Efficient Solving of Quantified Inequality Constraints over the Real Numbers*

Stefan Ratschan
Stefan.Ratschan@mpi-sb.mpg.de

December 8, 2018

Abstract

Let a quantified inequality constraint over the reals be a formula in the first-order predicate language over the structure of the real numbers, where the allowed predicate symbols are $\leq$ and $<$. Solving such constraints is an undecidable problem when allowing function symbols such sin or cos. In the paper we give an algorithm that terminates with a solution for all, except for very special, pathological inputs. We ensure the practical efficiency of this algorithm by employing constraint programming techniques.

1 Introduction

The problem of solving quantified constraints over the reals has numerous applications (we have created a web-page that lists more than fifty references [39]). However, it is undecidable, when allowing function symbols such as sin or cos [53], and highly complex when restricting oneselves to addition and multiplication [18, 59].

In this paper we give an algorithm that nevertheless always terminates for inputs that are stable in the sense that their truth value (in the case where all variables are quantified) does not change under small perturbations of the occurring constants. For example, the constraint $\forall x\ x^2+1 \geq 0$ is stable, whereas $\forall x\ x^2 \geq 0$ is not.

Furthermore, we ensure the practical efficiency of the algorithm by basing it on techniques from the field of constraint programming [8, 9, 49, 55]. The basic idea of these techniques is to reduce the average run-time of algorithms for computationally hard problems by replacing expensive exhaustive search as much as possible by methods for pruning elements from the search space for which it is easy to show that they do not contain solutions.

In this paper we extend this idea to quantified inequality constraints for which all (free and bound) variables are bounded to a closed interval: We try to prune elements from these bounds for which it is easy to show that they

*This is a revised and extended version of an earlier paper [42]. Please note the changed terminology that should contribute to a wider accessibility of the results.

do not contain solutions. When we cannot easily prune more elements, we do branching by splitting a bound into pieces (for quantified variables this means replacing sub-constraints of the form $\forall x \in I\ \phi$ by $\forall x \in I_1\ \phi\ \wedge\ \forall x \in I_2\ \phi$ where $I = I_1 \cup I_2$, or the corresponding existential case). This gives us new possibilities for pruning. We repeat the two steps until we have pruned all elements (or disproved the constraint). For computing elements of the bounds that do contain solutions we take the negation of the input constraint and again apply the above branch-and-prune approach.

In the paper we formalize this approach, study its properties in detail, improve it for an implementation, and do timings that show its efficiency.

As a side-effect, this paper even improves the current methods for numerical constraint satisfaction problems in the case where the solution set does not consist of finitely many, isolated solutions, which—up to now—was essential for their efficiency. For example, the book describing the system Numerica [56] explicitly states that for inputs not fulfilling that property the method creates a huge number of boxes.

In order to be able to reuse existing theory, algorithms and software for solving atomic inequality constraints (i.e., constraints of the form $t \geq 0$ or $t > 0$, where t is a term), and to be able to benefit from further progress in this area, the paper employs a parametric approach: It takes as input theory and algorithms from constraint programming, and provides as output corresponding new theory and algorithms for solving quantified constraints. More specifically, building upon the notion of a narrowing operator [55, 7, 25], it takes as input: a specification describing a consistency notion for atomic inequality constraints (e.g., box-consistency [8]), and a narrowing operator that implements this specification. It provides as output: a specification describing a corresponding consistency notion for quantified constraints, a narrowing operator that implements this specification, and an algorithm for computing approximate solutions of quantified constraints over the reals that uses this narrowing operator for pruning. These outputs are accompanied with proofs of their usefulness/optimality.

For the special case where the only allowed function symbols are addition and multiplication, up to recently, all algorithms have been based on computer algebra methods [14, 11], which resulted in certain drawbacks (e.g., low practical efficiency, restriction to polynomials, unwieldy output expressions). In an earlier paper [41] the author of this paper proposed a scheme for solving quantified constraints approximately that followed the idea of quantifier elimination by cylindrical algebraic decomposition [14, 11], but decomposed space into floating-point boxes instead of semi-algebraic cells. This approach was successful in showing that one can efficiently compute approximate solutions of quantified constraints using interval methods. However it still had several drawbacks. Especially, it was not clear when and how to optimize box splitting, because the algorithm was not separated into (inherently exponential) search, and pruning. The current paper provides a solution to this, and other, problems of the older approach.

The following special cases of the general problem have been studied using interval or constraint satisfaction methods:

- The case of expressions of the form $\forall \vec{p}\ \phi(\vec{p}, \vec{q})$, where $\phi(\vec{p}, \vec{q})$ is a system of strict inequalities [28, 33, 27] using methods that repeatedly bisect the

free-variable space, and test after each bisection, whether the system of inequalities holds everywhere on the resulting box.

- The case of expressions of the form $\forall p\ \phi(p, \vec{q})$, where $\phi(p, \vec{q})$ is a system of strict inequalities [6], using methods that correspond to our case for universal quantification, conjunction and atomic constraints, but without branching in the universally quantified variables.

- The case of quantified systems of equations where certain variables occur only once, and the quantifiers obey certain orderings (various results by S. Shary, see just for example [51]).

- The case of disjunctive constraints. This has been done in the discrete case [29, 57], and in the continuous case for disjunctive constraints occuring in interactive graphical applications [34], and in a similar way as in this paper for speeding up solving of factorizable constraints [24].

See also an overview on methods for solving quantified inequalities in control [19]. For improving box splitting strategies for inequality constraints Silaghi, Sam-Haround and Faltings [52] use information from the negation of the input constraints. A similar problem is the problem of extending the bounds of universal quantifiers, for which a method based on constraint propagation [13] is provided for the special case of a system of inequalities for which all variables are universally quantified. Also for discrete domains there is a lot of recent interest solving constraints with quantifiers [10, 21], or related stochastic constraints [58].

Some of the above [13, 6, 52] take a similar approach of using the negation of the input to compute positive information. However, they do not address the question of being able to compute answers for all except unstable inputs.

The content of the paper is as follows: Section 2 gives various preliminaries; Section 3 introduces a framework for reusable pruning based on the notion of narrowing operator; Section 4 describes an according notion of consistency for quantified constraint that allows us to specify the pruning power of narrowing operators; Section 5 gives a generic algorithm for pruning that implements a narrowing operator; Section 6 bases an according branch-and-prune solver for quantified constraints on this pruning algorithm. Section 7 discusses how to arrive at an efficient implementation of such a solver; Section 8 presents timings of such an implementation; Section 9 discusses the relation of the results to symbolic quantifier elimination algorithms; and Section 10 concludes the paper.

2 Preliminaries

We fix a set V of variables. A quantified constraint (or short: constraint) is a formula in the first-order predicate language over the reals with predicate and function symbols interpreted as suitable relations and functions, and with variables in V. We take over a large part of the according predicate-logical terminology without explicit definitions.

In this paper we restrict ourselves to the predicate symbols $<$, $>$, $\leq$, and $\geq$, and assume that equalities are expressed by inequalities on the residual

(i.e., $f = 0$ as $|f| \leq \varepsilon$ or $f^2 \leq \varepsilon$, where ε is a small positive real constant[1]). Furthermore we only deal with constraints without negation symbols because one can easily eliminate negation symbols from quantified constraints by pushing them down, and replacing atomic constraints of the form $\neg(f \leq g)$ by $f > g$, and $\neg(f < g)$ by $f \geq g$, respectively. For any quantified constraint ϕ, let $\dot{\neg}\phi$ (the *opposite of* ϕ) be the quantified constraints that results from $\neg\phi$ by eliminating the negation by pushing it down to the predicates.

We require that every quantifier be bounded by an associated *quantifier bound*, using expressions of the form $\exists x \in I$ or $\forall x \in I$, where I is a closed interval.

A *variable assignment* is a function from the set of variables V to $\mathbb{R}$. We denote the semantics of a constraint ϕ, the set of variable assignments that make ϕ true, by $[\![\phi]\!]$. For example, $[\![x^2 + y^2 \leq 1]\!]$ is the set of variable assignments that assign values within the unit disc to x and y.

For any variable assignment d, any variable $v \in V$ and any real number r, we denote by $d\frac{r}{v}$ the variable assignment that is equal to d except that it assigns r to v.

Let $\mathbb{I}$ be the set of closed real intervals. We denote by $I_1 \uplus I_2$ the smallest interval containing both intervals I_1 and I_2. A *box assignment* is a set of variable assignments that can be represented by functions from V to $\mathbb{I}$; that is, it contains all the variable assignments that assign elements within a certain interval to a variable. For example, for $V = \{x, y\}$, the set of variable assignments that assign an element of $[-1, 1]$ to both x and y is a box assignment—this set of variable assignments can be represented by the function that assigns $[-1, 1]$ to both x and y.

