Journal of Universal Computer Science, vol. 6, no. 9 (2000), 861-880
submitted: 22/6/00, accepted: 5/9/00, appeared: 28/9/00 © Springer Pub. Co.

Uncertainty Propagation in Heterogeneous Algebras for Approximate Quantified Constraint Solving

Stefan Ratschan
(Research Institute for Symbolic Computation, Austria
Stefan.Ratschan@risc.uni-linz.ac.at)

Abstract: When trying to solve quantified constraints (i.e., first-order formulas over the real numbers) exactly, one faces the following problems: First, constants coming from measurements are often only approximately given. Second, solving such constraints is in general undecidable and for special cases highly complex. Third, exact solutions are often extremely complicated symbolic expressions. In this paper we study how to do approximate computation instead — working on approximate inputs and producing approximate output. For this we show how quantified constraints can be viewed as expressions in heterogeneous algebra and study how to do uncertainty propagation there. Since set theory is a very fundamental approach for representing uncertainty, also here we represent uncertainty by sets. Our considerations result in a general framework for approximate computation that can be applied in various different domains.

Key Words: Constraints, uncertainty, interval computation, computational logic
Category: I.2.4, I.2.3, F.4.1

1 Introduction

Let a quantified constraint be a first-order formula over the real numbers. This means that 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. When trying to solve such constraints exactly, one hits upon the following problems: First, the constants occurring in constraints often come from measurements, and are therefore only approximately given. Second, the problem is in general undecidable (reducible to solving Diophantine equations) and for special cases (e.g., the decidable sub-theory of real-closed fields [dW93, Col75, Mis93, Hon92]) highly complex [DH88, Wei88, Ren92]. Third, exact solutions are often extremely complicated symbolic expressions that need further numeric processing to be useful.

In order to avoid these problems, in this paper we show how uncertain information can be propagated from approximate input, through approximate computation, to approximate output. This allows us to deal with approximately given input constants, to achieve efficiency by computing approximate results only, and to choose an approximation of the output that has a simple representation.

Set theory is one of the oldest methods for dealing with uncertainty: Given a set A, the formulation "for any a in A" can be found in almost every scientific book So we use set theory here, also. The basic idea is, that we can approximate an exact object a by a set $\mathbf{a}$ containing a. The smaller the set $\mathbf{a}$ is, the smaller the uncertainty about a is. When we want to compute the result of applying some mapping F on a, we apply F to the set $\mathbf{a}$ instead, to get the set of all possible results.

For using this scheme in approximate quantified constraint solving we associate with each constraint a function F such that for a certain a, $F(a)$ is the exact solution set of the constraint. Then we approximate a by a set $\mathbf{a}$ to do approximate computation.

We find this function F by showing that quantified constraints correspond to expressions in a heterogeneous algebra in which all the symbols occuring in constraints, except variables, act as function symbols. So we reduce approximate quantified constraint solving to uncertainty propagation in this algebra. Since heterogeneous algebras are very common in computer science (in the form of abstract data types), we develop a general method for propagating uncertainty represented by sets in heterogeneous algebras. To this end we introduce the notion of *heterogeneous power algebra* and prove various properties showing the usefulness of such power algebras for approximate computation in general. Furthermore we identify the *dependency problem* as a crucial object of study for implementing efficient approximation algorithms.

The contribution of this paper is a novel combination of ideas from various fields (uncertain reasoning in artificial intelligence, logic, universal algebra, interval mathematics, many-valued logic and others) and their application to quantified constraint solving. Thus our work is inherently interdisciplinary and gives a clarification of the interplay of these areas [Hoa96, HJ98]. See Section 8 for details on related work.

The structure of the paper is as follows: In Section 2, we discuss how quantified constraint solving gives rise to uncertainty propagation. In Section 3, we show how quantified constraints can be viewed as expressions in a heterogeneous algebra. In Section 4, we show how to propagate uncertainty over single mappings. In Section 5, we extend this to uncertainty propagation over expressions in heterogeneous algebra. In Section 6, we study intervals as a special representation of the occurring sets. In Section 7, we apply the results to quantified constraint solving. In Section 8, we discuss related work, and in Section 9, we draw final conclusions. Throughout the paper, boldface denotes sets that approximate single objects and $\mathbb{B}$ denotes the set containing the Boolean values T and F. Furthermore we allow multi-ary function composition such that $(f \circ (g_1, \ldots, g_n))(a) = f(g_1(a), \ldots, g_1(a))$.

2 Approximate Quantified Constraint Solving

In this section we discuss informally how quantified constraint solving gives rise to uncertainty propagation, and we fill out the formal details in the following sections. 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. We want to solve such constraints, where "solving" means to find the truth value of a closed formula and to find a simple representation of the solution set of an open formula (we will not formalize the notion of "simple" here — informally it means that such a representation should make it is easy to see whether a given element is in the solution set or not).

Here we want to propagate uncertainty for various reasons: First, the constants very often come from measurements, and are thus not exact. Second, the exact problem is in general not decidable, or — for special sub-theories — only decidable with extremely high complexity [Tar51, Col75, CH91, DH88, Wei88, Ren92]. Third, the outputs of known exact methods are often extremely complicated symbolic expressions, that are not always helpful in practice.

Here the first problem produces input uncertainty. For solving the second and third problem, we introduce additional uncertainty by replacing the objects in constraints that introduce complexity by sets that contain them. The bigger these sets are allowed to be, the more choices we have for representing them in a simple way, and the easier we can do efficient computation that produces simple output.

The objects that introduce complexity in constraints are represented by the variables. Consider the example $\forall x\ [x^2 \geq 0]$. One could view the variable x as a place-holder for real values. However, since x occurs quantified, the truth value of the whole constraint depends on infinitely many different values for x. So we view the n variables in a constraint as functions $d_1^{id}, \ldots, d_n^{id}$ such that $d_i^{id}(p_1, \ldots, p_n) = p_i$ (the *identity term functions*), where p_i represents the value assigned to the i-th variable.

We approximate these objects by sets that contain them. In the following sections we will show show to propagate the resulting uncertainty according to the following plan:

1. Assign to each constraint ϕ in n variables a mapping $[\![\phi]\!]$ that takes n functions in $\mathbb{R}^n \rightarrow \mathbb{R}$, such that $[\![\phi]\!](d_1^{id}, \ldots, d_n^{id})$ is the same as the logical meaning of ϕ.
2. Extend this mapping to sets of functions.

Then we can apply this mapping to sets that contain the identity term functions, to get approximate results. This allows incremental approximate computation, using the following scheme:

- Let $\mathbf{d_1^{id}}, \ldots, \mathbf{d_n^{id}}$ be sets containing the i-th identity term function, respectively.
- While the uncertainty of $[\![\phi]\!](\mathbf{d_1^{id}}, \ldots, \mathbf{d_n^{id}})$ is too high, remove elements from $\mathbf{d_1^{id}}, \ldots, \mathbf{d_n^{id}}$ (except the identity term functions).

Since this scheme starts with an approximation of high uncertainty that it improves incrementally, it is *tunable* in the sense, that a user can decide on the trade-off between efficiency and precision, either by fixing one of the two beforehand, or by interrupting during computation [BCK97].

3 Quantified Constraints As Expressions in a Heterogeneous Algebra

In this section we define the mapping $[\![\phi]\!]$ by showing how quantified constraints can be viewed as expressions in a heterogeneous algebra. For this we fix an ordered set of variables $\mathbf{V}$. The order on $\mathbf{V}$ allows us to speak of the i-th variable in a subset of $\mathbf{V}$. Now recall [SB81, EM85]:

Definition 1. A *heterogeneous algebra* consists of countably many sets $(\mathbb{A}_i)_{i\in\mathbb{N}}$ (the *sorts*), function symbols with signatures of the form $\mathbb{A}_{i_1} \times \cdots \times \mathbb{A}_{i_n} \rightarrow \mathbb{A}_j$, and an interpretation $I_{\mathbb{A}}$ that assigns according mappings to these symbols.

