Determining the Relationship among Sets

Hoon Hong and Stefan Ratschan

December 20, 1996

Abstract

Doing entailment/disentailment checks on sets is an important operation in constraint programming. This problem is generalized to the problem of answering queries about arbitrary sets using a certain query language. Furthermore an efficient algorithm for solving this problem for sets described by arbitrary non-linear polynomials is presented.

1 Introduction

One often wants to know how given sets relate to each other. For example one might ask: Is the set S_1 a subset of S_2? Are the sets S_1 and S_2 disjoint? In this paper we will devise algorithms for answering such queries where the sets are described by polynomial inequalities over the real numbers. In fact, most of the algorithms can be easily generalized to sets described in other ways. However we will not do this here, because it would needlessly complicate the presentation.

An important application of this problem arises in constraint logic programming ([6],[7], [8], [2]), where consistency and redundancy (entailment/disentailment) checks on sets described by constraints have to be done efficiently.

2 Problem

As already mentioned in the introduction we want to find an algorithm that answer queries about a given tuple of sets $S = (S_1, \ldots, S_n)$ where each $S_i \subseteq \mathbb{R}^m$. We will now make this problem precise. First we need to fix what is meant by query:

Definition 1 *A* ***query*** *over the sets* $S_1, \ldots, S_n$ *is a (first order) sentence using the following symbols:*

Constants: $\emptyset, \Omega, S_1, \ldots, S_n$

Function Symbols: $\cap, \cup, '$

Predicate Symbols: $=, \neq, \subseteq, \subset, \supseteq, \supset$

Logical Symbols: $\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$

where Ω *is the whole universe* $\mathbb{R}^m$ *and* $'$ *is the set complement operation in* $\mathbb{R}^m$. ■

Now the problem can be stated as follows:

GIVEN:	$S_1, \ldots, S_n$: sets in $\mathbb{R}^m$ $\phi_1, \ldots, \phi_k$: queries over $S_1, \ldots, S_n$
FIND:	$T_1, \ldots, T_k$, s.t. T_i is *true* if ϕ_i is *true* for $S_1, \ldots, S_n$, *false* otherwise.

Note that we allow more than one query. The reason is that, as it will be made clear later on, some computations can be shared among several queries.

This problem can be easily reduced to deciding sentences in the first order theory of real closed fields which has been shown to be decidable by Tarski around 1930 [9]. Practically feasible algorithms were given by Collins [1] and improved by Hong [3]. However these algorithms still have very high complexity which often makes them practically unusable. Morover in many practical applications one does not need an exact solution to the problem but only a solution which is correct up to a certion user-specified precision. Therefore we will try to gain efficiency by relaxing the problem in the following way:

GIVEN:	$S_1, \ldots, S_n$: sets in $\mathbb{R}^m$ $\phi_1, \ldots, \phi_k$: queries over $S_1, \ldots, S_n$ ε s.t. $\varepsilon \in \mathbb{Q}^+$
FIND:	$T_1, \ldots, T_k$, s.t. there is an $E \subset \mathbb{R}^m$ with $\mathrm{Vol}(E) \leq \epsilon$ s.t. T_i is *true* if ϕ_i is true for $S_1 \setminus E, \ldots, S_n \setminus E$, *false* otherwise.

3 Main Algorithm

In this section we will give a basic straight-forward algorithm. The main idea is to first extract from $S_1, \ldots, S_n$ exactly the information needed to compute the truth of arbitrary queries. Then the truth of the given queries can be decided from this.

What information about the sets is needed for answering arbitrary queries? The following examples provide the intuition. Consider the query $S_1 \subseteq S_2$. This query is true iff the set $S_1 \cap S_2'$ is empty. Also consider the query $S_1 \cap S_2 = \Omega$. This query is true iff all the sets $S_1 \cap S_2'$, $S_1' \cap S_2$ and $S_1' \cap S_2'$ are empty. So we guess that, if we know the (non)-emptyness of every set $P_1 \cap \ldots \cap P_n$, where P_i is either S_i or S_i', we can answer any query.