From now on we will use a box assignment and its interval function representation interchangeably. For any box assignment B, any variable $v \in V$, and any interval I, we denote by $B\frac{I}{v}$ the box assignment that is equal to B except that it assigns I to v. In the context of closed constraints we denote by the Boolean value **F** the empty box assignment and by the Boolean value **T** the box assignment that assigns the set of real numbers $\mathbb{R}$ to each variable, and allow the usual Boolean operation on them. We denote by $\{x \mapsto [-1, 1], y \mapsto\}$ a box assignment that assigns the interval $[-1, 1]$ to the variable x and an arbitrary interval to the variable y.

Traditionally, constraint programming techniques [8, 9, 49, 55] use boxes (i.e., Cartesian products of intervals) instead of box assignments. However, when working with predicate logic, the additional flexibility of box assignments is very convenient in dealing with the scoping of variables. For efficiency reasons, an actual implementation might represent box assignments by boxes.

3 A Framework for Reusable Pruning

Remember that our approach will be to solve quantified constraints by a branch and prune algorithm. Fortunately, there is already a lot of work done on how to do pruning on atomic constraints, and their conjunctions. The formal framework for this is the notion of a narrowing operator [5, 9], which specifies some

[1] The constant ε needs to be non-zero because otherwise solutions would vanish under small perturbations of ε, resulting in an unstable [44] constraint.

properties required of such an algorithm without regard to the concrete algorithm. In this paper we generalize this notion to quantified constraints. This will allow us to reuse existing theory, algorithms, and software implementing such narrowing operators. Readers who are only interested in a concrete pruning algorithm and not in a formal framework for reasoning about its properties can directly jump to Section 5.

The essential difference between our approach and the classical one is that quantified constraints also store information about the range of variables within the constraint, and so we allow a narrowing operator to modify constraints also. For this we use pairs (ϕ, B), where ϕ is a quantified constraint and B is a box assignment. We call such pairs *bounded constraints*, and the second element of a bounded constraint its *free-variable bound*.

Definition 1 *A* narrowing operator *is a function N on bounded constraints such that for bounded constraints (ϕ, B), and (ϕ', B'), and for $N(\phi, B) = (\phi_N, B_N)$, and $N(\phi', B') = (\phi'_N, B'_N)$,*

- *$B \supseteq B_N$ (contractance),*
- *$[\![\phi_N]\!] \cap B_N = [\![\phi]\!] \cap B_N$ (soundness),*
- *$B' \subseteq B$ implies $B'_N \subseteq B_N$ (monotonicity), and*
- *$N(N(\phi, B)) = N(\phi, B)$ (idempotence).*

Note that we use a soundness condition here, instead of a correctness condition: We require that the solution set of the resulting constraint be the same only on the *resulting* box, but not necessarily on the initial box. We will ensure full correctness by the next definition.

Constraint programming techniques for continuous domains traditionally compute outer approximations of the solution set. However, here we also want to compute inner approximations for three reasons: First, we need to compute such inner approximations for proving closed constraints to be true. Second, the solution set of quantified constraints with inequality predicates usually does not consist of singular points, but of sets that can have volume, and for many applications it is important to find points that are guaranteed to be within this solution set. And third, available inner approximations can speed up the computation of outer approximations and vice versa, because any element known to be within the solution set, or known to be not in the solution set, does not need to be inspected further.

So we will allow two kinds of narrowing operators—one that only removes elements not in the solution set, and one that only removes elements in the solution set.

Definition 2 *An* outer narrowing operator *is a narrowing operator $\overline{N}$ such that for every bounded constraint (ϕ, B), for the free-variable bound B_N of $\overline{N}(\phi, B)$, $B_N \supseteq B \cap [\![\phi]\!]$. An* inner narrowing operator *is a narrowing operator $\underline{N}$ such that for every bounded constraint (ϕ, B), for the free-variable bound B_N of $\underline{N}(\phi, B)$, $B_N \supseteq B \setminus [\![\phi]\!]$*[2]*.*

[2]This definition of inner narrowing operator differs from the one used by Benhamou and Goualard [6].

As discussed in the introduction, we get an inner narrowing operator from an outer narrowing operator by working on the opposite of the input:

Theorem 1 *Let $\overline{N}$ be a function on bounded constraints and let $\underline{N}(\phi, B) := (\neg\phi_N, B_N)$ where $(\phi_N, B_N) = \overline{N}(\neg\phi, B)$. Then $\overline{N}$ is an outer narrowing operator iff $\underline{N}$ is an inner narrowing operator.*

Proof. Obviously $\underline{N}$ is a narrowing operator iff $\overline{N}$ is a narrowing operator. $\overline{N}$ is outer narrowing iff $\underline{N}$ is inner narrowing because for any bounded constraint (ϕ, B), $B \cap [\![\neg\phi]\!] = B \setminus [\![\phi]\!]$, and $B \setminus [\![\neg\phi]\!] = B \cap [\![\phi]\!]$. ■

A similar observation has already been used for the special case of quantified constraints with one universal quantifier [6]. The above theorem allows us to concentrate on outer narrowing operators from now on. We get the corresponding inner narrowing operator for free by applying the outer narrowing operator on the opposite of the input.

4 Consistency of Quantified Constraints

In constraint programming the notion of consistency is used to specify the pruning power of narrowing operators. In this section we generalize this approach to quantified constraints. Again, readers who are only interested in a concrete algorithm and not in formal reasoning about its properties, can skip this section.

Clearly, it is not possible to prune the empty set further. So we require the following from a predicate on bounded constraints that we use for such specification purposes:

Definition 3 *A* consistency property *is a predicate C on bounded constraints such that for every constraint ϕ, $C(\phi, \emptyset)$.*

Example 1 *For an atomic bounded constraint (ϕ, B), $BC(\phi, B)$ holds iff there is no (canonical-interval-wide) face of the hyperrectangle described by B for which interval evaluation [36, 37, 27] will prove that it contains no element of $[\![\phi]\!]$. In this case we say that ϕ is* box-consistent *wrt. B [8, 55].*

The strongest form of consistency achievable using floating-point numbers is:

Example 2 *For a bounded constraint (ϕ, B), $HC(\phi, B)$ holds iff there is no box assignment B' with floating-point endpoints such that $B' \subset B$ and $[\![\phi]\!] \cap B' = [\![\phi]\!] \cap B$. In this case we say that ϕ is* hull-consistent *wrt. B [9].*

Note that in the constraint programming literature [12, 5, 8], the definition of hull-consistency usually assumes, that the according constraints have been decomposed into so-called *primitive* constraints. We do *not* follow this approach here, because that would blur the borderline between consistency properties and symbolic preprocessing of constraints.

The following is the strongest form of consistency that does not result in loss of information, that is, for which an outer narrowing operator exists.

Definition 4 *For a bounded constraint (ϕ, B), $TC(\phi, B)$ holds iff there is no box assignment B' such that $B' \subset B$ and $[\![\phi]\!] \cap B' = [\![\phi]\!] \cap B$. In this case we say that ϕ is* tightly consistent *wrt. B.*

Now we can use consistency properties as specifications for the effectiveness of narrowing operators:

Definition 5 *Given a consistency property C, a narrowing operator $\overline{N}$* ensures *C iff for all bounded constraints (ϕ, B), $C(\overline{N}(\phi, B))$ holds.*

Now we assume a certain consistency property on literals (i.e., atomic constraints and their negations) and lift it to a corresponding consistency property on quantified constraints.

Definition 6 *Given a quantified constraint ϕ and a consistency property C on literals, let $\widehat{C}$ be the following predicate:*

- *if ϕ is a literal, then $\widehat{C}(\phi, B)$ iff $C(\phi, B)$*
- *if ϕ is of the form $\phi_1 \wedge \phi_2$ then $\widehat{C}(\phi, B)$ iff $\widehat{C}(\phi_1, B)$ and $\widehat{C}(\phi_2, B)$.*
- *if ϕ is of the form $\phi_1 \vee \phi_2$ then $\widehat{C}(\phi, B)$ iff $B = B_1 \uplus B_2$ where $\widehat{C}(\phi_1, B_1)$ and $\widehat{C}(\phi_2, B_2)$.*
- *if ϕ is of the form $Qy \in I'\ \phi'$, where Q is a quantifier, then $\widehat{C}(\phi, B)$ iff $\widehat{C}(\phi', B\frac{I'}{y})$.*

Theorem 2 *For a consistency property C, $\widehat{C}$ is also a consistency property.*

If for a bounded constraint (ϕ, B), $\widehat{C}(\phi, B)$ holds, we say that (ϕ, B) is *structurally C-consistent*.

Note that, in the above definition, recursion for quantification puts the quantifier bound into the free-variable bound of the quantified constraint. This means that a narrowing operator will also have to modify the quantifier bounds in order to achieve structural consistency.