Since we only deal with heterogeneous algebras in this paper we now drop the word "heterogeneous" and just speak of algebras. Given an algebra $\mathbb{A}$, one can build expressions from function symbols and variables in the usual way such that the signatures fit. We say that an expression is *free* iff every variable occurs at most once. We denote the meaning of an expression e by $[\![e]\!]_{\mathbb{A},V}$. This is a mapping $\mathbb{A}_{i_1} \times \cdots \times \mathbb{A}_{i_{|V|}} \to \mathbb{A}_j$, defined as follows:

$$[\![f(e_1, \ldots, e_k)]\!]_{\mathbb{A},V} := I_{\mathbb{A}}(f) \circ ([\![e_1]\!]_{\mathbb{A},V}, \ldots, [\![e_k]\!]_{\mathbb{A},V}),$$

and

$$[\![v]\!]_{\mathbb{A},V}(a_1, \ldots, a_{|V|}) := a_i,$$

where v is the i-th variable in V. By using this variable-set argument instead of explicitly writing down a variable assignment, as commonly used in logic [EFT84] and universal algebra, we can use function application in the usual form. In the following we sometimes use the abbreviation $[\![e]\!]_{\mathbb{A}}$ for $[\![e]\!]_{\mathbb{A},V}$, where V is the set of variables in e.

Now we show how quantified constraints can be viewed as expressions in such an algebra such that the meaning associated to a constraint by this algebra is equal to its logical meaning. We have the following sorts:

– The n-dimensional solution sets, which we model by functions $\mathbb{R}^n \to \mathbb{B}$ that return T if an element is in the solution set and F if it is not in the solution set.
– The functions denoted by terms (*n-dimensional term functions*), which we model by functions $\mathbb{R}^n \to \mathbb{R}$.

Let $\mathbb{D}_n^{\mathbb{B}}$ denote the n-dimensional solution sets, and let $\mathbb{D}_n^{\mathbb{R}}$ denote the n-dimensional term functions. Now we view all the symbols occurring in constraints, except variables, as function symbols, and define an interpretation $I_{\mathbb{D}}$ that assigns mappings to these function symbols.

Let us first look at examples: The mapping corresponding to conjunction takes two solution sets and produces a solution set. A representation of its application to certain inputs can be seen in figure 1. In a similar way, the mapping corresponding to addition takes two term functions and produces a term function. A representation of its application to certain inputs can be seen in figure 2. As another example, the mapping corresponding to existential quantification takes a solution set and produces a solution set. A representation of its application to a certain input can be seen in figure 3. Here one can already see that, for simplicity reasons, we model quantification in such a way that it does not change the dimension of solution set, but only makes it independent of the quantified variable.

Let us formalize this. We assign the following signatures to the symbols:

Constants: $\mathbb{D}_n^{\mathbb{R}}$
Function symbols (e.g., $+$, $\times$, sin): $\mathbb{D}_n^{\mathbb{R}} \times \cdots \times \mathbb{D}_n^{\mathbb{R}} \to \mathbb{D}_n^{\mathbb{R}}$
Predicates (e.g., $=$, $<$, $\leq$): $\mathbb{D}_n^{\mathbb{R}} \times \ldots \mathbb{D}_n^{\mathbb{R}} \to \mathbb{D}_n^{\mathbb{B}}$
Connective $\neg$: $\mathbb{D}_n^{\mathbb{B}} \to \mathbb{D}_n^{\mathbb{B}}$

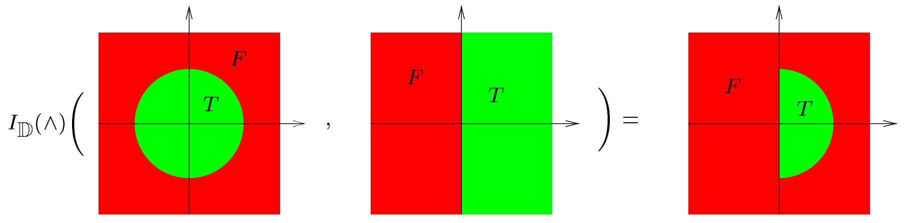

Figure 1: Mapping of Conjunction

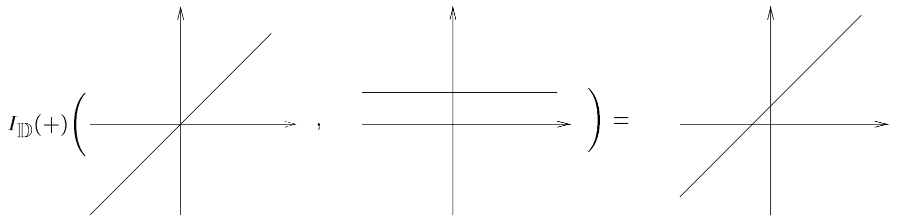

Figure 2: Mapping of Addition

Connectives $\vee$ and $\wedge$: $\mathbb{D}_n^{\mathbb{B}} \times \mathbb{D}_n^{\mathbb{B}} \to \mathbb{D}_n^{\mathbb{B}}$
Quantifiers $\forall$ and $\exists$: $\mathbb{D}_n^{\mathbb{B}} \to \mathbb{D}_n^{\mathbb{B}}$

Again notice that we defined all these signatures on n-dimensional solution sets and term functions, although some of the according variables might occur bound in a constraint. In this case the resulting solution sets and term functions will be independent of these variables. In the extreme case of a constraint with no free variables, the result will be either a function that is true everywhere, or false everywhere.

Let us denote by $I_{\mathbb{R}}$ the usual assignment of functions to function symbols, and predicates to predicate symbols, and by $I_{\mathbb{B}}$ the usual assignment of Boolean functions to logical connectives. Now we can define the function $I_{\mathbb{D}}$ that assigns the desired mappings to individual symbols.

- For constants c, $p \in \mathbb{R}^n$, $I_{\mathbb{D}}(c)(p) = c$
- For k-ary function and predicate symbols f (e.g., $+$, sin, $\leq$), $d_1, \ldots, d_k \in \mathbb{D}_n^{\mathbb{R}}$, $I_{\mathbb{D}}(f)(d_1, \ldots, d_k) = I_{\mathbb{R}}(f) \circ (d_1, \ldots, d_k)$
- For k-ary connectives f (e.g., $\neg$, $\wedge$), $d_1, \ldots, d_k \in \mathbb{D}_n^{\mathbb{B}}$, $I_{\mathbb{D}}(f)(d_1, \ldots, d_k) = I_{\mathbb{B}}(f) \circ (d_1, \ldots, d_k)$
- For the quantifier $\forall x_i$, $d \in \mathbb{D}_n^{\mathbb{B}}$, and $(p_1, \ldots, p_n) \in \mathbb{R}^n$,
  $I_{\mathbb{D}}(\forall x_i)(d)(p_1, \ldots, p_i, \ldots, p_n) =$
  T if for all $q \in \mathbb{R}$, $d(p_1, \ldots, p_{i-1}, q, p_{i+1}, \ldots, p_n) = T$,
  F otherwise.

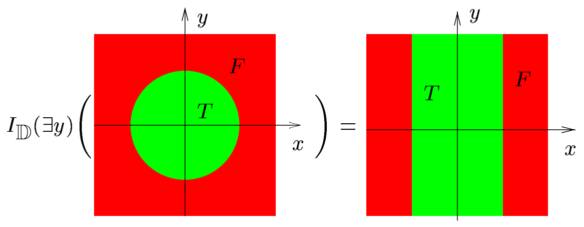

Figure 3: Mapping of Existential Quantification

– For the quantifier $\exists x_i$, $d \in \mathbb{D}_n^{\mathbb{B}}$, and $(p_1, \ldots, p_n) \in \mathbb{R}^n$,
  $I_{\mathbb{D}}(\exists x_i)(d)(p_1, \ldots, p_i, \ldots, p_n) =$
  T if there is a $q \in \mathbb{R}$ s.t. $d(p_1, \ldots, p_{i-1}, q, p_{i+1}, \ldots, p_n) = T$,
  F otherwise.

This finishes the definition of the algebra $\mathbb{D}$. It is easy to prove that, for any constraint ϕ in n variables, $[\![\phi]\!]_{\mathbb{D}}(d_1^{id}, \ldots, d_n^{id})$ is the same solution set (truth value) as provided by the usual first-order semantics of constraints (see appendix).

