion of cylindrical algebraic decomposition, as used in Collins' method [7, 3]. In contrast to the latter it takes into account details of the structure of input constraints, also in non-prenex form. This results in a tree whose nodes correspond to the quantifiers, connectives, or atomic constraints of the input.

The current implementation [42] of the algorithm considers the symbols $+$, $\times$, $\leq$, $<$, $=$ and $\neq$ but can be easily extended to further symbols (like $\sin, \exp$, etc.). It uses various standard tools from interval computation (e.g., range overestimation) and the tightening method of Hong and Stahl [29].

Special cases of our general problem have already been successfully studied by the interval community [48, 5, 16, 15]. Other applications of interval computations to computer algebra mainly are based on the observation that often approximate intermediate computation suffices for producing exact results [32, 9].

The structure of the paper is as follows: In section 2, we describe a basic algorithm. In section 3, we improve it to an incremental approximation algorithm. In section 4, we explain the central data structure. In section 5, we give a formalization of the algorithm. In section 6, we describe its implementation. In section 7, we discuss efficiency and give timings, and in section 8, we make final conclusions.

2 Basic Algorithm

In this section we present the formal problem specification and give a basic algorithm for fulfilling the specification. This algorithm mainly serves for illustration, and does not yet take into account efficiency considerations.

As already mentioned, a (quantified) constraint is a first-order formula over the reals. So it contains quantifiers ($\exists$, $\forall$), connectives ($\wedge$, $\vee$, $\neg$), predicate symbols (e.g., $=$, $<$, $\leq$), function symbols (e.g., $+$, $-$, $\times$, sin, exp), rational constants and variables ranging over real numbers. The semantics of quantified constraints is the usual one of the first-order predicate language and the theory of real numbers. In this paper we additionally allow floating point interval constants — we will define the according semantics later. In order to avoid technical problems, from now on we only allow constraints where all quantified variables are mutually different, and different from the free variables. This can be easily achieved by renaming of variables.

A constraint is a *proper* sub-constraint of a constraint ϕ iff it is a sub-constraint of ϕ but not equal to ϕ. It is an *immediate* sub-constraint of ϕ iff it is a proper sub-constraint of ϕ but has no super-constraint that is a proper sub-constraint of ϕ.

We are interested in computing the solution set of constraints, which can be approximated as follows (using slightly more specialized notions than in [44, 43]):

Definition 1 *For any constraint ϕ in n free variables and $x \in \mathbb{R}^n$, $t \in \{\mathbf{T}, \mathbf{F}, \mathbf{U}\}$ is an* approximate truth value of ϕ at x *iff $t = \mathbf{T}$ implies that for all choices of elements of the interval constants in ϕ, x is in the according solution set of ϕ and $t = \mathbf{F}$ implies that for all choices of elements of the interval constants in ϕ, x is not in the according solution set of ϕ. For a set $B \subseteq \mathbb{R}^n$, $t \in \{\mathbf{T}, \mathbf{F}, \mathbf{U}\}$ is an* approximate truth value of ϕ on B *iff for all $x \in B$, x is an approximate truth value of ϕ at x.*

If an approximate truth value is different from $\mathbf{U}$, we say that it is *known*.

Definition 2 *For any constraint ϕ in n free variables and $B \subseteq \mathbb{R}^n$, an* approximate solution set of ϕ on B *is a function $S : B \to \{\mathbf{T}, \mathbf{F}, \mathbf{U}\}$ such that for all $x \in \mathbb{R}^n$, $S(x)$ is an approximate truth value of ϕ at x. An* approximate solution set of ϕ *is an approximate solution set of ϕ on $\mathbb{R}^n$.*

We say that an approximate solution set is *trivial* iff it assigns **U** everywhere. Now we can measure the quality of approximations:

Definition 3 *The* error *of an approximate solution set S is the volume of the set of all elements x where $S(x)$ is not known.*

Now we can give the exact input/output specification of our problem (see figure 1).

GIVEN:	A constraint ϕ, $\varepsilon \in \mathbb{R}^+$
FIND:	An approximate solution set of ϕ with error smaller than ε

Figure 1: Problem Specification

We use the following basic algorithm for fulfilling this specification:

1. **Approximation:** Compute approximate solution sets of the atomic subconstraints of ϕ up to a sufficient error bound.

2. **Propagation:** Propagate these approximate solution sets to an approximate solution set of the total constraint ϕ.

Here we have to clarify two questions:

- What does "sufficient" mean in step one?

- How to do propagation in step two?

We first concentrate on the second question. Observe that from approximate solution sets of all the immediate sub-constraints of a constraint one can compute an approximate solution set of the total constraint. So connectives and quantifiers L can act as functions that take approximate solution sets of constraints $\phi_1, \ldots, \phi_n$ and produce an approximate solution set of $L(\phi_1, \ldots, \phi_n)$. For any symbol L we denote this *propagation function* by P_L. For lack of space we do not give a detailed definition but only provide two representative examples here (for details see [44]), where for n-dimensional approximate solution sets S, S_1, S_2, and $x \in \mathbb{R}^n$:

$$P_\wedge(S_1, S_2)(x) = \begin{cases} \mathbf{T} & \text{if } S_1(x) = \mathbf{T} \text{ and } S_2(x) = \mathbf{T}, \\ \mathbf{F} & \text{if } S_1(x) = \mathbf{F} \text{ or } S_2(x) = \mathbf{F}, \text{ and} \\ \mathbf{U} & \text{otherwise.} \end{cases} \tag{1}$$

$$P_\exists(S)(x) = \begin{cases} \mathbf{T} & \text{if there is a } y \in \mathbb{R} \text{ such that } S(x,y) = \mathbf{T}, \\ \mathbf{F} & \text{if for all } y \in \mathbb{R},\ S(x,y) = \mathbf{F}, \text{ and} \\ \mathbf{U} & \text{otherwise.} \end{cases} \tag{2}$$