Exactly this information is depicted by Venn- diagrams ([10]). Every region of a Venn-diagram represents a unique set $P_1 \cap \ldots \cap P_n$ where P_i is either S_i or S_i'. We will mark every non- empty region with a dot. For example, in figure 1 the Venn- diagram of two disjoint sets which fill the whole space is shown.

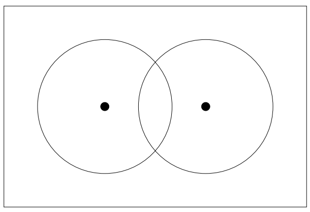

Figure 1: Venn- diagram

Theorem 1 *The truth of any query over sets $S_1, \ldots, S_n$ can be decided from the Venn-diagram of these sets.*

In order to be able to prove this, we will define some notions in a formal way:

Definition 2 *A* ***Venn- diagram*** *of a tuple* $S = (S_1, \ldots, S_n)$ *of sets is the set of non-empty regions of* S*, where a* **region** *is a formula* $P_1 \cap \ldots \cap P_n$*, where* P_i *is either* S_i *or* S_i'*. A region is* **contained** *in a set* S_i *iff it is of the form* $P_1 \cap \ldots \cap S_i \cap \ldots \cap P_n$*.*

Theorem 1 can be proven by providing an algorithm that decides the truth of any query from the Venn- diagram of S. Figure 2 shows such an algorithm. The idea is that the regions of the Venn- diagram of S are representatives of (possibly) infinite sets and checking a query can be done by checking it just for these representatives. More formally this means that queries can be evaluated in the factor structure wrt. the regions, where a symbol S_i in the query stands for the regions of the Venn- diagram which are contained in S_i. The algorithm makes use of the fact that Venn- diagrams are defined as sets, which means that the operations $\cap$, $\cup$, $'$ and predicates like $\subseteq$ can be used on them.

For example we might want to decide the truth of the query $S_1 = \emptyset$ for the Venn-diagram depicted in figure 1. Then the term S_1 evaluates to the Venn- diagram which is just marked with the left point and $\emptyset$ evalutes to the empty diagram. So the query is false since these two are different.

From now on we will denote the Venn- diagram of a tuple of sets S by ν^S.

Theorem 2 *The algorithm of figure 2 is correct.*

Proof. For notational brevity we will denote the set of all Venn- diagrams of n sets by $\mathcal{V}_n$, the set of all regions of n sets by $\mathcal{R}_n$.

Let $eval$ be a function s.t. for any query ϕ over a tuple of sets $S = (S_1, \ldots, S_n)$, $eval_S(\phi)$ is the truth value of ϕ for S and for any term ψ in the query language $eval_S(\psi)$ is the resulting set in $\mathbb{R}^m$. Let $VennReps_S(X)$ be the set of all regions R for which $eval_S(R) \cap X$ is non-empty. Let the functions $EvalQuery$ and $EvalTerm$ denote the corresponding functions in figure 2.

Now we have to prove: For all sentences ϕ of the query language over a tuple of sets $S = (S_1, \ldots, S_n)$:

$$EvalQuery(\phi, \nu^S) \text{ iff } eval_S(\phi).$$

- Claim 1: For any term ψ in the query language over a tuple of sets $S = (S_1, \ldots, S_n)$ we have: $EvalTerm(\psi, \nu^S) = VennReps_S(eval_S(\psi))$

  - Induction Base:

    $\psi = \emptyset$**:** $EvalTerm(\emptyset, \nu^S) = \emptyset$, $VennReps_S(eval_S(\emptyset)) = VennReps_S(\emptyset) = \emptyset$.

    $\psi = \Omega$**:** $EvalTerm(\Omega, \nu^S) = \nu^S$,
    $VennReps_S(eval_S(\Omega)) = VennReps_S(\mathbb{R}^m) = \{R \in \mathcal{R}_n | eval_S(R) \neq \emptyset\} = \nu^S$.