4 Uncertainty Propagation Over Mappings

In the following we show how mappings can be extended to sets and study some properties of the result that are needed to study correctness, termination and efficiency of incremental approximate computation. All the notions introduced in this section are straightforward generalizations of corresponding notions in interval mathematics [Kea96, Moo66, Neu90, Sta96]. In the following we use the convention that the operations $\in$, $\subseteq$ and $\supseteq$ can be used on tuples of sets in the obvious componentwise way, and similarly on mappings on sets, for example:

Definition 2. For mappings $\mathbf{f} : \wp(\mathbb{A}_1) \cdots \times \wp(\mathbb{A}_n) \to \wp(\mathbb{A}')$ and $\mathbf{g} : \wp(\mathbb{A}_1) \times \cdots \times \wp(\mathbb{A}_n) \to \wp(\mathbb{A}')$, $\mathbf{f} \subseteq \mathbf{g}$ iff for all $\mathbf{a} \in \wp(\mathbb{A}_1) \times \cdots \times \wp(\mathbb{A}_n)$, $\mathbf{f}(\mathbf{a}) \subseteq \mathbf{g}(\mathbf{a})$. In this case we say that $\mathbf{f}$ is *tighter* than $\mathbf{g}$.

Now we assume that the arguments to a mapping are not exactly known, but only known to be elements of a certain set. So, corresponding to the mapping, we need an according mapping on sets that propagates this information correctly.

Definition 3. Given mappings $f : \mathbb{A}_1 \times \cdots \times \mathbb{A}_n \to \mathbb{A}'$ and $\mathbf{g} : \wp(\mathbb{A}_1) \times \cdots \times \wp(\mathbb{A}_n) \to \wp(\mathbb{A}')$, $\mathbf{g}$ is an *enclosure* of f iff for all $\mathbf{a} \in \wp(\mathbb{A}_1) \times \cdots \times \wp(\mathbb{A}_n)$, for all $b \in \mathbf{a}$, $f(b) \in \mathbf{g}(\mathbf{a})$.

In other words, enclosures overestimate the range of a mapping on the argument sets (i.e., $\mathbf{g}(\mathbf{a}) \supseteq \{f(b) | b \in \mathbf{a}\}$).

Sometimes we require mappings to propagate precise data (i.e., sets with just one element) without loss of information:

Definition 4. Given mappings $f : \mathbb{A}_1 \times \cdots \times \mathbb{A}_n \to \mathbb{A}'$ and $\mathbf{g} : \wp(\mathbb{A}_1) \times \cdots \times \wp(\mathbb{A}_n) \to \wp(\mathbb{A}')$, $\mathbf{g}$ is *point-preserving of* f iff for all $a \in \mathbb{A}_1 \times \cdots \times \mathbb{A}_n$, $\mathbf{g}(\{a\}) = \{f(a)\}$.

Usually we want that more certainty in the input to a mapping results in more certainty in the output:

Definition 5. A mapping $\mathbf{f} : \wp(\mathbb{A}_1) \times \cdots \times \wp(\mathbb{A}_n) \to \wp(\mathbb{A}')$ is *inclusion monotone* iff for all sets $\mathbf{a}, \mathbf{b} \in \wp(\mathbb{A}_1) \times \cdots \times \wp(\mathbb{A}_n)$ s.t. $\mathbf{a} \subseteq \mathbf{b}$, $\mathbf{f}(\mathbf{a}) \subseteq \mathbf{f}(\mathbf{b})$.

Lemma 6. *Let* $f : \mathbb{A}_1 \times \cdots \times \mathbb{A}_n \to \mathbb{A}'$ *and* $\mathbf{g} : \wp(\mathbb{A}_1) \times \cdots \times \wp(\mathbb{A}_n) \to \wp(\mathbb{A}')$*. If* $\mathbf{g}$ *is inclusion-monotone and point-preserving of* f*, then it is an enclosure of* f*.*

Proof. We only have to prove that $\mathbf{g}$ is an enclosure of f, which means that for all $\mathbf{a} \in \wp(\mathbb{A}_1) \times \cdots \times \wp(\mathbb{A}_n)$, for all $b \in \mathbf{a}$, $f(b) \in \mathbf{g}(\mathbf{a})$.

Let $\mathbf{a} \in \wp(\mathbb{A}_1) \times \cdots \times \wp(\mathbb{A}_n)$ and $b \in \mathbf{a}$ arbitrary but fixed. We have to prove that $f(b) \in \mathbf{g}(\mathbf{a})$. Since $\mathbf{g}$ is point-preserving of f, we have: $\mathbf{g}(\{b\}) = \{f(b)\}$. Since $\mathbf{g}$ is inclusion-monotone and $\mathbf{a} \supseteq \{b\}$, $\mathbf{g}(\mathbf{a}) \supseteq \mathbf{g}(\{b\}) = \{f(b)\}$. Thus $f(b)$ is a member of $\mathbf{g}(\mathbf{a})$. □

Of course the converse of lemma 6 is not the case. The best possible enclosure of a mapping is defined as follows:

Definition 7. For every mapping $f : \mathbb{A}_1 \times \cdots \times \mathbb{A}_n \to \mathbb{A}'$, and for all $\mathbf{a_1} \in \wp(\mathbb{A}_1), \ldots, \mathbf{a_n} \in \wp(\mathbb{A}_n)$,

$$\mathbf{Ext}(f)(\mathbf{a_1}, \ldots, \mathbf{a_n}) := \{f(b_1, \ldots, b_n) | b_1 \in \mathbf{a_1}, \ldots, b_n \in \mathbf{a_n}\}$$

We call $\mathbf{Ext}(f)$ the *tight extension* of f.

It can be easily checked that this mapping is an enclosure and point-preserving of f. Furthermore it is inclusion-monotone.

5 Power Algebras

In the last section we have shown how to extend arbitrary mapping to sets. But applying this to the mapping $[\![\phi]\!]_{\mathbb{D}}$ is a very hard task in general. Fortunately we can solve this problem by using our interpretation of constraints as expressions in the algebra $\mathbb{D}$. In this algebra the symbols occurring in constraints are interpreted as simple mappings, and the complicated mapping $[\![\phi]\!]_{\mathbb{D}}$ is composed from these simple mapping. Thus, instead of composing these mappings and then extending the result to sets, we first extend the simple mappings to sets, and then compose the result. For this we introduce, for any algebra $\mathbb{A}$, according algebras over the power sets of the sorts of $\mathbb{A}$:

Definition 8. Given an algebra $\mathbb{A}$, a *power algebra* $\mathbb{A}^*$ of $\mathbb{A}$ consists of the sorts $(\wp(\mathbb{A}_i))_{i \in \mathbb{N}}$, and the same function symbols, where any set $\mathbb{A}_i$ in the signature of a function symbol is changed to $\wp(\mathbb{A}_i)$.

We can now adapt the definitions of enclosure, point-preserving and inclusion-monotone analogously to power algebras:

Definition 9. Given an algebra $\mathbb{A}$ and a power algebra $\boldsymbol{\mathbb{A}}^{*}$ of $\mathbb{A}$, $\boldsymbol{\mathbb{A}}^{*}$ is *point-preserving/an enclosure* of $\mathbb{A}$ iff for every expression e, $\llbracket e \rrbracket_{\boldsymbol{\mathbb{A}}^{*}}$ is point-preserving/ an enclosure of $\llbracket e \rrbracket_{\mathbb{A}}$. It is *inclusion-monotone* iff for every expression e, $\llbracket e \rrbracket_{\boldsymbol{\mathbb{A}}^{*}}$ is inclusion-monotone.

Now we show that these properties can be ensured by assigning appropriate mappings to the function symbols of the power algebra $\boldsymbol{\mathbb{A}}^{*}$ (i.e., by proving that properties on the left-hand side of figure 4 propagate to the right-hand side). All these theorems are generalizations of corresponding theorems that can be found in any textbook on interval arithmetic (e.g., [Moo66, Neu90]). The following lemma can be easily proven by induction over the structure of expressions.

$$\begin{tikzcd}
I_{\boldsymbol{\mathbb{A}}^{*}}(f) \arrow[r, "\text{compose expressions}"] \arrow[d, leftrightarrow] & \llbracket e \rrbracket_{\boldsymbol{\mathbb{A}}^{*}} \arrow[d, leftrightarrow] \\
I_{\mathbb{A}}(f) \arrow[r, "\text{compose expressions}"] & \llbracket e \rrbracket_{\mathbb{A}}
\end{tikzcd}$$

Figure 4: Power algebra properties

Lemma 10. *Given an algebra $\mathbb{A}$, a power algebra $\boldsymbol{\mathbb{A}}^{*}$ of $\mathbb{A}$ is point-preserving/an enclosure of $\mathbb{A}$ iff for all function symbols f, $I_{\boldsymbol{\mathbb{A}}^{*}}(f)$ is point-preserving/an enclosure of $I_{\mathbb{A}}(f)$. It is inclusion-monotone iff all function symbols f, $I_{\boldsymbol{\mathbb{A}}^{*}}(f)$ is inclusion-monotone*