Example 3 *The bounded constraint $\big(\exists y \in [0,1]\big[x^2 + y^2 \leq 1 \wedge y \geq 0\big], \{x \mapsto [-1,1], y \mapsto\}\big)$ is structurally tightly consistent, and it will be the result of applying an according narrowing operator to an input such as $\big(\exists y \in [-2,2]\big[x^2 + y^2 \leq 1 \wedge y \geq 0\big], \{x \mapsto [-2,2], y \mapsto\}\big)$.*

Note that Definition 6 is compatible with the usual consistency notions for sets of constraints [8, 7]. For example a set of atomic constraints $\{\phi_1, \ldots, \phi_n\}$ is box-consistent wrt. a box assignment B iff $(\phi_1 \wedge \cdots \wedge \phi_n, B)$ is $\widehat{BC}$-consistent. In addition, the method for solving constraints with one universally quantified variable by Benhamou and Goualard [6] computes a special case of Definition 6.

In the following sense, our definition of $\widehat{C}$-consistency is optimal (remember that tight consistency is the strongest possible consistency property).

Theorem 3 *A $\widehat{TC}$-consistent bounded constraint (ϕ, B), where ϕ contains neither conjunctions nor universal quantifiers, is TC-consistent.*

Proof. We proceed by induction over the structure of constraints. The atomic case trivially holds. Now assume constraints of the following types:

- For a TC-consistent bounded constraint of the form $(\phi_1 \vee \phi_2, B)$, by definition, we have $B = B_1 \uplus B_2$ where $TC(\phi_1, B_1)$ and $TC(\phi_2, B_2)$. By the induction hypothesis both (ϕ_1, B_1) and (ϕ_2, B_2) are tightly consistent. So for no box assignment $B_1' \subset B_1$, we have $B_1' \supseteq B_1 \cap [\![\phi_1]\!]$, and for no box assignment $B_2' \subset B_2$, we have $B_2' \supseteq B_2 \cap [\![\phi_2]\!]$. Thus also for no box assignment $B' \subset B_1 \uplus B_2$, we have $B' \supseteq B \cap ([\![\phi_1]\!] \cup [\![\phi_2]\!]) = B \cap [\![\phi_1 \vee \phi_2]\!]$.
- For a $\widehat{TC}$-consistent bounded constraint of the form $(\exists y \in I'\ \phi', B)$, by definition $\widehat{TC}(\phi', B\frac{I'}{y})$. Thus, by the induction hypothesis $(\phi', B\frac{I'}{y})$ is tightly consistent. So for no box assignment $B_p' \subset B\frac{I'}{y}$, $B_p' \supseteq B\frac{I'}{y} \cap [\![\phi']\!]$. As a consequence also for no box assignment $B_p \subset B$, $B_p \supseteq B \cap [\![\exists y \in I'\ \phi']\!]$, and so $(\exists y \in I'\ \phi', B)$ is tightly consistent.

■

The fact that the above theorem does not hold for constraints with conjunctions is well known [9]. It is illustrated in Figure 1, where both ϕ_1 and ϕ_2 are tightly consistent wrt. the box B (i.e. the larger box encloses the ellipses tightly), but $\phi_1 \wedge \phi_2$ is only tightly consistent wrt. the smaller box B' (i.e., the smaller, but not the larger box, encloses the intersection of the ellipses tightly).

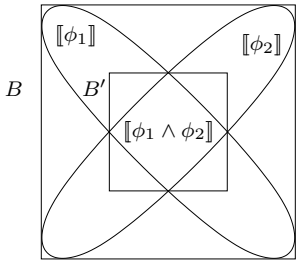

Figure 1: Conjunction—Structural Tight Consistency

For universal quantification there is a similar problem: In Figure 2, ϕ is tightly consistent wrt. the box B, but when considering $\forall y \in I\ \phi$ one can still narrow B horizontally. So any stronger consistency notion would have to treat universal quantification differently from existential quantification.

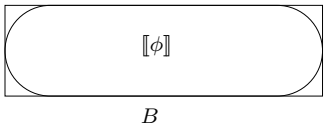

Figure 2: Universal Quantification—Structural Tight Consistency

5 Pruning Algorithm

In this section we give an algorithm for pruning quantified constraints that can use an arbitrary algorithm for pruning atomic constraints. This algorithm fulfills the formal properties introduced in Sections 3 and 4.

The algorithm proceeds recursively according to the structure of constraints. For conjunctions this means the usual: We prune wrt. the individual sub-constraints, until this does not result in any further improvements, that is, until we reach a fix-point. For disjunctions we prune the individual sub-constraints and combine the result by taking the smallest box assignment containing the union.

For existentially quantified bounded constraints of the form $(\exists x \in I\ \phi, B)$ we proceed as shown in Figure 3, where the horizontal axis represents the free-variable bound B (ignoring the component corresponding to the variable x) and the vertical axis the quantifier bound I. Below and to the left of these axis we show the changes on the corresponding elements. We recursively prune the bounded sub-constraint $(\phi, B\frac{I}{x})$ consisting of the sub-constraint ϕ, and the box assignment that is the same as B except that it assigns I to the variable x. We use the result to remove these elements from the free-variable bound B and the quantifier bound I for which recursive pruning showed that they do not contain any solution. This results in the new free-variable bound B' and quantifier bound $B'(x)$.

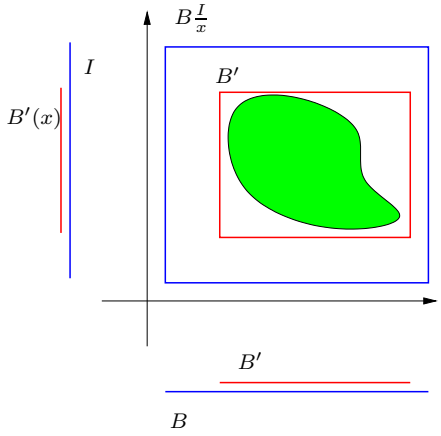

Figure 3: Existential Pruning

For universally quantified bounded constraints of the form $(\forall x \in I\ \phi, B)$, if pruning of the sub-constraints removes elements from I, the whole constraint is false, and we can replace B by the empty set (Figure 4). If no such elements are removed then we just prune B accordingly (Figure 5).

To formalize the above, we let fix be a partial function such that, for a set of functions $\{f_1, \ldots, f_n\}$, for all a, $\text{fix}(\{f_1, \ldots, f_n\})(a)$ is a fixpoint of applying $\{f_1, \ldots, f_n\}$ to a, if such a fixpoint exists, and is undefined, otherwise. Now we have:

Definition 7

- *For atomic* ϕ, $\overline{N}_A(\phi, B) = \overline{A}(\phi, B)$,

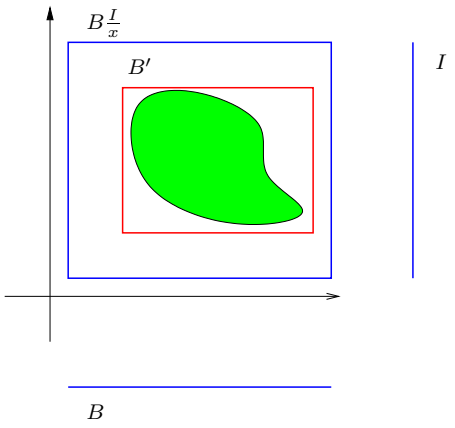

Figure 4: Universal Pruning

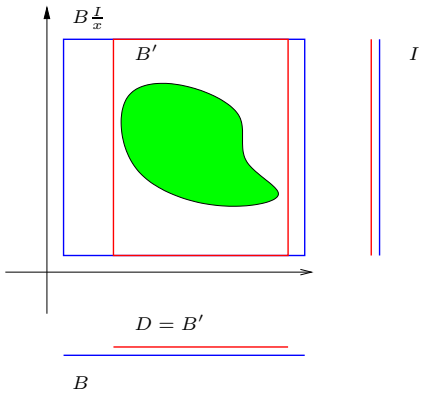

Figure 5: Universal Pruning

- $\overline{N}_A(\phi_1 \wedge \phi_2, B) = \mathit{fix}(\{\overline{N}'_1, \overline{N}'_2\})(\phi_1 \wedge \phi_2, B)$
  where $\overline{N}'_1(\phi_1 \wedge \phi_2, B) = (\phi'_1 \wedge \phi_2, B')$, *where* $(\phi'_1, B') = \overline{N}_A(\phi_1, B)$,
  and $\overline{N}'_2(\phi_1 \wedge \phi_2, B) = (\phi_1 \wedge \phi'_2, B')$, *where* $(\phi'_2, B') = \overline{N}_A(\phi_2, B)$.
- $\overline{N}_A(\phi_1 \vee \phi_2, B) = (\phi'_1 \vee \phi'_2, B'_1 \uplus B'_2)$
  where $(\phi'_1, B'_1) = \overline{N}_A(\phi_1, B)$ *and* $(\phi'_2, B'_2) = \overline{N}_A(\phi_2, B)$
- $\overline{N}_A(\exists x{\in}I\ \phi, B) = (\exists x{\in}B'(\underline{x})\ \phi', B')$,
  where $(\phi', B') = \overline{N}_A(\phi, B\frac{I}{x})$
- $\overline{N}_A(\forall x{\in}I\ \phi, B) = (\forall x{\in}I\ \phi', D)$,
  where $(\phi', B') = \overline{N}_A(\phi, B\frac{I}{x})$
  and $d \in D$ *iff for all* $r \in I$, $d\frac{r}{x} \in B'$