By recursive application of these propagation functions over the structure of constraints we can compute an approximate solution set of a constraint from approximate solution sets of its atomic sub-constraints. This process can be visually illustrated by viewing constraints as trees, in which the atomic constraints are leaves and the connectives and quantifiers are nodes (e.g., the tree corresponding to the constraint $\exists x\, [\phi_1 \wedge \phi_2]$ can be seen in figure 2). We compute approximate solution sets of constraints by computing approximate solution sets of the tree leaves and propagating this information up to an approximate solution set of the total constraint.

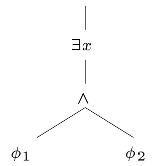

Figure 2: Tree Representation of Constraints

Now we investigate what "sufficient" means in step 1. Here we need to ensure that the error of the resulting approximate solution set of the total constraint is smaller than the error bound required by the specification (figure 1). That is, we need to find a positive bound on the error of the approximate atomic solution sets, such that the error of the resulting approximate solution set of the total constraint is smaller than the user-specified error bound. Unfortunately, as shown in [43], this is not possible in general. In other words, it can happen that one must do exact computation, even if one allows approximation in the output. This is an inherent problem of approximate computation [50]. But fortunately, there is a remedy for this, that works by replacing the classical quantifiers by different, approximate quantifiers [43].

For lack of space, we cannot give a detailed description of the semantics of approximate quantifiers, but only an informal one: The first step for their introduction is, to allow quantifiers with a positive real annotation q, with the intuitive meaning that a constraint $\exists_q x\ \phi(x)$ is true iff the volume of the solution set of ϕ is greater than q. Also for these quantifiers there are no sufficient bounds on the error of atomic sub-constraints. But this changes if we allow quantifiers to be annotated with an interval $[\underline{q}, \overline{q}]$, where $\underline{q} < \overline{q}$, with the intuitive meaning that the true annotation can be somewhere within this interval. This allows an algorithm to choose the most suitable value between $\underline{q}$ and $\overline{q}$ and we don't care which one. This means that if the quantified solution set is greater than $\overline{q}$ then the result is **T**, if the quantified solution set is smaller or equal $\underline{q}$ then the result is **F**, and otherwise the result is allowed to be any of **T** or **F** (see figure 3).

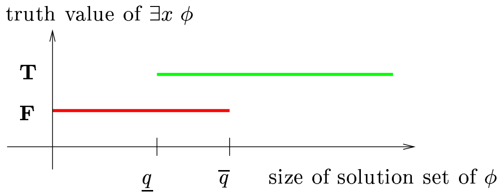

Figure 3: Approximate Quantifiers

For constraints that contain approximate quantifiers instead of classical quantifiers, one can always find the needed bounds. But if a constraint contains interval coefficients then it may be impossible to compute approximate solution sets of atomic constraints to the required bounds. In this case we have to make the annotation interval of the approximate quantifiers wider, which relaxes the required error bounds.

Approximate quantifiers have the additional advantage, that they allow reasoning about the size of solution sets. For example, this is useful if one is only interested in solution sets of a certain size. But since constraints that contain equalities generate solution sets of zero volume, approximate quantifiers are of limited use for such inputs. So we concentrate on constraints without equalities in this paper.

3 Incremental Approximation

For our basic algorithm we need a-priori bounds on the error of the atomic approximate solution sets computed in step 1. These error bounds are independent of the actual input and have to take into account the worst case. Thus they are much too high for the average case, and usually approximate solution sets of much higher error suffice. But this fact can only be detected during computation. So, instead of computing the approximate solution sets of the atomic constraints at once, we do this by decreasing their error (*refining them*) step by step. For this we initialize them with trivial approximate solution sets, and continue with the following algorithm:

1. **Approximation:** Refine approximate solution sets of the atomic subconstraints.

2. **Propagation:** Propagate these approximate solution sets to an approximate solution set of the total constraint.

until the desired overall error bound is reached.

By making sure that the error of the approximations of the atomic sub-constraints goes to zero, we can ensure that we eventually reach the a-priori error bounds. So we can make the algorithm terminate for all inputs constraints that contain approximate quantifiers instead of classical ones.

It makes sense to apply the idea of incrementality even further. For this we view constraints as trees and recall that the approximation step of our algorithm refines the approximate solution sets of the tree leaves each time it is called. However, the propagation step does not take this into account, and totally recomputes the approximate solution set of the tree root every time it is called, even if approximate solution sets at tree leaves only change very little, and result in a very small or even non-existent change of the tree root.

So we introduce incrementality also in the propagation step. We do not recompute the approximate solution sets at the tree nodes, but keep them in memory, and change them incrementally. At the first sight this seems to be costly in memory usage, but we will see in the next section that this is not the case.

For arriving at an efficient implementation of the resulting algorithm we need to fill out various details. On the one hand we need to describe how to represent approximate solution sets on computers such that one can easily compute them in the first step, and propagate them in the second step. On the other hand we have to state search heuristics such as: How much to refine the approximate solution sets of atomic sub-constraints before doing propagation? Which sub-constraints to refine where? In the following sections we will describe these details.