For an algebra $\mathbb{A}$, we call the power algebra that assigns $\mathbf{Ext}(f)$ to every function symbol f, *tight power algebra* and denote it by $\boldsymbol{\mathbb{A}}^{\mathbf{Ext}}$. But, unfortunately, the tight power algebra looses information, because not for every expression e, $\llbracket e \rrbracket_{\boldsymbol{\mathbb{A}}^{\mathbf{Ext}}} = \mathbf{Ext}(\llbracket e \rrbracket_{\mathbb{A}})$. This can be illustrated using the following example: Take the expression $x \wedge \neg x$ over the Booleans, where x can be either true or false. Then the extensions of the mapping denoted by $x \wedge \neg x$ contains just false, but computation in the tight power algebra gives a set containing both true and false ($\{T, F\} \wedge \neg\{T, F\} = \{T, F\} \wedge \{T, F\} = \{T, F\}$). So, if we specialize the power algebra $\boldsymbol{\mathbb{A}}^{*}$ in diagram 4 to $\boldsymbol{\mathbb{A}}^{\mathbf{Ext}}$, then the property of being the tight extension does not propagate from the left-hand to the right-hand side.

The reason for this is, that power algebras do not take into account the fact, that variables in expressions can be the same (i.e., they depend on each other). Just like in interval mathematics, we call this problem *dependency problem*. Fortunately this problem occurs only in expressions that are not free.

Lemma 11. *Let $f : \mathbb{A}_1 \times \cdots \times \mathbb{A}_k \to \mathbb{A}'$, and let $e_1, \ldots, e_k$ be expressions such* *that they do not have any variable in common. Then, for a properly typed* $\mathbf{a}$,

$$\mathbf{Ext}(f)(\mathbf{Ext}(\llbracket e_1 \rrbracket_{\mathbb{A},V})(\mathbf{a}), \ldots, \mathbf{Ext}(\llbracket e_k \rrbracket_{\mathbb{A},V})(\mathbf{a})) = \mathbf{Ext}(f \circ (\llbracket e_1 \rrbracket_{\mathbb{A},V}, \ldots, \llbracket e_k \rrbracket_{\mathbb{A},V}))(\mathbf{a})$$

Proof. Since $e_1, \ldots, e_k$ do not have any variable in common, each e_i depends on a different set of variables. So the possible argument tuples to f are exactly the same, independent of whether we evaluate the $e_1, \ldots, e_k$ on sets separately, or in parallel:

$$\{(b_1, \ldots, b_k) | b_1 \in \mathbf{Ext}(\llbracket e_1 \rrbracket_{\mathbb{A},V})(\mathbf{a}), \ldots, b_k \in \mathbf{Ext}(\llbracket e_k \rrbracket_{\mathbb{A},V})(\mathbf{a})\} = \{(\llbracket e_1 \rrbracket_{\mathbb{A},V}(a), \ldots, \llbracket e_k \rrbracket_{\mathbb{A},V}(a)) | a \in \mathbf{a}\}$$

And thus:

$$\begin{aligned} &\mathbf{Ext}(f)(\mathbf{Ext}(\llbracket e_1 \rrbracket_{\mathbb{A},V})(\mathbf{a}), \ldots, \mathbf{Ext}(\llbracket e_k \rrbracket_{\mathbb{A},V})(\mathbf{a})) = \\ &\quad \{f(b_1, \ldots, b_k) | b_1 \in \mathbf{Ext}(\llbracket e_1 \rrbracket_{\mathbb{A},V})(\mathbf{a}), \ldots, b_k \in \mathbf{Ext}(\llbracket e_k \rrbracket_{\mathbb{A},V})(\mathbf{a})\} = \\ &\quad\quad \{f(\llbracket e_1 \rrbracket_{\mathbb{A},V}(a), \ldots, \llbracket e_k \rrbracket_{\mathbb{A},V}(a)) | a \in \mathbf{a}\} = \\ &\quad\quad\quad \mathbf{Ext}(f \circ (\llbracket e_1 \rrbracket_{\mathbb{A},V}, \ldots, \llbracket e_k \rrbracket_{\mathbb{A},V}))(\mathbf{a}) \end{aligned}$$

□

By induction over the structure of expressions we get:

Theorem 12. *For free expressions e,*

$$\llbracket e \rrbracket_{\mathbb{A}^{\mathbf{Ext}}} = \mathbf{Ext}(\llbracket e \rrbracket_{\mathbb{A}}).$$

Definition 13. Two expressions e_1 and e_2 are *equivalent* iff $\llbracket e_1 \rrbracket_{\mathbb{A}} = \llbracket e_2 \rrbracket_{\mathbb{A}}$.

We know that, in general, computation in $\mathbb{A}^{\mathbf{Ext}}$ can lose information for non-free expressions. But it can happen that there is an equivalent expression that yields a tighter evaluation, or even an equivalent expression that has no dependency problem at all:

Definition 14. An expression e is *information preserving* iff

$$\llbracket e \rrbracket_{\mathbb{A}^{\mathbf{Ext}}} = \mathbf{Ext}(\llbracket e \rrbracket_{\mathbb{A}}).$$

Of course, given an expression e, we are interested in finding an equivalent but information-preserving expression. If there is no such information-preserving expression, then we would at least like to find expressions where the dependency problem is as small as possible, or at least expressions where it is as small as possible for a certain certain class of input sets.

In quantified constraint solving the dependency problem has only been addressed for sub-theories up to now: The case of the real numbers with function symbols such as $+$, $-$, $\times$ is an important question in the area of interval mathematics (see e.g., [Kea96, Moo66, Sta96], or [Cap79] for symbolic transformations of expressions to reduce the dependency problem, or [ACS94, Han75] for special representations of intervals to deal with the problem). For the case of the Boolean values with negation, conjunction and disjunction (this corresponds to a many-valued logic [Bel77]) we have developed a new algorithm for computing information-preserving expressions. This will be reported elsewhere.

6 Representing Power Algebras by Intervals

Of course the representation of arbitrary power sets of the sorts of a domain $\mathbb{A}$ is in general an even harder problem than representing single elements. Thus we choose special sets that are easily representable and round [Alb80, KM80, KM81] other sets. We use rounding operators $R_i : \wp(\mathbb{A}_i) \to \mathbb{A}'_i$, for which:

- for all $\mathbf{a} \in \wp(\mathbb{A}_i)$, $\mathbf{a} \subseteq R_i(\mathbf{a})$ (rounding increases uncertainty)
- for all $\mathbf{a} \in \wp(\mathbb{A}_i)$, $R_i(R_i(\mathbf{a})) = R_i(\mathbf{a})$ (do not round elements that are already rounded)
- for all $\mathbf{a}, \mathbf{b} \in \wp(\mathbb{A}_i)$, $\mathbf{a} \subseteq \mathbf{b}$ implies $R_i(\mathbf{a}) \subseteq R_i(\mathbf{b})$ (rounding is inclusion monotone)

Given a power algebra $\mathbb{A}^*$ and rounding operators R_i we can then construct a new power algebra $\mathbb{A}'$ in which for all function symbols f with signature $\wp(\mathbb{A}_{i_1}) \times \cdots \times \wp(\mathbb{A}_{i_n}) \to \wp(\mathbb{A}_j)$, for all $\mathbf{a} \in \wp(\mathbb{A}_{i_1}) \times \cdots \times \wp(\mathbb{A}_{i_n})$, $I_{\mathbb{A}'}(f)(\mathbf{a}) = R_j(I_{\mathbb{A}^*}(f)(\mathbf{a}))$. In this case we say that $\mathbb{A}'$ *is the approximation of* $\mathbb{A}^*$ *implied by the* R_i. Then we can restrict ourselves to rounded sets for computation.

If we have a (partial) order $\leq$ on the $\mathbb{A}_i$ then we can choose the set of intervals $[\underline{a}, \overline{a}] := \{x \mid \underline{a} \leq x \leq \overline{a}\}$ with endpoints in $\mathbb{A}_i \cup \{-\infty, \infty\}$ as such a representation. This is a quite simple, and thus efficient notation for denoting sets [NN97]. So we also use them for building power algebras. Here we require that in $\mathbb{A}_i$ the biggest lower bound and smallest upper bound is defined for any set (in this case the $\mathbb{A}_i$ are complete lattices). Now we can do rounding by taking the smallest superset that is an interval. For any set $\mathbf{a}$ we will denote this by $\mathit{Encl}(\mathbf{a})$. It can be easily checked that Encl is a rounding operator. Now we can define the tightest possible extension up to rounding by Encl.