Example 4 *For the input* $\big(\exists y{\in}[-2,2]\big[x^2+y^2 \leq 1 \wedge y \geq 0\big]\,, \{x \mapsto [-2,2], y \mapsto \}\big)$ *already used in Example 3, a narrowing operator based on tight consistency applies itself recursively to* $(x^2 + y^2 \leq 1 \wedge y \geq 0\,, \{x \mapsto [-2,2], y \mapsto [-2,2]\})$. *Repeated applications of the atomic narrowing operator—until a fixpoint is reached—will create the constraint* $(x^2{+}y^2 \leq 1{\wedge}y \geq 0\,, \{x \mapsto [-1,1], y \mapsto [0,1]\})$. *As a final result we get* $\big(\exists y \in [0,1]\big[x^2 + y^2 \leq 1 \wedge y \geq 0\big]\,, \{x \mapsto [-1,1], y \mapsto\}\big)$.

Example 5 *For the input* $(\forall x \in [-2,2]\, x \geq 0, \{x \mapsto\})$, *the algorithm will first narrow* $(x \geq 0, \{x \mapsto [-2,2]\})$ *to* $(x \geq 0, \{x \mapsto [0,2]\})$ *and then create* $(\forall x \in [-2,2]\, x \geq 0, \emptyset)$, *indicating that the constraint is false.*

In the rest of this section we will use the results of Section 3 and 4 for studying the properties of the introduced pruning algorithm. Readers not interested in these formal properties can directly jump to Section 6 for an according solver.

Note that the fixed-point operator fix could result in a partial function, that is, the algorithm could fail to terminate. For ensuring termination, we require that pruning atomic constraints eventually terminates even when intermingled with shrinking of the free-variable bound by other operations:

Definition 8 *A narrowing operator* N *is* finitely contracting *iff there is no infinite sequence* $(\phi_1, B_1), (\phi_2, B_2), \ldots$ *for which for all* $k \in \mathbb{N}$, *for* $(\phi', B') = N(\phi_k, B_k)$, $\phi_{k+1} = \phi'$ *and* B_{k+1} *is a strict subset of* B'.

This property usually holds for practical implementations, because of the finiteness of floating point numbers.

Lemma 1 *If* $\overline{A}$ *is a finitely contracting narrowing operator then* $\overline{N}_A$ *is a total function.*

Proof. We assume that $\overline{A}$ is finitely contracting but $\overline{N}_A$ is not total. This can only happen if $\mathrm{fix}(\{\overline{N}'_1, \overline{N}'_2\})(\phi_1 \wedge \phi_2, B)$ is undefined. Consider the sequence $(\phi^1_1 \wedge \phi^1_2, B_1), (\phi^2_1 \wedge \phi^2_2, B_2), \ldots$ of bounded constraints created by repeated applications of $\overline{N}'_1$ and $\overline{N}'_2$. Here $(\phi^1_1, B_1), (\phi^2_1, B_2), \ldots$ is an infinite sequence as in Definition 8. So $\overline{N}_A$ is not finitely contracting, and by induction also $\overline{A}$ is not finitely contracting—a contradiction. ■

The algorithm fulfills the properties needed by the formal framework of narrowing operators and gives a unique result (despite the non-unique definition of fixpoint operator):

Theorem 4 *For every finitely contracting (atomic) outer narrowing operator $\overline{A}$, $\overline{N}_A$ is a unique outer narrowing operator.*

Proof. Contractance and idempotence hold by easy induction. For proving that $\overline{N}_A$ is outer narrowing, sound and monotonic we proceed by induction. The ground case of atomic constraints holds by definition. Now we have:

- Obviously the composition of two narrowing operators is also a narrowing operator. So, for constraints of the form $\phi_1 \wedge \phi_2$ we just need to show that both $\overline{N}'_1$ and $\overline{N}'_2$ are outer narrowing. For $(\phi'_1, B'_1) = \overline{N}(\phi_1, B)$ and $(\phi'_2, B'_2) = \overline{N}(\phi_2, B)$, by the induction hypothesis $B'_1 \supseteq B \cap [\![\phi_1]\!]$ and $B'_2 \supseteq B \cap [\![\phi_2]\!]$. Thus also $B'_1 \cap B'_2 \supseteq B \cap [\![\phi_1]\!] \cap [\![\phi_2]\!] = B \cap [\![\phi_1 \wedge \phi_2]\!]$. The induction step for soundness and monotonicity is easy.

- For constraints of the form $\phi_1 \vee \phi_2$, for $(\phi'_1, B'_1) = \overline{N}(\phi_1, B)$ and $(\phi'_2, B'_2) = \overline{N}(\phi_2, B)$, by the induction hypothesis $B'_1 \supseteq B \cap [\![\phi_1]\!]$ and $B'_2 \supseteq B \cap [\![\phi_2]\!]$. Thus also $B'_1 \uplus B'_2 \supseteq B'_1 \cup B'_2 \supseteq B \cap ([\![\phi_1]\!] \cup [\![\phi_2]\!]) = B \cap [\![\phi_1 \vee \phi_2]\!]$. The induction step for soundness and monotonicity is easy.

- For constraints of the form $\exists x \in I\ \phi$, the induction step for outer narrowing is easy. For soundness we have to prove that $[\![\exists x \in I\ \phi]\!] \cap B' = [\![\exists x \in B'(x)\ \phi']\!] \cap B'$, where $B' = \overline{N}_A(\phi, B\frac{I}{x})$. Now by the outer narrowing property $B' \supseteq B \cap [\![\phi]\!]$, and so $[\![\exists x \in I\ \phi]\!] \cap B' = [\![\exists x \in B'(x)\ \phi]\!] \cap B'$. This is equal to $[\![\exists x \in B'(x)\ \phi']\!] \cap B$, because by the induction hypothesis $[\![\phi']\!] \cap B' = [\![\phi]\!] \cap B'$.

- For constraints of the form $\forall x \in I\ \phi$, for outer narrowing we have to prove that $D \supseteq B \cap [\![\forall x \in I\ \phi]\!]$, where D is defined as in the corresponding rule of Definition 7. So we assume a variable assignment d that is both in B and $[\![\forall x \in I\ \phi]\!]$, and prove that $d \in D$. This means that we have to prove that for all $r \in I$, $d\frac{r}{x} \in B'$, where $(\phi', B') = \overline{N}_A(\phi, B\frac{I}{x})$. This is clearly the case by the semantics of universal quantification and the induction hypothesis.

  For soundness we have to prove that $[\![\forall x \in I\ \phi]\!] \cap D = [\![\forall x \in I\ \phi']\!] \cap D$, where D is as above. By the quantifier semantics it suffices to prove that $[\![\phi]\!] \cap D\frac{I}{x} = [\![\phi']\!] \cap D\frac{I}{x}$. This holds, because for all variable assignments $d \in D$, for all $r \in I$, $d\frac{r}{x} \in B'$, and moreover, by the induction hypothesis $[\![\phi']\!] \cap B' = [\![\phi]\!] \cap B'$.

The uniqueness of the fixpoint operator follows from contractance and monotonicity of narrowing operators [16, 3]. ■

By easy induction we also get:

Theorem 5 *$\overline{N}_A$ ensures $\widehat{A}$-consistency.*

By applying Theorem 1 we get a corresponding inner narrowing operator $\underline{N}_A$ from $\overline{N}_A$. Note, however, that $\overline{N}_A$ and $\underline{N}_A$ do not commute, and $\overline{N}_A \circ \underline{N}_A$ is not idempotent.

As for the classical conjunctive case, the complexity of the algorithm in a floating-point implementation is polynomial in the problem dimension (the

number of floating point numbers that one can remove from the quantification bounds is polynomial). So, as desired, pruning is efficient compared to expensive exhaustive search, and even more so compared to the doubly exponential complexity of symbolic solvers [14, 11].

Note however, that the cardinality of the usual float-point representations is so high that, in addition, one should take care that the worst-case complexity is not reached in practice.

6 Solver

Now a branch-and-prune algorithm can do pruning according to Definition 7, and branching means replacing sub-constraints of the form $\forall x \in I\ \phi$ by $\forall x \in I_1\ \phi \wedge \forall x \in I_2\ \phi$, or sub-constraints of the form $\exists x \in I\ \phi$ by $\exists x \in I_1\ \phi \vee \exists x \in I_2\ \phi$, where $I = I_1 \cup I_2$. We assume branching to be *fair*, in the sense that every bound will eventually be split (finding such a strategy is easy, but finding an optimal branching strategy is highly non-trivial).