    $\psi = S_i$**:** $EvalTerm(S_i, \nu^S)$ is the set of regions of ν^S of the form $P_1 \cap \ldots \cap S_i \cap \ldots \cap P_n$.
    $VennReps_S(eval_S(S_i))) = VennReps_S(S_i) = \{R \in \mathcal{R}_n | eval_S(R) \cap S_i \neq \emptyset\}$

  - Induction Step:
    $EvalTerm(\psi_1 \cap \psi_2, \nu^S) = EvalTerm(\psi_1, \nu^S) \cap EvalTerm(\psi_2, \nu^S)$.
    By the induction hypothesis this is:
    $eval_S(\psi_1) \cap eval_S(\psi_2) = eval_S(\psi_1 \cap \psi_2)$.
    Similar for all other function symbols.

- Claim 2: For $X = eval_S(\psi)$ we have: $\bigcup_{R \in VennReps_S(X)} eval_S(R) = X$.

  $\supseteq$**:** $\bigcup_{R \in \mathcal{R}_n} eval_S(R) = \mathbb{R}^m$ and $VennReps_S(X)$ contains all regions which have elements of X.

$$T \leftarrow \text{EvalQuery}(\phi,\, V)$$

Input: ϕ: a query
V: the Venn- diagram of a tuple of sets S.

Output: T: *true* iff ϕ is true for S.

(1) **if** ϕ **is** $\psi_1 = \psi_2$ **then** $T \leftarrow \text{EvalTerm}(\psi_1,\, V) = \text{EvalTerm}(\psi_2,\, V)$
(2) **if** ϕ **is** $\psi_1 \subseteq \psi_2$ **then** $T \leftarrow \text{EvalTerm}(\psi_1,\, V\ ) \subseteq \text{EvalTerm}(\psi_2,\, V)$
$\dots$
(3) **if** ϕ **is** $\neg\phi_1$ **then** $T \leftarrow$ **not** $\text{EvalQuery}(\phi_1,\, V)$
(4) **if** ϕ **is** $\phi_1 \wedge \phi_2$ **then** $T \leftarrow \text{EvalQuery}(\phi_1,\, V)$ **and** $\text{EvalQuery}(\phi_2,\, V)$
(5) **if** ϕ **is** $\phi_1 \vee \phi_2$ **then** $T \leftarrow \text{EvalQuery}(\phi_1,\, V)$ **or** $\text{EvalQuery}(\phi_2,\, V)$

$$M \leftarrow \text{EvalTerm}(\psi,\, V)$$

Input: ψ: a term in the query language
V: a Venn diagram of tuple of sets $S = (S_1, \dots, S_n)$

Output: M: ψ evaluated in the universe of the regions of V

(1) **if** ψ **is** $\emptyset$ **then** $M \leftarrow \emptyset$
(2) **if** ψ **is** Ω **then** $M \leftarrow V$
(3) **if** ψ **is** S_i **then** $M \leftarrow$ the set of regions of V that are contained in S_i
(4) **if** ψ **is** ψ_1' **then** $M \leftarrow V\setminus \text{EvalTerm}(\psi_1,\, V)$
(5) **if** ψ **is** $\psi_1 \cap \psi_2$ **then** $M \leftarrow \text{EvalTerm}(\psi_1,\, V)\cap \text{EvalTerm}(\psi_2,\, V)$
(6) **if** ψ **is** $\psi_1 \cup \psi_2$ **then** $M \leftarrow \text{EvalTerm}(\psi_1,\, V) \cup \text{EvalTerm}(\psi_2,\, V)$

Figure 2: Deciding a query from a Venn- diagram

$\subseteq$: $eval_S(R) \cap eval_S(\psi) \neq \emptyset \Rightarrow eval_S(R) \subseteq eval_S(\psi)$ by assuming a $\overline{x}$ s.t. $\overline{x} \in eval_S(R)$ and $\overline{x} \in eval_S(\psi)$. Then a $x \in eval_S(R)$ is also in $eval_S(\psi)$ because x is in- and outside of exactly the same sets as $\overline{x}$ by definition of region.

- Claim 3: For every predicate symbol $\sim$ of the query language, for all terms ψ_1, ψ_2 in the query language over a tuple of sets $S = (S_1, \ldots, S_n)$ we have:

  $VennReps(eval_S(\psi_1)) \sim VennReps(eval_S(\psi_2))$ iff $eval_S(\psi_1) \sim eval_S(\psi_2)$.