Definition 15. For every mapping $f : \mathbb{A}_1 \times \cdots \times \mathbb{A}_n \to \mathbb{A}'$, for all $\mathbf{a_1} \in \wp(\mathbb{A}_1), \ldots, \mathbf{a_n} \in \wp(\mathbb{A}_n)$,

$$\mathbf{ExtI}(f)(\mathbf{a_1}, \ldots, \mathbf{a_n}) := \mathit{Encl}(\mathbf{Ext}(f)(\mathbf{a_1}, \ldots, \mathbf{a_n}))$$

We call $\mathbf{ExtI}(f)$ the *tight interval extension* of f

For certain mappings f, $\mathbf{ExtI}(f)$ and $\mathbf{Ext}(f)$ coincide, for example over the reals if f is continuous (see [Kos98] for a discussion of a similar question from the algebraic point of view). Otherwise $\mathbf{ExtI}(f)$ overestimates $\mathbf{Ext}(f)$, but is still point-preserving, and an enclosure of f, and also inclusion-monotone.

For any algebra $\mathbb{A}$ we can use the approximation of $\mathbb{A}^{\mathbf{Ext}}$ implied by Encl, which we denote by $\mathbb{A}^{\mathbf{ExtI}}$. By lemma 10, $\mathbb{A}^{\mathbf{ExtI}}$ is again point-preserving, and an enclosure of $\mathbb{A}$, and inclusion-monotone. But the analogous version of theorem 12, that for free expressions e, $[\![e]\!]_{\mathbb{A}^{\mathbf{ExtI}}} = \mathbf{ExtI}([\![e]\!]_{\mathbb{A}})$ does not hold. This can be seen on the following example: Let f be such that $f(x) = -1$ if $x < 0$, and $f(x) = 1$ otherwise. Let g be such that $g(x) = 1$ if $x = 0$, and $g(x) = 0$ otherwise. Then $\mathit{Encl}(g(f([-1, 1]))) = \mathit{Encl}(g(\{-1, 1\})) = [0, 0]$, but $\mathit{Encl}(g(\mathit{Encl}(f([-1, 1])))) = [0, 1]$.

But in the case where, for all mappings f, $\mathbf{ExtI}(f)$ and $\mathbf{Ext}(f)$ coincide, no information is lost by switching to intervals, and for free expressions e, even $[\![e]\!]_{\mathbb{A}^{\mathbf{ExtI}}} = \mathbf{Ext}([\![e]\!]_{\mathbb{A}})$.

In domains with infinite sorts we often do not want to allow arbitrary elements of such a sort to form interval endpoints but only a finite subset. For example over $\mathbb{R}$ one often just allows floating point values $\mathbb{F}$. In this case instead of rounding to the smallest interval superset, one does rounding to the smallest interval superset with endpoints in the chosen subset of the sort (e.g., floating point endpoints).

7 Application to Quantified Constraint Solving

We use the algebra $\mathbb{D}$, as described in section 3, as a starting point and approximate it by the tight power algebra $\mathbf{D}^{\mathbf{Ext}}$. Here it is easy to deal with approximate constants. As shown in section 3, in $\mathbb{D}$ they are 0-ary function symbols with signature $\mathbb{D}_n^{\mathbb{R}}$ that are interpreted as constant functions. So we can simply assign the according set of constant functions to this function symbol in the power algebra.

Provided that all constants are exact, by lemma 10, $\mathbf{D}^{\mathbf{Ext}}$ is point-preserving, so $[\![\phi]\!]_{\mathbf{D}^{\mathbf{Ext}}}(\{d_1^{id}\}, \ldots, \{d_n^{id}\})$ yields the (exact) solution set of ϕ. Furthermore $\mathbf{D}^{\mathbf{Ext}}$ is an enclosure of $\mathbb{D}$, so $[\![\phi]\!]_{\mathbf{D}^{\mathbf{Ext}}}(\mathbf{d_1^{id}}, \ldots, \mathbf{d_n^{id}})$, where $\mathbf{d_1^{id}}, \ldots, \mathbf{d_n^{id}}$ are approximations of the identity term functions, yields an approximation of the exact solution set of ϕ. Moreover $[\![\phi]\!]_{\mathbf{D}^{\mathbf{Ext}}}$ is inclusion-monotone, so we can improve the approximation $[\![\phi]\!]_{\mathbf{D}^{\mathbf{Ext}}}(\mathbf{d_1^{id}}, \ldots, \mathbf{d_n^{id}})$ by improving the approximations $\mathbf{d_1^{id}}, \ldots, \mathbf{d_n^{id}}$.

These properties allow us to apply the incremental approximation scheme from the end of section 2. Since $\mathbf{D}^{\mathbf{Ext}}$ is an enclosure of $\mathbb{D}$, the method yields a correct result, if it terminates. Since $\mathbf{D}^{\mathbf{Ext}}$ is inclusion-monotone, it gets nearer to the correct result in each turn of the loop. However, the method still does not necessarily terminate. The problem is that, even if the size of the argument sets tends to the one-elementary set and all constants are exact, the size of the result set does not necessarily tend to a one-elementary set. The reason for this lies in the quantifiers, and does not occur for constraint without quantifiers. In order to make the method always terminate on constraints with quantifiers, also, one needs to replace the classical quantifiers by different, approximate quantifiers.

For lack of space, we cannot give a detailed description of the semantics of approximate quantifiers here (see [Rat00c]), 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 these quantifiers cannot assure the termination of the method. 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 $\overline{q}$ the result is T, if the quantified solution set is smaller or equal $\underline{q}$ the result is F, and otherwise the result is allowed to be any of T or F (see figure 5). A formal definition of the semantics and properties of approximate quantifiers requires considerably more effort [Rat00c].

The question remains, how to represent subsets of $\mathbb{D}_1^{\mathbb{B}}, \mathbb{D}_2^{\mathbb{B}}, \ldots$ and subsets of $\mathbb{D}_1^{\mathbb{R}}, \mathbb{D}_2^{\mathbb{R}}, \ldots$ on computers (i.e., how to represent the power algebra $\mathbf{D}^{\mathbf{Ext}}$). Of

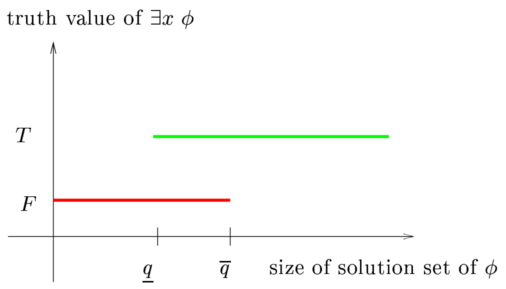

Figure 5: Approximate Quantifiers

course there are many possibilities for this. Here we describe one that has been a successfully starting point for our implementation [Rat00a].

As proposed in section 6, we use interval representation. We take the usual order on the real numbers and the order $F < T$ on the Booleans and extend it element-wise to all elements of the sorts $\mathbb{D}_1^{\mathbb{B}}, \mathbb{D}_2^{\mathbb{B}}, \ldots$ and $\mathbb{D}_1^{\mathbb{R}}, \mathbb{D}_2^{\mathbb{R}}, \ldots$. This enables us to compute in $\mathbb{D}^{\mathbf{ExtI}}$ by using interval representation $[\underline{p}, \overline{p}]$ for subsets of the sorts, where the bounds $\underline{p}$ and $\overline{p}$ are functions $\mathbb{R}^n \to \mathbb{R}$ and $\mathbb{R}^n \to \mathbb{B}$.

We need a representation of the bounds of these intervals, for which $[\![\phi]\!]_{\mathbb{D}^{\mathbf{ExtI}}}$ is easy to compute. It can be easily shown that, if the inputs to computations in $\mathbb{D}^{\mathbf{ExtI}}$ are intervals of functions in which $[\underline{p}(a), \overline{p}(a)]$ is constant on boxes (i.e., Cartesian products of intervals) then all computation results also have this form. So we can represent all functions by sets of pairs that consist of a real box plus a Boolean or real interval.