4 Cylindrical Box Decompositions

One cannot represent and compute arbitrary real numbers on computers, so one also cannot hope to do this with arbitrary approximate solution sets. So we restrict the class of approximate solution sets to a sub-class for which there is a representation that allows an efficient implementation of the algorithm given in the last section. This representation turns out to be a generalization of the notion of cylindrical decomposition, as introduced by G. Collins [7, 3].

Recall that an approximate solution set is a function that assigns either **T**, **F** or **U** to an $x \in \mathbb{R}^n$. Since we cannot represent such functions in general, we restrict ourselves to functions that are constant on finitely many non-empty connected subsets of $\mathbb{R}^n$. We call such a subset of $\mathbb{R}^n$ a *region*, and a finite collection of (not necessarily disjoint) regions whose union is a set B, a *decomposition of B*. So we represent approximate solution sets by functions from a decomposition of $\mathbb{R}^n$ to $\{\mathbf{T}, \mathbf{F}, \mathbf{U}\}$. In this case we say that the approximate solution set is *based on* the according decomposition.

Usually we allow only certain decompositions by fixing a subset of $\bigcup_{i\in\mathbb{N}} 2^{\mathbb{R}^i}$ (a *decomposition structure*). The two most important decomposition structures are the following:

Semi-algebraic sets: Resulting decompositions are called *algebraic decompositions*. This is the decomposition structure used by Collins' method [7, 3].

Real boxes: We call the resulting decompositions *box decompositions*.

From now on we will use the latter decomposition structure, because it allows simple and efficient operations. This makes it very suitable for approximate computation. However, we will see that the class of approximate solution sets representable by box decompositions is still too big to allow efficient propagation. Thus we restrict this class further by imposing additional conditions on the allowed approximate solution sets.

Let us first study the case of a propagation function P_L resulting from a connective L that we apply to approximate solution sets based on decompositions $D_1, \dots, D_n$ resulting in an approximate solution set based on a decomposition D'. As can be seen on the example of equation (1), for connectives the result at a point x depends on the inputs only at x. This means that, if we allow $D_1, \dots, D_n$ to be of arbitrary form, then we need for every non-empty combination of regions from each of the $D_1, \dots, D_n$ a different region in D'. This leads to unacceptable combinatorial explosion. Fortunately we can solve this problem very easily by ensuring that $D_1, \dots, D_n$ are exactly the same . Then D' is also the same as the decompositions $D_1, \dots, D_n$, and we only need to combine the associated truth values according to the connective L.

Now let us study the case of a propagation function P_Q resulting from a quantifier Q that we apply to an approximate solution set based on a decomposition D resulting in an approximate solution set based on a decomposition D'. As can be seen on the example of equation (2), for quantifiers the result at a point x depends not only on the input at x, but on all (x, y), where y in $\mathbb{R}$ (the *points below* x). This means that, if we allow D to be of arbitrary form, then we need for every non-empty projection of combinations of regions in D a different region in D' (in figure 4 for all points x on the x-axis we need to search for all regions that contain points below x). Again this leads to combinatorial explosion and unacceptable theoretical [40, 12] and practical [36] complexity. The problem is even worse for the case of approximate quantifiers, where we need to compute the volume of the true and false regions below each point x.

For ensuring that this problem does not occur, we need:

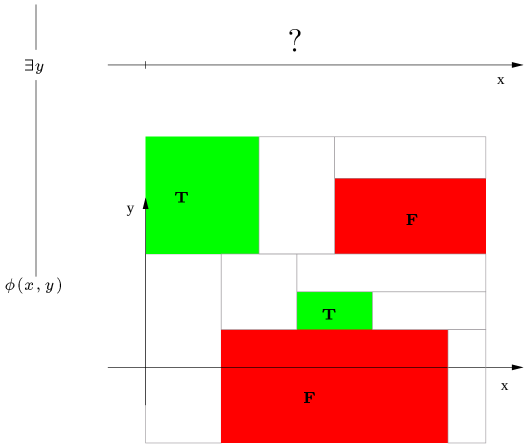

Figure 4: Propagation over Quantifiers

Definition 4 *A* stack below *an n-dimensional region B is a decomposition of $B \times \mathbb{R}$ (of the* cylinder below *B) such that for all x in B, for all regions C in the decomposition there is a point below x that is in C.*

If a subset of the regions in D forms a stack below some region B then the result of applying P_Q is the same for all elements of B, and we can compute it by just combining the approximate truth values of all regions in the stack. Furthermore, for approximate quantifiers, it is very easy to compute the volume of true and false regions in each stack. So, if we ensure that D consists of stacks only (see figure 5), D' consists of the regions below which the stacks are built, and we can easily do computation.