  This is true because by claim 2 both $eval_S(\psi_1)$ and $eval_S(\psi_2)$ are unions of evaluated regions.

Now the main proof can be presented. It will be done by induction on the structure of the formula ϕ.

- Induction Base: ϕ is atomic. For every predicate symbol $\sim$ of the query language, for all queries $\psi_1 \sim \psi_2$ we have:

  $EvalQuery(\psi_1 \sim \psi_2, \nu^S)$ iff $EvalTerm(\psi_1, \nu^S) \sim EvalTerm(\psi_2, \nu^S)$ iff (claim 1)

  $VennReps(eval_S(\psi_1)) \sim VennReps(eval_S(\psi_2))$ iff (claim 3)

  $eval_S(\psi_1) \sim eval_S(\psi_2)$ iff $eval_S(\psi_1 \sim \psi_2)$

- Induction Steps:

  $\phi_1 \wedge \phi_2$: $EvalQuery(\phi_1 \wedge \phi_2, \nu^S)$ iff $EvalQuery(\phi_1, \nu^S)$ and $EvalQuery(\phi_2, \nu^S)$ iff (induction hypothesis) $eval_S(\phi_1)$ and $eval_S(\phi_2)$ iff $eval_S(\phi_1 \wedge \phi_2)$.

  Similar for $\wedge, \neg, \Rightarrow, \Leftrightarrow$. ∎

Now we know that any two tuples of sets make the same queries true iff they have the same Venn- diagram. Exactly the elements of such an equivalence class wrt. truth of queries, have the same Venn- diagram. So we can take this Venn- diagram as the representative for the equivalence class and we can say that a query ϕ is true for a Venn-diagram V iff the algorithm of figure 2 gives true, in which case ϕ is true for all tuples of sets S which have the Venn- diagram V.

This immediately leads us to the main algorithm given in figure 3.

$T \leftarrow$ IsTrue(S, ϕ)

Input: S: a list of sets in $\mathbb{R}^m$, where $|S| = n$
ϕ: a list of queries

Output: T: a list of bools s.t. T_i is *true* iff ϕ_i is true for S_i and *false* otherwise.

Local: V: a Venn diagram

(1) [Compute Venn- diagram.]
$V \leftarrow \nu^S$

(2) [Check Satisfyibility.]
for $i \leftarrow 1, \ldots, |\phi|$ **do**
$T_i \leftarrow$ EvalQuery(ϕ_i, V)

Figure 3: Main Algorithm

4 Computing Venn- diagrams

It remains to be investigated how the Venn- diagram ν^S can be computed. We will try to reduce this problem to the problem of finding the Venn- diagram of just one set. Observe that there are exactly three different such Venn- diagrams (see figure 4).

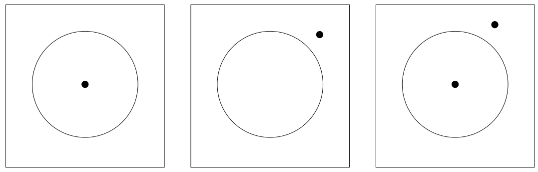

Figure 4: Basic Venn- diagrams

Before showing how to reconstruct the Venn- diagram of more then one sets from these basic diagrams, we will first describe the other direction (getting the basic diagrams from the diagrams of more sets). This can be done by defining a projection operator on Venn- diagrams.

Definition 3 *For a Venn- diagram V of n sets,* $\mathbf{Projection_{p_1,\ldots,p_k}(V)}$ *is the Venn- diagram of k sets consisting of exactly the regions $P_{p_1} \cap \ldots \cap P_{p_k}$ which can be obtained from regions in V by dropping the terms P_i for which there is no j s.t. $i = p_j$.*

Graphically speaking the projection operator removes sets from a Venn- diagram by removing the borders between regions s.t. a region in the new Venn- diagram is non- empty iff one of the merged regions was non- empty in the old diagram (see the example in figure 5).