See figure 6 for an example of uncertainty propagation using this representation. For computing the value of the constraint $\forall x\ [xx \geq 0]$ we start with an approximation of the identity term function of x (upper left corner, the intervals are the second element of each pair contained in the above set of pairs). We multiply two copies of this term function in $\mathbb{D}^{\mathbf{ExtI}}$ by applying $\mathbf{ExtI}(\times)$ to them and arrive at an approximation of the term function of xx (upper right corner). Here one can see the effect of the dependency problem: Multiplication does not detect that its arguments are identical, and thus the approximation contains functions with negative values. Now we apply $\leq$ to this term function (lower left corner) and detect that for the boxes $[[-\infty, -1]]$ and $[[1, \infty]]$ the solution set is surely true (i.e., the interval $[T, T]$), and in $[-1, 1]$ we do not have information (i.e., the interval $[F, T]$). Now we apply universal quantification to this approximate solution set: The result (lower right corner) depends on the value of x on the whole real line, and we cannot infer any information because of the interval $[F, T]$ assigned to the box $[[-1, 1]]$ of the argument. The situation changes if we make the approximation of the identity term function of x smaller by bisecting the box $-1, 1$ into two pieces with values $[-1, 0]$ and $[0, 1]$, respectively: Then multiplication creates the value $[0, 1]$ for both of them, and we can infer that the the approximate solution set of $x\ [xx \geq 0]$ contains all real numbers and thus the whole constraint is true.

More complicated examples show, that the major computational complexity

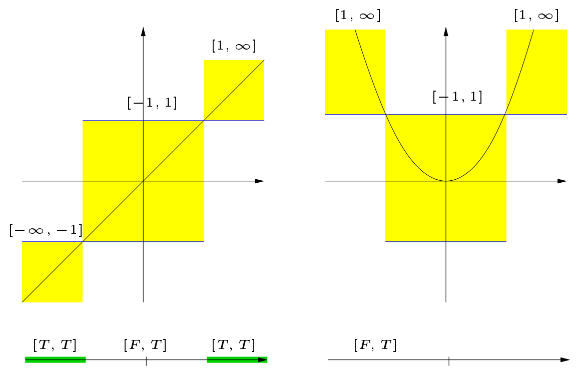

Figure 6: Propagation Example

lies in propagating approximate solution sets over quantifiers — these represent the only mappings where the result does not only depend on one value of each argument, but on infinitely many values.

8 Related Work

Set theory is one of the oldest methods for dealing with uncertainty, as can be seen from the ubiquitous formulation "for any a in A". Cantor constructed the real numbers by sequences of nested intervals [Can70]. Other approaches to modeling uncertainty in mathematics are, for example, functions from the reals to sets of reals [AF90]. In contrast to our approach, classical mathematics is more interested in infinitary notions like limit and continuity, and less in actual computation with approximate values.

Especially for dealing with uncertainty, set theory has been extended to fuzzy set theory [Zad65, Zim91]. The area of interval mathematics (see e.g., [Kea96, Moo66, Neu90, Sta96]) uses real intervals for dealing with uncertainty. Examples of other domains, where sets have been used for representing uncertainty, are the complex numbers [Hen71, KU80, Nic80, RL71], finite domains (constraint programming [Bar, MS98], many-valued logic [Urq86, Bel77]) and functions [BH98, CR91, KM84].

A large part of our approach is a generalization of interval mathematics, and a lot of theorems and proofs are straightforward generalizations of corresponding theorems there. Several other authors also provided generalizations of interval arithmetic. Apostolatos and Karabatzos [AK80] generalized various basic definitions and theorems from interval arithmetic to arbitrary sets in a similar way as in Section 4, and applied their theory to notions such as fixed-point iteration. Other authors provided general algebraic interval structures [Kla76, Klu81], or developed and applied general theories of rounding in connection with interval arithmetic [Alb80, KM80, KM81]. The observation that interval arithmetic is related to 3-valued logic, also appears quite often in the literature (see for example [Jah80]). Similar many-valued logics arise in artificial intelligence [Bel77].

In our terminology this is just a construction of an power algebra from the Booleans.

In computer science similar approaches have been used for modeling nondeterminism in abstract data types [WM97, Hes88, Hus93], where usually multialgebras (i.e., algebras over functions $A_1 \times \cdots \times A_n \to \wp(A'))$ are used. There one usually starts from a nondeterministic specification and then tries to arrive at a (often deterministic) implementation. In contrast to that, in our approach we start from an deterministic (exact) specification and then approximate it via a nondeterministic (approximate) implementation. Furthermore the theory of power domains has been developed for modeling sets and non-determinism in functional languages [Plo76, Smy78].

The traditional way of dealing with uncertainty is probability theory, where probability distributions are used for this purpose. For example [Ber96] and [KFG$^+$99] study the connection of this approach to the set based approach.

The starting point for our work was the work by Hong and Neubacher [Hon95, HN96, Neu97] on approximate quantified constraint solving based on interval arithmetic. Their work raised a lot of interesting problems, for example: How to transform the input formulas in order to optimize the performance? Which data-structures to use? How to ensure convergence? Which search strategies to use? These questions were hard to study because of the intricacy of their algorithms (over 10 pages of pseudo-code). Their work can be viewed as a special instantiation of our framework. Our contribution provides a clear theoretical basis, and structures their algorithms by abstracting away details that are unnecessary for studying the preceding questions. This already helped in answering some of them [Rat00c, Rat00b, Rat98] and allowed us to design new and more efficient algorithms based on the gained insight. So our framework seems to be useful for further considerations there, but also in other areas which deal with uncertainty (e.g., interval mathematics, many-valued logic).

9 Conclusion

We have shown how approximate quantified constraint solving gives rise to uncertainty propagation in a heterogeneous algebra. Since heterogeneous algebras are ubiquitous in computing, we have developed a general framework for propagating uncertainty in them by introducing the notion of heterogeneous power algebra. We have studied properties of this notion interesting for approximate computation and have identified the dependency problem as crucial.

The generality of the notion of heterogeneous algebra makes our framework for approximate computation applicable in various other areas — for example, interval arithmetic and certain many-valued logics [Bel77, NRSS97] are an instance of the scheme. Also the application to quantified constraint solving can be used over different domains than the real numbers.

The application of the resulting method to quantified constraint solving yields an approximation method with various favorable properties: First, it allows us to process input with inexact constants. Second, we can deal with an undecidable problem that has extremely high complexity for subproblems. Third, we can choose approximations that result in simple numerical output. Furthermore the algorithm is tunable in the sense, that the user can decide upon the trade-off between precision and efficiency.

Based on the work in this paper we have implemented a method for approximate quantified constraint solving [Rat00a], that builds on earlier work by Hong, Neubacher, Stahl and others [HN96, HS94, HNS94]. For this we use special data-structures that allow efficient execution of the involved operations [Rat00b]. The formal framework developed in this paper makes it now possible to further improve the implementation, by allowing the study of questions such as:

- Which search strategies and heuristics should be used, for example for deciding when and how to improve the approximation of which input set?
- Which alternative representations could be used for representing sets of solution sets? For example one could try to represent pieces of solution sets by real polytopes instead of real boxes.
- How to preprocess the input constraints to improve efficiency? This includes transformations for minimizing the dependency problem and propagation time. One promising approach might be to apply symbolic simplifications in the style of [DS97, Wei88].
- How to do computation if the constraints are built incrementally (e.g., coming from a Constraint Logic Programming [JL87] system). Our method seems to be very well suited for this.

A Equivalence of Meaning of Constraints

Here we prove the equivalence of the meaning of constraints as expressions in the algebra $\mathbb{D}$ and as logical formulas. For this let us denote the term function in $\mathbb{D}^{\mathbb{R}}_{|V|}$ and solution set in $\mathbb{D}^{\mathbb{B}}_{|V|}$ given by the logical meaning of a term or formula ϕ respectively, by $s_V(\phi)$, where V is a superset of the set of free variables of ϕ. Sometimes we drop V and write $s(\phi)$ for $s_V(\phi)$ where V is the set of all variables occurring in ϕ. We assume that the reader is familiar with the exact formal definition of $s_V(\phi)$.

Theorem 16. *For every constraint ϕ in variables V, for $d = (d_1^{id}, \ldots, d_{|V|}^{id})$,*

$$[\![\phi]\!]_{\mathbb{D},V}(d) = s_V(\phi)$$

Proof. We have to prove that for all $(p_1, \ldots, p_{|V|})$ in $\mathbb{R}^{|V|}$,

$$[\![\phi]\!]_{\mathbb{D},V}(d)(p_1, \ldots, p_{|V|}) = s_V(\phi)(p_1, \ldots, p_{|V|}).$$

We proceed by induction on the structure of formulas.