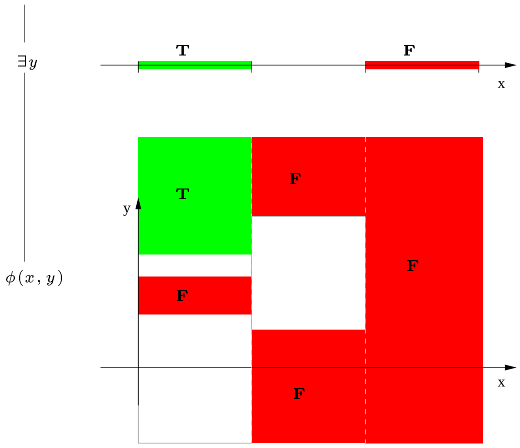

Figure 5: Cylindrical Propagation

Now let us sum up the above:

Definition 5 *For an n-ary logical symbol L and approximate solution sets based on decompositions $D_1, \ldots, D_n$ the* propagation conditions *are fulfilled iff*

- *if L is a logical connective then $D_1 = \cdots = D_n$, and*
- *if L is a quantifier (in this case $n = 1$) then D_1 is a set of stacks.*

Observe that, since all regions are boxes, the condition on quantifiers is just a generalization of the condition on boolean connectives if we view quantifiers as connectives with infinitely many sub-constraints. Now we have to ensure that these propagation conditions are always fulfilled during the propagation process:

Definition 6 *Given a constraint ϕ, an assignment of decompositions to atomic sub-constraints of ϕ is* cylindrical wrt. ϕ *iff, during a propagation of approximate solution sets represented by these decompositions, the propagation conditions are fulfilled for all quantifiers and logical connectives in ϕ.*

Let ϕ' be a sub-constraint of a constraint ϕ, let B be a subset of the free-variable space of ϕ, and let B' be a subset of the free-variable space of ϕ'. Then B' is the *embedding* of B into the free-variable space of ϕ' iff B' is the maximal set that is equal to B on the common variables of ϕ and ϕ'.

When representing approximate solution sets based on cylindrical decomposition assignments one can exploit the cylindricity structure and store everything in one tree.

Definition 7 *Given a constraint ϕ in n free variables, and a box $B \subseteq \mathbb{R}^n$, a* cylindrical decomposition tree of ϕ on B *is a tree with the top-level symbol of ϕ at the root, and*

- *if ϕ is atomic then an approximate truth value of ϕ on B,*
- *if ϕ is Boolean then for each immediate sub-constraint ψ of ϕ a cylindrical decomposition tree of ψ on B as a child, and*
- *if ϕ is quantified on the variable v with immediate sub-constraint ψ then cylindrical decomposition trees of ψ on $B \times C$ labeled with C, where the C form a box decomposition of $\mathbb{R}$, as children.*

This defines a tree like the one in figure 6. Such a tree has a similar structure as the tree representation of constraints, with the difference that, after a new variable is introduced by quantifiers, the tree splits into stacks.

Each cylindrical decomposition tree of a constraint ϕ on a box B represents an assignment of approximate solution sets on the embedding of B into the respective free-variable space to the atomic sub-constraints of ϕ: Each path from the root to a leaf defines an atomic sub-constraint of ϕ with a box in $\mathbb{R}^n$ and an associated approximate truth value.

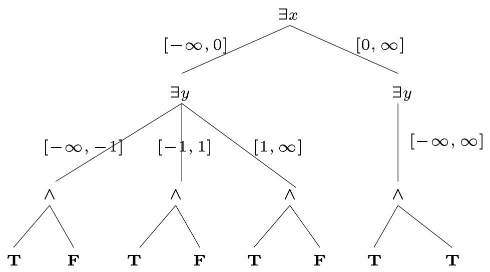

Figure 6: Cylindrical Decomposition Tree

By recursively building according cylindrical decomposition trees (one for each box of the decomposition of the total constraint) we get:

Proposition 1 *Let S be an assignment of approximate solution sets to the atomic sub-constraint of a constraint ϕ, such that the according assignment of decompositions is cylindrical wrt. ϕ. Then S can be represented by a set of cylindrical decomposition trees of ϕ.*

So, by representing the assignment of approximate solution sets to the atomic sub-constraints by cylindrical decomposition trees we can do the propagation step of our algorithm fast. Now we can make important observations:

- By assigning truth values not only to the leaves but to all nodes, we can store approximate solution sets of *all* sub-constraints of a quantified constraint.
- The results of the recursive application of propagation functions P_L can be stored within one and the same tree by just annotating higher and higher levels of nodes with approximate truth values.

As a result we can easily do incremental computation of the approximate solution sets of all sub-constraints of a constraint. It remains to prove that the representation fulfills the propagation conditions.

Proposition 2 *Given a constraint ϕ and a cylindrical decomposition tree of ϕ on B, the represented assignment of decompositions to the atomic sub-constraint of ϕ is cylindrical wrt. ϕ.*

Proof. We prove that during the recursive application of the propagation functions P_L to a cylindrical decomposition tree according to the structure of ϕ the propagation conditions are fulfilled. We already know that the results of the recursive application of P_L can be stored within the tree by just annotating higher and higher levels of nodes with approximate truth values. So we prove that for any level of nodes corresponding to a symbol L and decompositions represented by one level deeper, the propagation conditions are fulfilled. We have the following cases:

- The symbol L is a connective. Then $D_1 = \cdots = D_n$ because all paths from the root of the tree to corresponding leaves pass through the same interval labels.
- The symbol L is a quantifier. Then each child of a corresponding node represents a stack.

Note that the definition of cylindrical decomposition tree can easily deal with quantifiers in blocks (i.e., quantifiers that quantify more than one variable). This means that a tree node can contain a quantifier block and the children of the corresponding node have box labels of the same dimension as the quantifier block has variables instead of interval labels.