For disproving a closed constraint ϕ we repeatedly branch and prune the input constraint $(\phi, \mathbf{T})$ until $\mathbf{T}$ is pruned to $\mathbf{F}$ (remember that $\mathbf{F}$ is an abbreviation for the empty box assignment). For proving we do the same on the opposite of ϕ. For computing the truth-value we do both things in parallel.

A solver for open quantified constraints could for example use the following specification:

Given:

- A quantified constraint ϕ with n free variables,
- $B \subseteq \mathbb{R}^n$,
- $\varepsilon \in \mathbb{R}^+$

Find: Sets of boxes Y, N s.t.

- all elements of Y are in the solution set of ϕ,
- all elements of N are not in the solution set of ϕ,
- $\mathrm{Vol}(B \setminus \bigcup Y \setminus \bigcup N) \leq \varepsilon$

This specification allows the user to decide on the trade-off between run-time and precision. When choosing a large ε, only a small part of the solution set of ϕ within B will be characterized by Y and N, when choosing an ε close to zero, almost the whole set will be characterized.

An according solver is an easy extension of the closed case that would record the boxes that narrowing of the input constraint proved to be not in the solution set, and narrowing of the opposite of the input constraint proved to be in the solution set. Furthermore, in addition to branching the quantified variables it also has to branch the free-variable bound. We call the resulting algorithm, a *parallel branch-and-prune solver*.

For discussing termination of such a branch-and-prune solver it is important to see, that the problem of computing truth-values is undecidable. So it is impossible to find an algorithm that terminates always. A solution to this problem is to require termination for all, except very special cases.

For this we observe that the truth-value/solution set of a quantified constraint can be numerically unstable [44]. An example is the quantified constraint $\exists x \in [-1, 1] \ -x^2 \geq 0$ which is true, but becomes false under arbitrarily small positive perturbations of the constant 0. As a consequence, it is not possible to design an algorithm based on approximation that will always terminate (with a correct result). Note that this situation is similar for most computational problems of continuous mathematics (e.g., solving linear equations, solving differential equations). However, as in these cases, most inputs are still numerically stable (in fact, in a certain, realistic model this is the case with probability one [40]). One can even argue that, philosophically speaking, the stable problems are exactly the problems that model real-life problems in a meaningful way.

It is beyond the scope of this paper to present all the formal details for characterizing stable quantified constraints and we will introduce the necessary concepts in a semi-formal way. For this we replace the discrete notion of truth of a quantified constraint by a continuous notion [44]. We interpret universal quantifiers as infimum operators, existential quantifiers as supremum operators, conjunction as minimum, disjunction as maximum, atomic constraints of the form $f > g$ or $f \geq g$ as the function denoted by $f - g$, and atomic constraints of the form $f < g$ or $f \leq$ g as the function denoted by $g - f$. We call the result the *degree of truth* of a quantified constraint and denote it by $[\![\phi]\!]^\circ$ for any constraint ϕ. This function assigns to every variable assignment a real value that is independent of the variables that are not free in ϕ. The idea is that the degree of truth is greater or equal zero for variable assignments that make ϕ true, and less or equal zero for variable assignments that make ϕ false. One can prove [44] that the problem of computing the truth value of a closed quantified constraint is numerically stable (or: *stable*) iff its degree of truth is non-zero.

We assume that the given narrowing operator for atomic constraints eventually succeeds for stable inputs:

Definition 9 *An outer narrowing operator $\overline{A}$ is* converging *iff for all atomic constraints ϕ and sequences $B^0 \supseteq B^1 \supseteq \ldots$ such that*

- *for all $i \in \mathbb{N}$, $B^i \supseteq B^{i+1}$,*
- *and $\bigcap_{i \in \mathbb{N}} B^i = \{d\}$, where the degree of truth of ϕ at the variable assignment d is negative,*

there is a k, such that for all $l \geq k$, the free-variable bound of $\overline{A}(\phi, B^l)$ is empty.

Note that this trivially holds for tight consistency. For hull and box consistency it is necessary that the number of bits used in floating point computation is high enough for a given sequence of boxes. For box consistency, in addition, interval evaluation has to converge for all atomic constraints. This is the case if they only contain continuous functions such as $+$, $\times$, exp, and sin, on which we can implement interval evaluation, see Theorem 2.2 [27]).

However, the above property is in general impossible to fulfill for any narrowing operator based on fixed-precision floating-point arithmetic. Still, one can easily overcome this difficulty, by using sufficient precision [47] (see Theorem 2.1.5 [37]). Moreover, the application of our method to real-life problems has shown that usual double-precision floating-point arithmetic almost always suffices in practice.

Lemma 2 *Let $\overline{A}$ be a converging outer narrowing operator and let the sequence $(\phi^1, B^1), (\phi^2, B^2), \ldots$ be such that*

- *for all $i \in \mathbb{N}$, $B^i \supseteq B^{i+1}$, and*
- *$\bigcap_{i \in \mathbb{N}} B^i = \{d\}$ such that the degree of truth of ϕ at the variable assignment d is negative,*
- *for all $i \in N$, ϕ^{i+1} results from ϕ^i by branching, and*
- *for all $\varepsilon > 0$ there is a k such that for all $l \geq k$, the volume of all quantification sets in ϕ^l is less or equal ε.*[3]

Then there is a k such that for all $l \geq k$, the free-variable bound of $\overline{N}_A(\phi^l, B^l)$ is empty.

Proof. We proceed by induction over the structure of the constraint ϕ^1. For atomic constraints the lemma holds because $\overline{A}$ is converging. Now consider the following cases:

- For constraints of the form $\phi_1 \wedge \phi_2$, $[\![\phi_1 \wedge \phi_2]\!]^\circ(d) = \min\{[\![\phi_1]\!]^\circ(d), [\![\phi_2]\!]^\circ(d)\}$ being negative implies that either $[\![\phi_1]\!]^\circ(d)$ is negative or $[\![\phi_2]\!]^\circ(d)$ is negative. Therefore at least one of the sequences $(\phi_1^1, B^1), (\phi_1^2, B^2), \ldots$ and $(\phi_2^1, B^1), (\phi_2^2, B^2), \ldots$ where ϕ_1^i is the sub-constraint of ϕ^i corresponding to ϕ_1 and ϕ_2^i is the sub-constraint of ϕ^i corresponding to ϕ_2, fulfills the preconditions of the induction hypothesis. Let $r \in \{1, 2\}$ be the number of this sequence. Then there is a k, such that for all $k \geq l$, the free-variable bound of $\overline{N}_A(\phi_r^l, B^l)$ is empty. Thus, by definition of $\overline{N}_A$, the corresponding free-variable bound is also empty in the original sequence.
- For constraints of the form $\phi_1 \vee \phi_2$, $[\![\phi_1 \vee \phi_2]\!]^\circ(d) = \max\{[\![\phi_1]\!]^\circ(d), [\![\phi_2]\!]^\circ(d)\}$ being negative implies that both $[\![\phi_1]\!]^\circ(d)$ and $[\![\phi_2]\!]^\circ(d)$ are negative. Therefore both sequences $(\phi_1^1, B^1), (\phi_1^2, B^2), \ldots$ and $(\phi_2^1, B^1), (\phi_2^2, B^2), \ldots$ where ϕ_1^i is the sub-constraint of ϕ^i corresponding to ϕ_1 and ϕ_2^i is the sub-constraint of ϕ^i corresponding to ϕ_2, fulfill the preconditions of the induction hypothesis. As a consequence there is a k_1, such that for all $l \geq k_1$, the free-variable bound of $\overline{N}_A(\phi_1^l, B^l)$ is empty, and there is a k_2, such that for all $l \geq k_2$, the free-variable bound of $\overline{N}_A(\phi_1^l, B^l)$ is empty. Thus, by definition of $\overline{N}_A$, for all $l \geq \max\{k_1, k_2\}$ the free-variable bound of the l-th element in the original sequence is empty.
- Constraints of the form $\forall x \in I\ \phi'$, are replaced by branching into the form $\forall x \in I_1\ \phi' \ \wedge \ \ldots \ \wedge \ \forall x \in I_k \phi'$. Since the degree of truth of $\forall x \in I\ \phi'$ at d is negative, by definition of infimum, there is a $b \in I$ for which the degree of truth of ϕ' at $d \times b$ is negative. Consider the sequence for which the i-th element consists of the branch of ϕ^i that contains $d \times b$, and of B^i. This sequence fulfills the preconditions of the induction hypothesis, and as a consequence there is a k, such that for all $k \geq l$, the k-th free-variable bound in this sequence is empty. Thus, by definition of $\overline{N}_A$, the corresponding free-variable bound is also empty in the original sequence.

[3] This item formalizes the notion of a fair branching strategy.