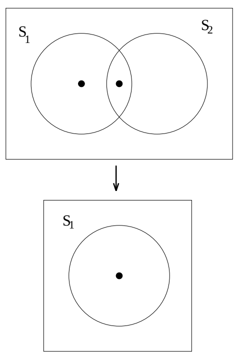

Figure 5: Projection

The following theorem tells us that projection is compatible with the construction of Venn- diagrams.

Theorem 3

$$Projection_{p_1,\ldots,p_k}(\nu^{(S_1,\ldots,S_n)}) = \nu^{(S_{p_1},\ldots,S_{p_k})}$$

This theorem implies that the Venn- diagram of sets $(S_1, \ldots, S_n)$ is in the collection of Venn- diagrams V for which $Projection_i(V) = \nu^{S_i}$. If this collection has exactly one element then we immediately can obtain $\nu^{(S_1,\ldots,S_n)}$ from the ν^{S_i}.

Now we will characterize exactly this situation. To this end we will call the first diagram in figure 4 *all*, the second one *none* and the last one *some*.

Theorem 4 *Iff for all except possibly one* $1 \leq i \leq n$, ν^{S_i} *is* all *or* none *then there is exactly one* V *s.t. for all* $1 \leq i \leq n$, $Projection_i(V) = \nu^{S_i}$.

This means that iff for all except possibly one $1 \leq i \leq n$, ν^{S_i} is *all* or *none* then we can compute the overall Venn- diagram by simply inverting the projection operator. So we have to get rid of all except possibly one *some* diagram for the individual sets. This can be done by observing that for a set S in $\mathbb{R}^m$ which has the *some* diagram we can restrict our viewpiont to only parts of $\mathbb{R}^m$ where the Venn- diagram of S will be *all* or *none*. There will still be parts where the diagram is *some*, but as soon as we know that both *all* and *none* parts are occuring we know that the Venn- diagram in $\mathbb{R}^m$ is *some*. We will denote the Venn- diagram in a subuniverse B of $\mathbb{R}^m$ by ν_B^S and we have:

Theorem 5 $\nu^S = \text{some}$ *iff there are* $B_1, B_2 \subset \mathbb{R}^m$ *s.t.* $\nu_{B_1}^S = \text{all}$ *and* $\nu_{B_2}^S = \text{none}$.

This suggests a divide, conquer and combine strategy for computing the Venn- diagrams. Our first, simple algorithm will recursively divide $\mathbb{R}^m$ into subboxes until the Venn- diagram of $S_1, \ldots, S_n$ in a subbox $B \in \mathbb{IR}^m$ is either *all* or *none*.

If a region is in the Venn- diagram of sets in a box then it is also in the Venn- diagram of the sets in any super- box. So the combination step can then be done by using the following:

Theorem 6 *For any tuple* B' *s.t. all* $B_i' \subset B$ *and* $B_1' \cup \ldots \cup B_n' = B$ *we have:*

$$\nu_B^S = \nu_{B_1'}^S \cup \ldots \cup \nu_{B_n'}^S$$

Now we can give a (non- terminating) sketch of an algorithm for computing the Venn- diagram of given sets within a box (figure 6). It ignores the possibility that one basic Venn diagram could be *some* and uses the symbol *don't know* instead which means that we do not know anything about the Venn- diagram within a certain box. Specifications of subalgorithms for analyzation and division into subboxes are also given. In $\mathbb{R}^m$ these sub- algorithms could for example be done by range overestimation and bisection respectively.

Termination can be ensured by the following possibilities:

- Calling a more sophisticated algorithm if the boxes get too small (e.g. Hong's QEP- CAD program [3]).
- Ignoring small boxes s.t. the overall volume of the ignored boxes is smaller then the user- provided error bound.

The presented versions do not show these possibilities and are only non- terminating sketches of the full (terminating algorithms).

5 Improvements

5.1 Straighforward

The algorithm of figure 6 can be improved because if the full Venn- diagram Ω is reached in the combination step then nothing can change any more and the algorithm can return. This is done by merging the conquer and combination steps (see figure 7).

By accumulating (with the $\cup$ operation) the Venn- diagram information through the whole tree instead of computing it in each recursion branch separately, Ω can be reached earlier (see figure 8). Here the empty Venn- diagram $\emptyset$ has to be passed to the third parameter in the initial call.