Base Cases:

- For any formula x_i,

$$[\![x_i]\!]_{\mathbb{D},V}(d)(p_1, \ldots, p_{|V|}) = s_V(x_i)(p_1, \ldots, p_{|V|})$$

By definition of identity term function, $[\![x_i]\!]_{\mathbb{D},V}(d)(p_1, \ldots, p_{|V|}) = p_i$, which is equal to the right-hand side of the preceding equation.

- For any constant c,

$$[\![c]\!]_{\mathbb{D},V}(d)(p_1, \ldots, p_{|V|}) = s_V(c)(p_1, \ldots, p_{|V|})$$

Both sides of this equation are equal to the constant denoted by c.

Induction Steps: – For any formula of the form $f(\phi_1,\ldots,\phi_k)$, where f is a k-ary logical connective:

$$[\![f(\phi_1,\ldots,\phi_k)]\!]_{\mathbb{D},V}(d)(p_1,\ldots,p_{|V|}) = s_V(f(\phi_1,\ldots,\phi_k))(p_1,\ldots,p_{|V|})$$

This can be proved as follows:

$$\begin{array}{rl}
[\![f(\phi_1,\ldots,\phi_k)]\!]_{\mathbb{D},V}(d) = & \text{def. of } [\![\ ]\!]_{\mathbb{D}} \\
(I_{\mathbb{D}}(f)\circ([\![\phi_1]\!]_{\mathbb{D},V},\ldots,[\![\phi_k]\!]_{\mathbb{D},V}))(d) = & \text{def. of } \circ \\
I_{\mathbb{D}}(f)([\![\phi_1]\!]_{\mathbb{D},V}(d),\ldots,[\![\phi_k]\!]_{\mathbb{D},V}(d)) = & \text{def. of } I_{\mathbb{D}} \\
I_{\mathbb{B}}(f)\circ([\![\phi_1]\!]_{\mathbb{D},V}(d),\ldots,[\![\phi_k]\!]_{\mathbb{D},V}(d)) = & \text{induction hypothesis} \\
I_{\mathbb{B}}(f)(s_V(\phi_1),\ldots,s_V(\phi_k)) = & \text{def. of s} \\
s_V(f(\phi_1,\ldots,\phi_k)) &
\end{array}$$

– For any formula of the form $f(\phi_1,\ldots,\phi_k)$, where f is a k-ary predicate or function symbol:

$$[\![f(\phi_1,\ldots,\phi_k)]\!]_{\mathbb{D},V}(d)(p_1,\ldots,p_{|V|}) = s_V(f(\phi_1,\ldots,\phi_k))(p_1,\ldots,p_{|V|})$$

This can be proved as follows:

$$\begin{array}{rl}
[\![f(\phi_1,\ldots,\phi_k)]\!]_{\mathbb{D},V}(d) = & \text{def. of } [\![\ ]\!]_{\mathbb{D}} \\
(I_{\mathbb{D}}(f)\circ([\![\phi_1]\!]_{\mathbb{D},V},\ldots,[\![\phi_k]\!]_{\mathbb{D},V}))(d) = & \text{def. of } \circ \\
I_{\mathbb{D}}(f)([\![\phi_1]\!]_{\mathbb{D},V}(d),\ldots,[\![\phi_k]\!]_{\mathbb{D},V}(d)) = & \text{def. of } I_{\mathbb{D}} \\
I_{\mathbb{R}}(f)\circ([\![\phi_1]\!]_{\mathbb{D},V}(d),\ldots,[\![\phi_k]\!]_{\mathbb{D},V}(d)) = & \text{induction hypothesis} \\
I_{\mathbb{R}}(f)(s_V(\phi_1),\ldots,s_V(\phi_k)) = & \text{def. of s} \\
s_V(f(\phi_1,\ldots,\phi_k)) &
\end{array}$$

– For any formula of the form $(\forall x_i)\phi$,

$$[\![(\forall x_i)\phi]\!]_{\mathbb{D},V}(d)(p_1,\ldots,p_{|V|}) = s_V((\forall x_i)\phi)(p_1,\ldots,p_{|V|})$$

This can be proved as follows:

$$\begin{array}{rl}
[\![(\forall x_i)\phi]\!]_{\mathbb{D},V}(d)(p_1,\ldots,p_{|V|}) = & \text{def. of } [\![\ ]\!]_{\mathbb{D}} \\
I_{\mathbb{D}}(\forall x_i)([\![\phi]\!]_{\mathbb{D},V}(d))(p_1,\ldots,p_{|V|}) = & \text{def. of } I_{\mathbb{D}} \\
(\text{for all } q\in\mathbb{R}, [\![\phi]\!]_{\mathbb{D},V}(d)(p_1,\ldots,q,\ldots,p_{|V|}) = T) = & \text{induction hyp.} \\
(\text{for all } q\in\mathbb{R},\ s_V(\phi)(p_1,\ldots,q,\ldots,p_{|V|}) = T) = & \text{def. of s} \\
s_V((\forall x)\phi)(p_1,\ldots,p_{|V|}) &
\end{array}$$

– For any formula of the form $(\exists x_i)\phi$,

$$[\![(\exists x_i)\phi]\!]_{\mathbb{D},V}(d)(p_1,\ldots,p_{|V|}) = s_V((\exists x_i)\phi)(p_1,\ldots,p_{|V|})$$

Proof similar to previous case.

□

Acknowledgments

Most of this research was done as a part of the author's Ph.D. work. The author thanks his Ph.D. advisor, Hoon Hong, for all the guidance that is reflected in both the structure and content of this paper. The author also is grateful for the detailed and constructive criticism made by a referee. This work was supported by the Austrian Science Fund FWF in the frame of the project SFB F1303.

References

[ACS94] M. V. A. Andrade, J. L. D. Comba, and J. Stolfi. Affine arithmetic. In *Abstracts of the International Conf. on Interval and Computer-Algebraic Methods in Science and Engineering (INTERVAL/94)*, pages 36–40, 1994.

[AF90] Jean-Pierre Aubin and Helene Frankowska. *Set-valued Analysis.* Birkhäuser, Boston, 1990.

[AK80] N. Apostolatos and G. Karabatzos. Set functions and applications. In Karl L. E. Nickel, editor, *Interval Mathematics*, pages 1–24. Academic Press, 1980.

[Alb80] R. Albrecht. Roundings and approximations in ordered sets. In G. Alefeld and R.D. Grigorieff, editors, *Fundamentals of Numberical Computation (Computer-Oriented Numerical Analysis)*, volume 2 of *Computing Supplementum*, pages 17–31. Springer, 1980.

[Bar] Roman Bartak. Guide to constraint programming. `http://kti.ms.mff.cuni.cz/~bartak/constraints/index.html`.

[BCK97] Maria Beltran, Gilbert Castillo, and Vladik Kreinovich. Algorithms that still produce a solution (maybe not optimal) even when interrupted: Shary's idea justified. *Reliable Computing*, 3(3):39–53, 1997.

[Bel77] N.D. Belnap, Jr. A useful four-valued logic. In J. Michael Dunn and G. Epstein, editors, *Modern Uses of Multiple- Valued Logic*, pages 8–37. Reidel, 1977.

[Ber96] D. Berleant. Automatically verified arithmetic on probability distributions and intervals. In R. Baker Kearfott and Vladik Kreinovich, editors, *Applications of Interval Computations.* Kluwer, 1996.

[BH98] Martin Berz and Georg Hoffstätter. Computation and application of taylor polynomials with interval remainder bounds. *Reliable Computing*, 4:83–97, 1998.

[Can70] Georg Cantor. Über einen die geometrischen Reihen betreffenden Lehrsatz. *Crelles Journal f. Mathematik*, 72:130–138, 1870.

[Cap79] G. Caplat. Symbolic preprocessing in interval function computing. In *Symbolic and algebraic computation, EUROSAM '79*, LNCS 72, pages 369–382, Marseille, 1979.

[CH91] George E. Collins and Hoon Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12:299–328, 1991. Also in [CJ98].

[CJ98] B. F. Caviness and J. R. Johnson, editors. *Quantifier Elimination and Cylindrical Algebraic Decomposition.* Texts and Monographs in Symbolic Computation. Springer, 1998.

[Col75] George 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 [CJ98].