- Constraints of the form $\exists x \in I\ \phi'$, are replaced by branching into the form $\exists x \in I_1\ \phi' \vee \ldots \vee \exists x \in I_k \phi'$. Since the degree of truth of $\exists x \in I\ \phi'$ at d is negative, by definition of supremum, for all $b \in I$ the degree of truth of ϕ' at $d \times b$ is negative. This means that each sequence for which the i-th element consists of a branch of ϕ^i and of B^i fulfills the preconditions of the induction hypothesis, and as a consequence there is a k, such that for all $k \geq l$, the k-th free-variable bound in this sequence is empty. Thus, by definition of $\overline{N}_A$, the corresponding free-variable bound is also empty in the original sequence.

  ■

This implies:

Lemma 3 *A branch-and-prune algorithm for disproving a closed constraint succeeds iff the degree of truth of the input is negative.*

From Lemma 3 and its dual version we get:

Theorem 6 *For stable inputs a parallel branch-and-prune solver eventually computes the truth value for closed inputs, and fulfills the above solver specification for open inputs.*

7 Efficient Implementation

In this section we show how to extend the basic solver for allowing an efficient implementation.

7.1 Connectives with Arbitrary Arity

The first step for arriving at an efficient implementation, is to treat conjunctions and disjunctions not as binary operators, but as operators with arbitrary arity (see Definition 7). It is easy to adapt the according algorithms and proofs.

7.2 Quantifier Blocks

Treat quantifiers of the same kind in blocks. That is, quantify over a whole vector of variables at once, using quantifier bounds that are boxes instead of intervals. This allows more flexibility for branching.

7.3 Removing Empty Disjunctive Branches

Pruning might show that one of the branches of a disjunction has an empty solution set. Currently (see Definition 7) this information is forgotten. In order to prevent this we simply remove the corresponding sub-constraint in such as case.

7.4 Combination with Negated Constraint

The parallel branch-and-prune solver developed in Section 6 independently works on proving and disproving the input constraint. For proving, it employs a branch-and-prune procedure on the negation of the input, for disproving, on the input itself. Working on the input and its negation separately has the disadvantage that information computed for one is not used for the other. In order to improve this, we do both on the same constraint, repeatedly negating it in between. The result is Algorithm 1 for closed constraints and Algorithm 2 for open constraints. Here "Branch" and "Prune" do the obvious, except that in the second algorithm "Branch" takes a bounded constraint and returns a set of bounded constraints that either

- contains two elements whose union of bounds is equal to the input bound, or
- contains one element with a quantifier split into a conjunction (in the case of a universal quantifier), or a disjunction (in the case of an existential quantifier).

Algorithm 1 Combined Solver for Closed Constraints

Input: a closed quantified constraint ϕ
Output: the truth-value of ϕ

```
unknown ← T
while unknown do
  neg ← T
  φ' ← F
  (φ, unknown) ← Prune(φ, unknown)
  while unknown and φ ≠ φ' do
    φ' ← φ
    neg ← not neg
    if neg then
      (¬φ, unknown) ← Prune(¬φ, unknown)
    else
      (φ, unknown) ← Prune(φ, unknown)
    end if
  end while
  if unknown then
    φ ← Branch(φ)
  end if
end while
return neg
```

Theorem 7 *Algorithms 1 and 2 are correct and terminate for stable inputs.*

Proof. We just show the proof for Algorithm 1, the other case is similar.

If neg is false at the return statement, then pruning succeeded for ϕ, and so it is false. Otherwise, neg is true and so pruning succeeded for $\neg\phi$ and ϕ is true.

The innermost while loop always terminates because there are only finitely many floating point numbers, and so the narrowing operator can only do finitely

many changes after which $\phi = \phi'$. Termination of the overall loop again is a consequence of Lemma 2. ■

Algorithm 2 Combined Solver for Open Constraints

Input: a quantified constraint ϕ, a box B, and a positive real number ε
Output: Y, a list of boxes on which ϕ is true, and N, a list of boxes on which ϕ is false, such that the volume of $B \setminus \bigcup Y \setminus \bigcup N$ is less than ε.

```
U ← {(φ, B)}
Y ← ∅
N ← ∅
while Vol(⋃U) ≥ ε do
  choose and remove a bounded constraint (φ_U, B_U) from U
  neg ← T
  (φ'_U, B'_U) ← (φ_U, B_U)
  (φ_U, B_U) ← Prune(φ_U, B_U)
  Y ← Y ∪ B'_U \ B_U
  while B_U is non-empty and (φ_U, B_U) ≠ (φ'_U, B'_U) do
    (φ'_U, B'_U) ← (φ_U, B_U)
    neg ← not neg
    if neg then
      (¬φ, B_U) ← Prune(¬φ, B_U)
      Y ← Y ∪ B'_U \ B_U
    else
      (φ, B_U) ← Prune(φ, B_U)
      N ← N ∪ B'_U \ B_U
    end if
  end while
  if Vol(⋃U) ≥ ε then
    U ← U ∪ Branch(φ, B_U)
  end if
end while
```

7.5 Branching Strategy

For arriving at an implementation, a good strategy for choosing a (free-variable or quantification) bound for branching is crucial. We did not yet try to arrive at a theoretically well-founded or even optimal strategy. However, the following approach seems to work well in practice:

In Algorithm 2, for every element of the set U we store the level of the last splitting done (viewing constraints as trees), and for each conjunction or disjunction (of the original constraint, not the ones created by branching) the last branched sub-constraint. We choose the bounded constraint (ϕ_U, B_U) with largest B_U and branch a sub-constraint

- that is one level below the last splitting, or (if this is impossible) on the highest level,
- with the maximum volume of the quantifier bound for conjunctions or disjunctions created by branching, which is

- the next sub-constraint for all other conjunctions and disjunctions.

7.6 Incremental Disjunctive Pruning

The disjunctive case in Definition 7, has the disadvantage that it stores the intermediate results of narrowing all sub-constraints until computing the box union $\uplus$ of all of them. We can avoid this by using an incremental algorithm instead that intermingles the recursive calls with taking the box union of the result.

7.7 Shortcut for Disjunctions

When doing incremental pruning of disjunctions we might detect that the result will be the input box, before inspecting all sub-constraints. We can leave the according loop already at this point.

7.8 Reusing Dual Information

Sometimes an atomic narrowing operator computes the information that narrowing the negation will fail. For example, assume that a bound $[-2, 2]$ of a univariate atomic constraint has been pruned to $[-1, 1]$. For many narrowing operators this implies that the constraint does not hold on -1 and 1. Therefore, we cannot use the opposite constraint to prune $[-1, 1]$ further.

In the case of box consistency, which we use in our implementation, one works on atomic constraints one variable after the other. For each atomic sub-constraint/variable pair we can get the information that pruning or pruning on the opposite will not succeed, that is that this sub-constraint or its opposite is consistent. We store this information by assigning to each atomic sub-constraint two sets of variables (the *mark* and the *opposite mark*). Furthermore, in order to reflect the situation for box consistency we assume that consistency also takes into account variables.

Definition 10 *We call a bounded constraint* (ϕ, B) correctly marked *iff for every atomic sub-constraint* ϕ' *of* ϕ*, every box assignment* B' *that results from* B *by replacing all variables of* B *bounded by* ϕ *by these bounds, and every variable* v *in the mark of* ϕ'*,* (ϕ', B', v) *is consistent.*

Sometimes we change the bound of a variable. In this case some of the marks might not stay valid. For example, take a constraint of the form $\exists x \in I_x \exists y \in I_y\ [\phi_1 \wedge \phi_2]$, where ϕ_1 is an atomic constraint that contains both variables x and y, but ϕ_2 is an atomic constraint that only contains y. After pruning, the marks and opposite marks of both ϕ_1 and ϕ_2 can be set to $\{x, y\}$, indicating that no further pruning is possible. After branching the quantifier of y we have to remove the marks of both copies of ϕ_1, but we can keep the marks of the copies of ϕ_2 since ϕ_2 contains no variable affected by the branching. Therefore, further pruning will know that calls to the atomic narrowing operator of ϕ_2 are not necessary.

So denote by $\text{Notify}(\phi, V)$ the result of replacing all marks of sub-constraints that contain variables in V by the empty set.

Lemma 4 *For every bounded constraint (ϕ, B) that is correctly, and for every box B' such that for all $v \notin V$, $B'(v) = B(v)$, Notify$(\phi, V), B'$ is correctly marked.*

Proof. All sub-constraints of Notify(ϕ, V) that contain V have empty marks and Definition 10 requires nothing from them. For all bounded sub-constraints that do not contain V, the corresponding bound inherited from B is the same as the one from B', and therefore one does not have to change the corresponding marks. ■

Definition 11 *A narrowing operator* preserves marks *iff after applying it to a bounded constraint that is correctly marked the result is again correctly marked.*

We assume that we have an atomic narrowing operator that preserves marks and opposite marks. Of course this can be easily done by always setting the marks to the empty set. But, in our implementation, we will try to set them as large as possible.