Although we have done the above development only for box decompositions, it can be easily extended to different decomposition structures. Now let us compare it with the notion of cylindrical decomposition as introduced by George Collins [7, 3]:

Definition 8 *A decomposition D is* cylindrical *iff $r = 1$ and D is a stack below $\mathbb{R}^0$, or $r > 1$ and there is a cylindrical decomposition D' of $\mathbb{R}^{r-1}$ such that for each region R of D', some subset of D is a stack below R.*

The main difference is that, in contrast to Collins' definition, our notion takes the detailed tree structure of constraints into account. The consequence is that for each cylindrical decomposition there is a corresponding cylindrical assignment of decompositions to the atomic sub-constraints of a constraint. In the other direction, only certain assignments of decompositions to the atomic sub-constraints of a constraint correspond to cylindrical decompositions directly. We arrive at these assignments by enforcing certain conditions on constraints:

- They are in prenex form — then all assigned decompositions are equal.
- All quantifiers quantify exactly one variable (i.e., no quantifier blocks) — then all assigned decompositions are cylindrical.

By using a more general definition that can take the tree structure of constraints into account we gain additional efficiency and flexibility. For example, for the constraint $\exists y\ \phi_1(x, y) \ \wedge\ \exists y\ \phi_2(x, y)$ we can create different decompositions for both existential quantifiers and therefore cut the computation tree by utilizing dependencies between both branches (Collins' algorithm could be started to eliminate the two quantifiers separately, but this would not take into account dependencies between the branches; see [8] for an important step in a similar direction).

5 Formal Algorithm

In this section we formalize the algorithm developed so far and introduce a few more optimizations that arise from the introduced algorithmic details and the use of cylindrical decomposition trees. Readers not interested in algorithmic details can safely skip this section.

First we define the partial order $\succ$ ("contains more information") of figure 7 on approximate truth values. This defines a maximum operation $\cup$ on some (but not all) pairs of approximate truth values. This maximum operation naturally extends to approximate solution sets for which the maximum is defined on each element.

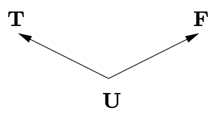

Figure 7: Order on Approximate Truth Values

The algorithm works on the following data:

Definition 9 *Given a constraint ϕ in n free variables,* $\mathbf{S}$ *is a* structural approximate solution set *of ϕ on a set $B \subseteq \mathbb{R}^n$, iff it is a function from all sub-constraints ϕ' of ϕ to approximate solution sets of ϕ' on the embedding of B into the free-variable space of ϕ'. A* structural approximate solution set *of a constraint ϕ is a structural approximate solution set of ϕ on the free-variable space of ϕ.*

We denote the set of all structural approximate solution sets of a constraint ϕ on a set B by $\mathbf{S}_{\phi,B}$. Now we model the operation on structural approximate solution sets that corresponds to taking sub-trees of cylindrical decomposition trees.

Definition 10 *Given a structural approximate solution set $\mathbf{S} \in \mathbf{S}_{\phi,B}$ such that ϕ is non-atomic, $\mathbf{S}'$ is a* restriction of $\mathbf{S}$, *iff $\mathbf{S} \in \mathbf{S}_{\phi',B'}$, $\mathbf{S}$ and $\mathbf{S}'$ coincide on their domain of definition, ϕ' is an immediate sub-constraint of ϕ, and*

- *if ϕ is Boolean, then B' is the embedding of B into the free variable-space of ϕ', and*
- *if ϕ is quantified, then B' is $B \times C$ where $C \subseteq \mathbb{R}$.*

Now we define an abstract data type that models the operations of structural approximate solution sets needed for our algorithm:

trivial: For a closed constraint ϕ and a sub-box B of the free-variable space of ϕ, $\text{trivial}(\phi, B)$ is the trivial structural approximate solution set of ϕ on B.

refine: For $\mathbf{S} \in \mathbf{S}_{\phi,B}$ where ϕ is atomic, $\text{refine}(\mathbf{S}) \in \mathbf{S}_{\phi,B}$, and:

- the error of $\text{refine}(\mathbf{S})(\phi)$ is smaller than the error of $\mathbf{S}(\phi)$, and
- for any $\varepsilon \in \mathbb{R}^+$ there is an n, such that the error of $\text{refine}^n(\mathbf{S})(\phi)$ is smaller than ε.

prop: For $\mathbf{S} \in \mathbf{S}_{\phi,B}$, $\text{prop}(\mathbf{S}) \in \mathbf{S}_{\phi,B}$, and $\text{prop}(\mathbf{S}) = \mathbf{S}$ except $\text{prop}(\mathbf{S})(\phi) = \mathbf{S}(\phi) \cup P_L(\mathbf{S}(\phi_1), \ldots, \mathbf{S}(\phi_n))$, where L is the top-level symbol of ϕ and $\phi_1, \ldots, \phi_n$ are the immediate sub-constraints of ϕ.

chooseRestr: For $\mathbf{S} \in \mathbf{S}_{\phi,B}$ where ϕ is not atomic, $\text{chooseRestr}(\mathbf{S}) \in \mathbf{S}_{\phi',B'}$, and $\text{chooseRestr}(\mathbf{S})$ is a restriction of $\mathbf{S}'$.

embedRestr: For $\mathbf{S} \in \mathbf{S}_{\phi,B}$, $\mathbf{S}' \in \mathbf{S}_{\phi',B'}$, such that $\mathbf{S}'$ is a restriction of $\mathbf{S}$, $\text{embedRestr}(\mathbf{S}, \mathbf{S}') \in \mathbf{S}_{\phi,B}$, and $\text{embedRestr}(\mathbf{S}, \mathbf{S}')$ is equal to $\mathbf{S}$ except that it is equal to $\mathbf{S}'$ on its domain of definition.

solution: For $\mathbf{S} \in \mathbf{S}_{\phi,B}$, and a sub-constraint ϕ' of ϕ, $\text{solution}(\mathbf{S}, \phi') = \mathbf{S}(\phi')$.