[CR91] George F. Corliss and Louis B. Rall. Computing the range of derivatives. In Edgar W. Kaucher, Svetoslav M. Markov, and Günter Mayer, editors, *Computer Arithmetic, Scientific Computation, and Mathematical Modelling*, volume 12 of *IMACS Annals on Computing and Applied Mathematics*, pages 195–212. J. C. Baltzer, Basel, 1991.

[DH88] J. H. Davenport and J. Heintz. Real quantifier elimination is doubly exponential. *Journal of Symbolic Computation*, 5:29–35, 1988.

[DS97] Andreas Dolzmann and Thomas Sturm. Simplification of quantifier-free formulae over ordered fields. *Journal of Symbolic Computation*, 24(2):209–231, 1997.

[dW93] Van der Waerden. *Algebra I.* Springer Verlag, 9th edition, 1993.

[EFT84] H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic.* Springer Verlag, 1984.

[EM85] Hartmut Ehrig and Bernd Mahr. *Algebraic Specification.* Springer Verlag, 1985.

[Han75] E. R. Hansen. A generalized interval arithmetic. In K. Nickel, editor, *Interval Mathematics*, volume 29 of *Lecture Notes in Computer Science*, pages 7–18, 1975.

[Hen71] Peter Henrici. Circular arithmetic and the determination of polynomial zeros. In *Conf. Appl. numerical Analysis*, volume 228 of *Lecture Notes Math.*, pages 86–92. Springer, 1971.

[Hes88] Wim H. Hesselink. A mathematical approach to nondeterminism in data types. *ACM Transactions on Programming Languages and Systems*, 10(1):87–117, 1988.

[HJ98] C.A.R. Hoare and He Jifeng. *Unifying Theories of Programming.* Series in Computer Science. Prentice Hall, 1998.

[HN96] Hoon Hong and Andreas Neubacher. Approximate quantifier elimination. In *Proceedings of IMACS-ACA'96*, 1996.

[HNS94] Hoon Hong, Andreas Neubacher, and Volker Stahl. The STURM library manual – a C++ library for symbolic computation. Technical Report 94-30, RISC Linz, 1994.

[Hoa96] C.A.R. Hoare. Unifying theories: A personal statement. *ACM Computing Surveys*, 28A(4), December 1996.

[Hon92] H. Hong. Heuristic search strategies for cylindrical algebraic decomposition. In Jacques Calmet et al., editors, *Proceedings of Artificial Intelligence and Symbolic Mathematical Computing, Springer Lecture Notes in Computer Science 737*, pages 152–165, 1992.

[Hon95] Hoon Hong. Symbolic-numeric methods for quantified constraint solving. In *International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics SCAN-95*, 1995. Invited Talk.

[HS94] Hoon Hong and Volker Stahl. Safe starting regions by fixed points and tightening. *Computing*, 53:323–335, 1994.

[Hus93] H. Hussmann. *Nondeterminism in Algebraic Specifications and Algebraic Programs.* Birkhäuser, 1993.

[Jah80] Karl-Udo Jahn. The importance of 3-valued notions for interval mathematics. In Karl L. E. Nickel, editor, *Interval Mathematics*, pages 75–98. Academic Press, 1980.

[JL87] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages, Munich, Germany*, pages 111–119. ACM, January 1987.

[Kea96] R. Baker Kearfott. Interval computations: Introduction, uses, and resources. *Euromath Bulletin*, 2(1):95–112, 1996.

[KFG$^+$99] Vladik Kreinovich, Scott Ferson, Lev Ginzburg, Harry Schulte, Matthew R. Barry, and Hung T. Nguyen. From interval methods of representing uncertainty to a general description of uncertainty. In *Proceeedings of the International Conference on Information Technology*, Bhubaneshwar, India, 1999.

[Kla76] Dieter Klaua. Intervallstrukturen geordneter Körper. (German). *Math. Nachr.*, 75:319–326, 1976.

[Klu81] U. Klug. Verallgemeinerte Intervallräume (German). *Math. Nachr.*, 102:347–359, 1981.

[KM80] U. W. Kulisch and W.L. Miranker. Arithmetic operations in interval spaces. In G. Alefeld and R.D. Grigorieff, editors, *Fundamentals of Numberical Computation (Computer-Oriented Numerical Analysis)*, volume 2 of *Computing Supplementum*, pages 51–67. Springer, 1980.

[KM81] Ulrich W. Kulisch and Willard L Miranker. *Computer Arithmetic in Theory and Practice.* Academic Press, 1981.

[KM84] Edgar W. Kaucher and Willard L. Miranker. *Self-Validating Numerics for Function Space Problems.* Academic Press, 1984.

[Kos98] Olga Kosheleva. When is the product of intervals also an interval. *Reliable Computing*, 4:179–190, 1998.

[KU80] R. Klatte and Ch. Ullrich. Complex sector arithmetic. *Computing*, 24:139–148, 1980.

[Mis93] B. Mishra. *Algorithmic Algebra.* Springer Verlag, 1993.

[Moo66] R. E. Moore. *Interval Analysis.* Prentice Hall, Englewood Cliffs, NJ, 1966.

[MS98] Kim Marriott and Peter J. Stuckey. *Programming with Constraints: an Introduction.* MIT Press, 1998.

[Neu90] Arnold Neumaier. *Interval Methods for Systems of Equations.* Cambridge Univ. Press, Cambridge, 1990.

[Neu97] Andreas Neubacher. *Parametric Robust Stability by Quantifier Elimination.* PhD thesis, Research Institute for Symbolic Computation - Universität Linz, October 1997.

[Nic80] K. Nickel. Arithmetic of complex sets. *Computing*, 24:97–105, 1980.

[NN97] Monica Nogueira and Amarendra Nandigam. Why interval? Because if we allow other sets, tractable problems become intractable. Technical Report UTEP-CS-97-7, University of Texas at El Paso, El Paso, 1997.

[NRSS97] Alioune Ngom, Corina Reischer, Dan A. Simovici, and Ivan Stojmenović. Set-valued logic algebra: A carrier computing foundation. *Multiple Valued Logic*, 2(3):183–216, 1997.

[Plo76] G. Plotkin. A power domain construction. *SIAM Journal on Computing*, 5(3):452–487, 1976.

[Rat98] Stefan Ratschan. *Approximate Constraint Logic Programming.* PhD thesis, RISC-Linz, 1998.

[Rat00a] Stefan Ratschan. Approximate quantified constraint solving (AQCS). `http://www.risc.uni-linz.ac.at/research/software/AQCS`, 2000. Software package.

[Rat00b] Stefan Ratschan. Approximate quantified constraint solving by cylindrical box decomposition. In *6th IMACS Conference on Applications of Computer Algebra: Interval and Computer-Algebraic Methods in Science and Engineering*, St. Petersburg, 2000.

[Rat00c] Stefan Ratschan. Convergence of quantified constraint solving by approximate quantifiers. Technical Report 00-23, Research Institute for Symbolic Computation (RISC) - Linz, 2000. Submitted for Publication.

[Ren92] James 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. Part I-III.

[RL71] J. Rokne and P. Lancaster. Complex interval arithmetic. *Comm. ACM*, 14:111–112, 1971.

[SB81] H.P. Sankappanavar Stanley Burris. *A course in universal algebra.* Springer Verlag, 1981.

[Smy78] M. B. Smyth. Power domains. *Journal of Computer and System Sciences*, 16, 1978.

[Sta96] Volker Stahl. *Interval Methods for Bounding the Range of Polynomials and Solving Systems of Nonlinear Equations.* PhD thesis, Research Institute for Symbolic Computation, Johannes Kepler University, A-4040 Linz, Austria, 1996.

[Tar51] Alfred Tarski. *A Decision Method for Elementary Algebra and Geometry.* Univ. of California Press, Berkeley, 1951. Also in [CJ98].

[Urq86] Alasdair Urquhart. Many-valued logic. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic, Vol. III: Alternatives in Classical Logic*, chapter III.2, pages 71–116. D. Reidel Publishing Company, 1986.

[Wei88] Volker Weispfenning. The complexity of linear problems in fields. *Journal of Symbolic Computation*, 5(1–2):3–27, 1988.

[WM97] Michał Walicki and Sigurd Meldal. Algebraic approaches to nondeterminism: An overview. *ACM Computing Surveys*, 29(1), 1997.

[Zad65] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[Zim91] H.-J. Zimmermann. *Fuzzy Set Theory - and Its Applications.* Kluwer, 1991.