Now, whenever applying the narrowing operator on atomic constraint/variable pairs we check the marks before. For more complicated constraints we have to adapt the narrowing operator of Definition 7 such that it updates the marks accordingly. This means that for disjunctions, if the box union is different from the boxes resulting from narrowing an individual constraints, then we have to do notification on it. In a similar way, for conjunctions, when computing the fixpoint, every time narrowing succeeds for a sub-constraint, we have to do notification for all other sub-constraints. Clearly, by Lemma 4 the adapted narrowing operator preserves marks and opposite marks.

Also for branching we have to do according notification for the changed variables. When using adapted branching and pruning in Algorithms 1 and 2, we set all marks to the empty set at the beginning and preserve marks and opposite marks throughout.

Clearly the resulting solver does less calls to the narrowing operator for projection constraints than the original one. Furthermore this also gives an improvement for the case of unquantified constraints for which the solution set does not consist of finitely many, isolated solutions.

8 Timings

We have implemented the algorithm described in this paper using as the atomic narrowing operator an algorithm [25] that computes something similar to box consistency. In Table 1 we compare this implementation described in this paper with an implementation of the older algorithm using "cylindrical box decomposition" [41] which also used the same atomic narrowing operator. Here the heading "0.1 old" refers to running the older algorithm until it leaves not more than a fraction of 1/10 of the solution space unknown, the heading "first" refers to running the current algorithm till it finds the first true box, and the heading "0.1" refers to running the current algorithm in the same way as the older one.

Columns headed by "Time" list the time in seconds needed to solve the problem where ε means less than a second and ∞ more than 10 minutes; columns headed by "Hits" list the number of calls to the atomic narrowing operator

for the new algorithms (this is a good efficiency measure because it ignores implementation details, and because atomic narrowing takes the largest part of the overall runtime); and columns headed by "Boxes" list the total number of boxes created (true, false, and unknown boxes).

Examples starting with McCallum are from computational geometry [35]—asking the questions whether there exists a solution to a given system of inequalities. The example "circle" simply computes a description of the unit disc $x^2 + y^2 \leq 1$, "silaghi" is a system of inequalities describing the geometry of a simple mechanical problem [52], the example "anderson2" is a system of polynomial inequalities from control engineering [2, 1, 23], the example "anderson2_proj" computes the projection of the former into two-dimensional space, the example "termination" proves the termination of a certain term-rewrite system [15]. All examples starting with "robust" are taken from robust control [19, 22, 33, 20, 28, 26] and laid out on a web-site [39].

For all examples we push quantifiers inside as much as possible by transforming $\forall(\phi_1 \wedge \phi_2)$ to $(\forall\phi_1) \wedge (\forall\phi_2)$ and the dual for existential quantification. Furthermore, in the few cases, where no quantification bounds were available, we introduced new, very large ones.

	0.1 old		first			0.1		
Example	Time	Boxes	Time	Hits	Boxes	Time	Hits	Boxes
McCallum_2_1	ε		ε	22		ε	22	
McCallum_2_2	ε		ε	157		ε	157	
McCallum_2_3	ε		ε	2854		ε	2854	
McCallum_2_4	ε		ε	153		ε	153	
anderson2	ε	391	10.95	150343	575	ε	3372	250
anderson2_proj	16.81	3466	ε	1129	21	2.69	21058	620
circle	ε	42	ε	19	10	ε	60	26
robust-1	ε	23	ε	13	3	ε	121	16
robust-2	∞		90.74	663011	1266	∞	∞	
robust-3	∞		ε	344	5	ε	936	9
robust-5	∞		ε	376	11	∞	∞	
robust-6	5.4	112	ε	34	3	256.14	564326	4309
silaghi1	ε	19	ε	25	6	ε	25	6
termination	∞		ε	90		ε	90	

Table 1: Comparison with Cylindrical Box Decomposition

As one can see, except for the example "robust-6" the number of generated boxes is much smaller for the new algorithm. For the examples where it is possible to make a clear comparison of the run-times, the new algorithm is also faster (again with the exception of "robust-6"). An analysis of the behavior for the outlier "robust-6" shows that in this case our branching heuristics do not work very well—alternative heuristics show a much better behavior. This suggests that a detailed study of such heuristics—expanding results for a simpler branch-and-bound approach [45] can still result in large improvements of the method.

In Table 2 we show the results of some of the algorithm improvements introduced in Section 7. We chose these that need a non-trivial implementation effort or these for which it is not totally clear that they improve the efficiency

of the algorithm. These are the ones described in Section 7.7 and Section 7.8.

	no reuse/no sh.cut		no reuse/sh.cut		reuse/no sh.cut		reuse/sh.cut	
Example	Time	Hits	Time	Hits	Time	Hits	Time	Hits
McCallum_2_1	ε	22	ε	22	ε	22	ε	22
McCallum_2_2	ε	182	ε	169	ε	150	ε	157
McCallum_2_3	ε	10059	ε	3005	ε	3760	ε	2854
McCallum_2_4	ε	163	ε	165	ε	138	ε	153
anderson2	ε	4034	ε	4251	ε	2964	ε	3372
anderson2_proj	8.49	78985	8.54	65401	7.57	55443	7.54	55244
circle	ε	95	ε	95	ε	60	ε	60
robust-1	ε	138	ε	139	ε	116	ε	121
robust-2	∞	∞	∞	∞	∞	∞	∞	∞
robust-3	ε	1623	ε	1115	ε	1003	ε	936
robust-5	∞	∞	∞	∞	∞	∞	∞	∞
robust-6	147.89	637216	285.52	702734	69.8	255163	256.14	564326
silaghi1	ε	31	ε	27	ε	27	ε	25
termination	ε	282	ε	102	ε	156	ε	90

Table 2: Comparison of Improvements

One can conclude that reusing dual information (Section 7.8) always improves the algorithm, sometimes significantly. This phenomenon also occurs for examples that do not contain any quantifiers. On the other hand, for taking shortcuts for disjunctions (Section 7.7), the influence on efficiency is inconclusive, especially when combined with the former improvement.

Note that often one can get even better run-times by symbolically eliminating linearly quantified variables before [59, 31]. Unfortunately, in some cases the result can be very large, destroying the positive effect of the eliminated variable. Future work will investigate this behavior in detail.

9 Relation to Classical Algorithms

A. Tarski [53] showed that quantified constraints over the reals with equality and inequality predicates, multiplication and addition admit quantifier elimination. Adding additional function symbols (e.g., sin, tan), usually removes this property [48, 54, 32]. Using the method in this paper one can still compute useful information for these cases, provided that the input is numerically stable.

The complexity bound supplied by Tarski's method has been improved several times [14, 46, 4]—but the problem is inherently doubly exponential [18, 59] in the number of quantifier alternations, and exponential in the number of variables.

The only general algorithm for which a practically useful implementation exists, is the method of *quantifier elimination by cylindrical algebraic decomposition* [14]. This algorithm employs similar branching as the algorithm presented in this paper. However, its branching operation is much more complicated because it branches into a finite set of *truth-invariant cells*, that is, into pieces whose value can be computed by evaluation on a single *sample point*. For being able to do this, its quantifier bounds can depend on the free variables,

and branching is done based on information from *projection polynomials*. For implementing these operations one needs expensive real algebraic number computations.

Instead of branching, quantifier elimination by *partial cylindrical algebraic decomposition* [15] employs pruning in a similar sense as described in this paper. However it still decomposes into truth-invariant cells, which again needs expensive computation of projection polynomials, and real algebraic numbers.

In contrast to this, the narrowing operator provided in this paper is cheap, and can do pruning in polynomial time. As a result, we have a clear separation between polynomial time pruning, and exponential branching. So we have a way of working around the high worst-case complexity of the problem, whenever a small amount of branching is necessary.

For inputs with free variables all of these algorithms produce symbolic output that is equivalent to the input, but quantifier-free. This output can be huge. In many applications, such output is only considered a transformation of the problem, but not a solution. In contrast to this, our algorithm produces explicit numerical output, that one can directly visualize for dimensions less than three.

10 Conclusion

In this paper we have provided an algorithm for solving quantified inequality constraints over the reals. Although this is an undecidable problem, the algorithm terminates for all, except pathological (i.e., unstable) inputs.

The result has several advantages over earlier approaches: Compared to symbolic approaches [14, 11] it is not restricted to polynomials, and avoids complicated and inefficient computation with real algebraic numbers. Furthermore it decreases the necessity for expensive space decomposition by extracting information using fast consistency techniques. Compared to earlier interval approaches that could deal with quantifiers of some form, it provably terminates for all except unstable inputs, and can either handle a more general case [28, 33, 6, 50], or provides a much cleaner, more elegant, and efficient framework [41].

As a side-effect, this paper even improves the current methods for (unquantified) numerical constraint satisfaction problems in the case where the solution set does not consist of finitely many, isolated solutions.