One can implement the needed abstract data type very easily by sets of cylindrical decomposition trees. For example, the function `chooseRestr` chooses a sub-tree, `embedRestr` replaces a sub-tree by another one, and `refine` splits one tree into several ones.

The result is algorithm 1. It has a more general output specification than the original one (figure 1), which can then be easily computed. The algorithm refines a structural approximate solution set $\mathbf{S}$ for which we can pass the trivial approximate solution set initially. For implementing the correct termination condition of our specification we close open constraints with an *error quantifier* for which the application of the according propagation function to an approximate solution set S returns $\mathbf{T}$ if the known volume of S is less than a certain ε and $\mathbf{U}$ otherwise.

By taking into account all the possibilities for incremental computation mentioned in section 3 the two steps from our initial algorithm are now

$$\mathbf{S}' \leftarrow \mathrm{Refine}(\mathbf{S})$$

Inputs:
$\mathbf{S}$: a structural approximate solution set of a constraint ϕ on a set B

Outputs:
$\mathbf{S}'$: a structural approximate solution set of ϕ on B, such that $\mathbf{S}' \succ \mathbf{S}$

do
if ϕ is atomic **then**
$\mathbf{S}' \leftarrow \mathrm{refine}(\mathbf{S})$
else
$\mathbf{S}' \leftarrow \mathrm{chooseRestr}(\mathbf{S})$
$\mathbf{S}' \leftarrow \mathrm{Refine}(\mathbf{S}')$
$\mathbf{S}' \leftarrow \mathrm{prop}(\mathrm{embedRestr}(\mathbf{S}, \mathbf{S}'))$
until $\mathbf{S}' \succ \mathbf{S}$

Algorithm 1: Refine

totally intermingled — the approximation step appears in the first part and the propagation step in the second part of the if-then-else construct.

Depending on the implementation of the function `chooseRestr` we arrive at different strategies. As long as `chooseRestr` is fair (in the sense that each sub-constraint and subset will eventually be chosen), then for constraints containing approximate quantifiers and exact constants only, each approximate solution set will eventually attain an arbitrary small positive error (see [43]). So in this case the algorithm always terminates. For constraints containing classical quantifiers or interval constants, it is still correct but does not necessarily terminate.

One can improve the algorithm further by introducing finer control of the termination condition of the loop. By staying in the loop longer one we do not need to choose a new restriction higher in the recursion tree. By leaving the loop earlier, one can try to choose different restrictions on which one suspects more efficient computation to be possible.

6 Implementation

As already mentioned, we have implemented the method for the symbols $+$, $\times$, $\leq$, $<$, $=$ and $\neq$ and the resulting package is publicly available [42]. The implementation uses floating point interval arithmetic (see e.g. [35, 1, 45, 38, 19, 41]) to approximate the according decomposition structure. One could easily extend this structure into a decomposition structure by adding rational number boxes or multi-precision floating point boxes, but practice has shown that the case when this is needed usually does not occur in practice — so we just return an error message when it occurs.

The implementation is based upon the STURM library, developed by H. Hong and others at RISC-Linz [28] and an implementation of the older method by A. Neubacher and H. Hong [27]. It can produce both numerical and graphical output of the computed approximate solution sets.

Currently we require that, in addition to the input constraint ϕ in n free variables and error bound ε, the user provides a starting box $B \subseteq \mathbb{R}^n$ of finite volume. Then we compute an approximate solution set of ϕ on B. Furthermore we also require that the range of each quantified variable is also restricted by such a box. This is only a minor restriction, because a user usually knows reasonable ranges for all the occurring variables from the application domain. Future versions will be extended to allow boxes of infinite volume also.

We use the tightening method of Hong and Stahl [29] combined with range computations for computing approximate solution sets of atomic constraints. Tightening takes a polynomial p, a box B and an integer v, and divides B along the variable v into pieces, returning two sets of boxes: One for which p is sign-invariant on each element, and one for which this is not the case. For the sign-invariant boxes we can easily detect the truth of the according atomic constraint. For the other boxes we continue by either doing tightening along another variable or doing a bisection and range tests on the pieces.

Our implementation uses the strategies that `chooseRestr` chooses child labeled with the highest volume for quantified constraints and it chooses a child with unknown approximate truth value that was chosen least often in the past for Boolean constraints.

Furthermore, as in the older method [27], before starting the algorithm we do simple symbolic simplifications on the input constraint, such as substituting constants, pushing quantifiers to the inside and so on.

7 Efficiency Considerations and Timings

In contrast to the classical exact algorithms the efficiency of the new algorithm critically depends on the fact whether the input describes a well-conditioned, well-posed or ill-posed problem, or in other words, whether small changes of the input result in small changes of the output. If the resulting changes of the output can be infinite, then the termination of the algorithm is not guaranteed [43], that is, we have an ill-posed problem. If the resulting changes are finite, but big, then the computation time is high, that is, we have an ill-conditioned problem.

If the input contains approximate instead of classical quantifiers, then the problem is always well-posed [43]. So we can make an ill-posed or ill-conditioned problem well-posed (i.e., we can *regularize* [13] the problem) by replacing the classical quantifiers with approximate ones or increasing the width of the annotation interval of the approximate quantifiers. A second critical parameters for the efficiency is, of course, the user-provided error bound. As a result a user can always decide whether to prefer a very precise result (by choosing a small error bound and small annotation widths of the approximate quantifiers), or fast results.

Table 7 presents detailed timings. They have been done on a 450MHz, Intel PIII processor, running Debian/GNU Linux. Memory usage has been restricted to 64MB, and running time to 20 minutes. Runs failing due to excessive run-time are marked by ∞_T and runs failing due to excessive memory requirements are marked by ∞_S. Runs finishing in less then one second (in this case start-up times highly influence the precision of the timings) are marked with ε.

Example	Vars	Deg.	QEPCAD	AQCS-old	AQCS	AQCS-AQ
MC6	3	2	2.6	∞_T	∞_T	ε
MC22	3	4	13.0	ε	ε	ε
MC23	3	4	2.3	ε	ε	ε
MC24	3	4	903.4	ε	ε	ε
L1	3	4	∞_T	∞_S	5.26	ε
L2	3	4	∞_T	45.9	ε	ε
LM	4	6	ε	∞_T	∞_T	16.8
HLS44	3	12	∞_T	∞_T	∞_T	ε
MAC	6	12	∞_T	∞_T	∞_T	∞_T
CD	5	6	6.2	∞_T	ε	ε
CF	5	8	∞_T	∞_T	186.3	107.7
C7	4	3	34.3	2.9	ε	ε
C12B	5	5	∞_S	∞_T	5.8	5.8
C19	6	6	∞_T	∞_T	19.3	11.8
C21	8	16	∞_S	∞_T	35.0	42.9

Figure 8: Timings

We compared the following programs:

QEPCAD: Package for "Quantifier elimination by partial cylindrical algebraic decomposition" by H. Hong and others. No special flags used (for controlling heuristics, adaption to problem class etc.)

AQCS-old: Older method for approximate quantified constraint solving by H. Hong and A. Neubacher [23, 27]

AQCS: New method described in this paper.

AQCS-AQ: New method, with classical quantifiers replaced by approximate quantifiers with the annotation interval $[0, 0.01]$.

All the examples contain only inequality predicate symbols and no equalities. They have the following origins:

Computational Geometry: MC6, MC22, MC23, M24 [33].

Stability of Differential Equations: HLS [26], MAC [24]

Control Theory: CD [11], CF [14], C7, C12B, C19, C21 (constructed analogously to CD).

Please note that the comparison to exact programs like QEPCAD only makes limited sense, because our approximate algorithm can prove its usefulness mainly in interactive operation: Depending on the intentions of the user, a higher or lower error bound, or inspections of different parts of the solution set might prove useful.

The conditioning of the problem can be very well read off by comparing the times of AQCS with classical quantifiers and AQCS with approximate quantifiers. If the first number is ∞ and the second small, then approximate quantification made the problem well-posed. If both are big then the problem lies in the overall efficiency of AQCS, which is only the case for the examples MAC, and — to limited degree — LM.

8 Conclusion

We have developed an algorithm for approximate quantified constraint solving and its efficient implementation. The algorithm works by incrementally refining an initial approximation. This is done by propagating approximate solution sets over the tree structure of constraints. To this end we have generalized the notion of cylindrical decomposition [7, 34, 30] and have introduced an efficient tree representation of the result.

The algorithm has several advantages: It can give approximate information even for the undecidable case of quantified constraints that contain transcendental functions, it is efficient on well-conditioned problems even for high precision, it allows the user to decide on the trade-off between efficiency and precision, it can deal with input that contains interval constants, it gives simple numerical output, and it allows reasoning about the size of solution sets using approximate quantifiers.

Furthermore the algorithm makes it easy to interactively increase the precision or to get partial results when interrupting the algorithm before its end [18, 47, 4]. Thus it is very well suited for interactive exploration of the solution space, which seems to support the natural approach of humans very well.

Future work will concentrate on efficiency improvements, such as:

- Devising computation strategies based on heuristics, as done by H. Hong for quantifier elimination by cylindrical algebraic decomposition [21, 25].

- Extending the algorithm in such a way that it can efficiently deal with equalities also, for example by introducing a notion of approximate equality.

Furthermore we will provide an interactive graphical front-end for supporting the exploration of the solution space.

The author thanks Chris Brown for help with porting QEPCAD, Andreas Neubacher for taking time to carefully answer questions about his software, Andreas Neubacher and Richard Liska for providing a lot of interesting application examples, and Hoon Hong for the initial motivation for studying approximate quantified constraint solving. This work was supported by the Austrian Science Fund FWF in the frame of the project SFB F1303.

References

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

[2] H. Anai and V. Weispfenning. Deciding linear-trigonometric problems. In *Proc. ISAAC 2000, International Symposium on Symbolic and Algebraic Computation.* ACM Press, 2000.

[3] D. S. Arnon, G. E. Collins, and S. McCallum. Cylindrical algebraic decomposition I: The basic algorithm. *SIAM Journal of Computing*, 13(4):865–877, 1984. Also in [6].

[4] M. Beltran, G. Castillo, and V. Kreinovich. Algorithms that still produce a solution (maybe not optimal) even when interrupted: Shary's idea justified. *Reliable Computing*, 3(3):39–53, 1997.

[5] F. Benhamou and F. Goualard. Universally quantified interval constraints. In *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming (CP'2000)*, Lecture Notes in Computer Science, Singapore, 2000. Springer Verlag.