In future work we will explore optimal branching strategies [45, 30, 17], exploit continuity information for efficiently dealing with equalities [43], exploit the structure of quantified constraints in special problem domains, and provide an implementation that allows the flexible exchange of different atomic narrowing operators.

This work has been supported by a Marie Curie fellowship of the European Union under contract number HPMF-CT-2001-01255.

References

[1] C. Abdallah, P. Dorato, W. Yang, R. Liska, and S. Steinberg. Applications of quantifier elimination theory to control system design. In *4th IEEE Mediterranean Symposium on Control and Automation*, Crete, Greece, 1996.

[2] B. D. O. Anderson, N. K. Bose, and E. I. Jury. Output feedback stabilization and related problems — solution via decision methods. *IEEE Trans. Automatic Control*, AC-20:53–66, 1975.

[3] K. R. Apt. The essence of constraint propagation. *Theoretical Computer Science*, 221(1–2), 1999.

[4] S. Basu, R. Pollack, and M.-F. Roy. On the combinatorial and algebraic complexity of quantifier elimination. In S. Goldwasser, editor, *Proceedings fo the 35th Annual Symposium on Foundations of Computer Science*, pages 632–641, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[5] F. Benhamou. Interval constraint logic programming. In Podelski [38].

[6] F. Benhamou and F. Goualard. Universally quantified interval constraints. In *Proc. of the Sixth Intl. Conf. on Principles and Practice of Constraint Programming (CP'2000)*, number 1894 in LNCS, Singapore, 2000. Springer Verlag.

[7] F. Benhamou, F. Goualard, L. Granvilliers, and J. F. Puget. Revising hull and box consistency. In *Int. Conf. on Logic Programming*, pages 230–244. MIT Press, 1999.

[8] F. Benhamou, D. McAllester, and P. Van Hentenryck. CLP(Intervals) revisited. In *International Symposium on Logic Programming*, pages 124–138, Ithaca, NY, USA, 1994. MIT Press.

[9] F. Benhamou and W. J. Older. Applying interval arithmetic to real, integer and Boolean constraints. *Journal of Logic Programming*, 32(1):1–24, 1997.

[10] L. Bordeaux and E. Monfroy. Beyond NP: Arc-consistency for quantified constraints. In P. Van Hentenryck, editor, *Proc. of Principles and Practice of Constraint Programming (CP 2002)*, number 2470 in LNCS. Springer, 2002.

[11] B. F. Caviness and J. R. Johnson, editors. *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Springer, Wien, 1998.

[12] J. G. Cleary. Logical arithmetic. *Future Computing Systems*, 2(2):125–149, 1987.

[13] H. Collavizza, F. Delobel, and M. Rueher. Extending consistent domains of numeric CSP. In *IJCAI-99*, Stockholm, 1999.

[14] G. E. Collins. Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition. In Caviness and Johnson [11], pages 134–183.

[15] G. E. Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12:299–328, 1991. Also in [11].

[16] P. Cousot and R. Cousot. Automatic synthesis of optimal invariant assertions: Mathematical foundations. In *Proceedings of the 1977 symposium on Artificial intelligence and programming languages*, pages 1–12, 1977.

[17] T. Csendes and D. Ratz. Subdivision direction selection in interval methods for global optimization. *SIAM Journal on Numerical Analysis*, 34(3):922–938, 1997.

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

[19] P. Dorato. Quantified multivariate polynomial inequalities. *IEEE Control Systems Magazine*, 20(5):48–58, October 2000.

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

[21] H. Fargier, J. Lang, and T. Schiex. Mixed constraint satisfaction: A framework for decision problems under incomplete knowledge. In *Proc. of AAAI'96*, 1996.

[22] 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*, pages 3029–3031, 1993.

[23] 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 Electr. Eng. (IEE), 1999.

[24] L. Granvilliers. A symbolic-numerical branch and prune algorithm for solving non-linear polynomial systems. *Journal of Universal Computer Science*, 4(2):125–146, 1998.

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

[26] L. Jaulin, I. Braems, and É. Walter. Interval methods for nonlinear identification and robust control. In *41st IEEE Conference on Decision and Control*, 2002.

[27] L. Jaulin, M. Kieffer, O. Didrit, and É. Walter. *Applied Interval Analysis, with Examples in Parameter and State Estimation, Robust Control and Robotics*. Springer, Berlin, 2001.

[28] L. Jaulin and É. Walter. Guaranteed tuning, with application to robust control and motion planning. *Automatica*, 32(8):1217–1221, 1996.

[29] J. Jourdan and T. Sola. The versatility of handling disjunctions as constraints. In M. Bruynooghe and J. Penjam, editors, *Proc. of the 5th Intl. Symp. on Programming Language Implementation and Logic Programming, PLILP'93*, number 714 in LNCS, pages 60–74. Springer Verlag, 1993.

[30] V. Kreinovich and T. Csendes. Theoretical justification of a heuristic subbox selection criterion for interval global optimization. *Central European Journal of Operations Research*, 9:255–265, 2001.

[31] R. Loos and V. Weispfenning. Applying linear quantifier elimination. *The Computer Journal*, 36(5):450–462, 1993.

[32] A. Macintyre and A. Wilkie. On the decidability of the real exponential field. In P. Odifreddi, editor, *Kreiseliana—About and Around Georg Kreisel*, pages 441–467. A K Peters, 1996.

[33] S. Malan, M. Milanese, and M. Taragna. Robust analysis and design of control systems using interval arithmetic. *Automatica*, 33(7):1363–1372, 1997.

[34] K. Marriott, P. Moulder, P. J. Stuckey, and A. Borning. Solving disjunctive constraints for interactive graphical applications. In *7th Intl. Conf. on Principles and Practice of Constraint Programming - CP2001*, number 2239 in LNCS, pages 361–376. Springer, 2001.

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

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

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

[38] A. Podelski, editor. *Constraint Programming: Basics and Trends*, volume 910 of *LNCS*. Springer Verlag, 1995.

[39] S. Ratschan. Applications of quantified constraint solving over the reals— bibliography. `http://www.mpi-sb.mpg.de/~ratschan/appqcs.html`, 2001.

[40] S. Ratschan. Real first-order constraints are stable with probability one. `http://www.mpi-sb.mpg.de/~ratschan/preprints.html`, 2001. Draft.

[41] S. Ratschan. Approximate quantified constraint solving by cylindrical box decomposition. *Reliable Computing*, 8(1):21–42, 2002.

[42] S. Ratschan. Continuous first-order constraint satisfaction. In J. Calmet, B. Benhamou, O. Caprotti, L. Henocque, and V. Sorge, editors, *Artificial Intelligence, Automated Reasoning, and Symbolic Computation*, number 2385 in LNCS, pages 181–195. Springer, 2002.

[43] S. Ratschan. Continuous first-order constraint satisfaction with equality and disequality constraints. In P. van Hentenryck, editor, *Proc. 8th International Conference on Principles and Practice of Constraint Programming*, number 2470 in LNCS, pages 680–685. Springer, 2002.

[44] S. Ratschan. Quantified constraints under perturbations. *Journal of Symbolic Computation*, 33(4):493–505, 2002.

[45] S. Ratschan. Search heuristics for box decomposition methods. *Journal of Global Optimization*, 24(1):51–60, 2002.

[46] J. Renegar. On the computational complexity and geometry of the first-order theory of the reals. *Journal of Symbolic Computation*, 13(3):255–352, March 1992.

[47] N. Revol and F. Rouillier. Motivations for an arbitrary precision interval arithmetic and the MPFI library. *Reliable Computing*, 2005. to appear.

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

[49] D. Sam-Haroud and B. Faltings. Consistency techniques for continuous constraints. *Constraints*, 1(1/2):85–118, September 1996.

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

[51] S. P. Shary. A new technique in systems analysis under interval uncertainty and ambiguity. *Reliable Computing*, 8:321–418, 2002.

[52] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Search techniques for non-linear constraints with inequalities. In *Proceedings of the 14th Canadian Conference on AI*, 2001.

[53] A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. Univ. of California Press, Berkeley, 1951. Also in [11].

[54] L. van den Dries. Alfred Tarski's elimination theory for real closed fields. *Journal of Symbolic Logic*, 53(1):7–19, 1988.

[55] P. Van Hentenryck, D. McAllester, and D. Kapur. Solving polynomial systems using a branch and prune approach. *SIAM Journal on Numerical Analysis*, 34(2):797–827, 1997.

[56] P. Van Hentenryck, L. Michel, and Y. Deville. *Numerica: A Modeling Language for Global Optimization*. The MIT Press, 1997.

[57] P. Van Hentenryck, V. Saraswat, and Y. Deville. The design, implementation, and evaluation of the constraint language cc(FD). In Podelski [38].

[58] T. Walsh. Stochastic constraint programming. In *Proc. of ECAI*, 2002.

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