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

[7] 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 [6].

[8] G. E. Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12:299–328, 1991. Also in [6].

[9] G. E. Collins and W. Krandick. A hybrid method for high precision calculation of polynomial real roots. In *Proceedings of the 1993 International Symposium on Symbolic and Algebraic Computation ISSAC'93*, pages 47–52, Kiev, Ukraine, 1993. ACM Press.

[10] P. Dorato, W. Yang, and C. Abdallah. Robust multi-objective feedback design by quantifier elimination. *Journal of Symbolic Computation*, 24:153–159, 1997.

[11] P. Dorato, W. Yang, and C. Abdallah. Robust multi-objective feedback design by quantifier elimination. *Journal of Symbolic Computation*, 24:153–159, 1997.

[12] H. Edelsbrunner. A new approach to rectangle intersections I, II. *Internat. J. Comput. Math.*, 13(3–4):209–219, 221–229, 1983.

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

[14] G. Fiorio, S. Malan, M. Milanese, and M. Taragna. Robust performance design of fixed structure controllers with uncertain parameters. In *Proceedings of the 32nd IEEE Conf. Decision and Control*, 1993.

[15] 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.

[16] J. Garloff and B. Graf. Solving strict polynomial inequalities by Bernstein expansion. In N. Munro, editor, *The Use of Symbolic Methods in Control System Analysis and Design*, pages 339–352. The Institution of Electrical Engineers (IEE), London, 1999.

[17] 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.

[18] J. Grass. Reasoning about computational resource allocation — an introduction to anytime algorithms. *ACM Crossroads*, 3(1), 1996. `http://www.acm.org/crossroads/xrds3-1/racra.html`.

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

[20] H. Hong. *Improvements in CAD-based Quantifier Elimination.* PhD thesis, The Ohio State Universitiy, 1990.

[21] H. Hong. Heuristic search strategies for cylindrical algebraic decomposition. In J. Calmet et al., editors, *Proceedings of Artificial Intelligence and Symbolic Mathematical Computing, Springer Lecture Notes in Computer Science 737*, pages 152–165, 1992.

[22] 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.

[23] 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.

[24] H. Hong. The exact region of stability for MacCormack scheme. *Computing*, 56(4), 1996.

[25] H. Hong. Heuristic search and pruning in polynomial constraints satisfaction. *Ann. Math. Artif. Intell.*, 19(3–4):319–334, 1997.

[26] H. Hong, R. Liska, and S. Steinberg. Testing stability by quantifier elimination. *Journal of Symbolic Comptutation*, 24(2):161–187, 1997.

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

[28] H. Hong, A. Neubacher, and V. Stahl. The STURM library manual – a C++ library for symbolic computation. Technical Report 94-30, RISC Linz, 1994.

[29] H. Hong and V. Stahl. Safe starting regions by fixed points and tightening. *Computing*, 53:323–335, 1994.

[30] M. Jirstrand. Cylindrical algebraic decomposition - an introduction. Technical report, Automatic Control group, Linköping, 1985.

[31] M. Jirstrand. Nonlinear control system design by quantifier elimination. *Journal of Symbolic Computation*, 24(2):137–152, 1997.

[32] J. R. Johnson. Real algebraic number computation using interval arithmetic. In *International Conference on Symbolic and Algebraic Computation (ISSAC'92)*, pages 195–205, Berkeley, CA USA, 1992. ACM Press.

[33] S. McCallum. Solving polynomial strict inequalities using cylindrical algebraic decomposition. *The Computer Journal*, 36(5):432–438, 1993.

[34] B. Mishra. *Algorithmic Algebra*. Springer Verlag, 1993.

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

[36] A. Neubacher. Oral Communication.

[37] A. Neubacher. *Parametric Robust Stability by Quantifier Elimination*. PhD thesis, Research Institute for Symbolic Computation - Universität Linz, October 1997.

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

[39] P. Pau and J. Schicho. Quantifier elimination for trigonometric polynomials by cylindrical trigonometric decomposition. *Journal of Symbolic Computation*, 29(6), June 2000.

[40] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer Verlag, 1985.

[41] 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.

[42] S. Ratschan. Approximate quantified constraint solving (AQCS). `http://www.risc.uni-linz.ac.at/research/software/AQCS`, 2000. Software package.

[43] S. Ratschan. Convergence of approximate real constraint solving by approximate quantifiers. Technical Report 00-23, Research Institute for Symbolic Computation (RISC) - Linz, 2000. Submitted to CONSTRAINTS.

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

[45] H. Ratschek and J. Rokne. *New Computer Methods for Global Optimizations*. Ellis Horwood Limited, 1988.

[46] D. Richardson. Some undecidable problems involving elementary functions of a real variable. *Journal of Symbolic Logic*, 33:514–520, 1968.

[47] S. P. Shary. On optimal solution of interval linear equations. *SIAM Journal on Numerical Analysis*, 32:610–630, 1995.

[48] S. P. Shary. Outer estimation of generalized solution sets to interval linear systems. *Reliable Computing*, 5:323–335, 1999.

[49] A. Tarski. *A Decision Method for Elementary Algebra and Geometry.* Univ. of California Press, Berkeley, 1951. Also in [6].

[50] K. Weihrauch. *Introduction to Computable Analysis.* Texts in Theoretical Computer Science. Springer Verlag, 2000.

[51] V. Weispfenning. The complexity of linear problems in fields. *Journal of Symbolic Computation*, 5(1–2):3–27, 1988.