Furthermore we do not need to check whether there is a single Venn- diagram fulfilling the properties in the recursion base. Instead it suffices to check whether the result of the $\cup$

$R \leftarrow \text{Venn}(S,\ B)$

Input: S: a list of sets in $\mathbb{R}^m$, where $|S| = n$
B: a box in $\mathbb{IR}^m$

Output: R: the Venn diagram of S in B (ν_B^S).

Local: P: a list of elements of $\{none, all, don't\ know\}$
B': a list of boxes in $\mathbb{IR}^m$
V: a list of Venn- diagrams

(1) [Recursion base.]
for $i \leftarrow 1, \ldots, |S|$ **do**
  $P_i \leftarrow \text{Analyze}(S_i,\ B)$
if $\neg(\exists i) P_i = \textit{don't know}$ **then**
  $R \leftarrow \text{Projection}^{-1}(P)$
  return
(2) [Divide.]
$B' \leftarrow \text{Divide}(B)$
(3) [Recursively Conquer.]
for $i \leftarrow 1, \ldots, |B'|$ **do**
  $V_i \leftarrow \text{Venn}(S,\ B'_i)$
(4) [Combine.]
$R \leftarrow \bigcup V$

$P \leftarrow \text{Analyze}(S,\ B)$

Input: S: a set in $\mathbb{R}^m$
B: a box in $\mathbb{IR}^m$

Output: P: an element of $\{none, all, don't\ know\}$ s.t.
$P = \textit{none}$ implies $S_i \cap B = \emptyset$
$P = \textit{all}$ implies $B \subseteq S_i$

$B' \leftarrow \text{Divide}(B)$

Input: B: a box in $\mathbb{IR}^m$

Output: B': a list of boxes in $\mathbb{IR}^m$ s.t. all $B'_i \subset B$ and $B'_1 \cup \ldots \cup B'_n = B$

Figure 6: First Algorithm

$R \leftarrow \mathrm{Venn}(S,\ B)$

Input: S: a list of sets in $\mathbb{R}^m$, where $|S| = n$
B: a box in $\mathbb{IR}^m$

Output: R: ν_B^S

Local: P: a list of elements of $\{none, all, don't\ know\}$
B': a list of boxes in $\mathbb{IR}^m$

(1) [Recursion base.]
for $i \leftarrow 1, \ldots, |S|$ **do**
$P_i \leftarrow \mathrm{Analyze}(S_i,\ B)$
if $\neg(\exists i) P_i =$ *don't know* **then**
$R \leftarrow \mathrm{Projection}^{-1}(P)$
return
(2) [Divide.]
$B' \leftarrow \mathrm{Divide}(B)$
(3) [Recursively Conquer and Combine.]
$R \leftarrow \emptyset$
while $i \leq |B'|$ **and** $R' \neq \Omega$ **do**
$R \leftarrow R \cup \mathrm{Venn}(S,\ B'_i)$
$i \leftarrow i + 1$

Figure 7: Venn

operation of the current Venn- diagram information with all Venn- diagrams fulfilling the properties is unique.

One further improvement can be achieved by reusing the results of the analyzation algorithm in further recursive calls. If the result for a box is *none* then the result for all subboxes will also be *none* and the same for *all*. Therefore we introduce an additional parameter in which the result of the analyzation algorithm is passed. In the initial call it should contain *don't know* for every element. The resulting algorithm can be seen in figure 9.

The uniqueness check could already succeed before all sets have been analyzed. Therefore it may help to put it into the analyzation loop (see figure 10).

The division cross- section is a box of smaller dimension than the full boxes. This means that it can be done fast compared to a full check. So one further improvement can be introduced by calling the algorithm recursively on this cross- section before the full subboxes (11).

5.2 Tightening

One further improvement can be obtained by using a more sophisticated division algorithm. Such a division algorithm would take care of the base case itself by showing that the Venn- diagram has a certain value on certain sub- boxes.

Such a division algorithm can be implemented by using the notion of tightening, introduced by Hong and Stahl ([5]). Its original purpose was to find as small as possible intervals enclosing all zeros of a polynomial within a box. But here exactly the complement is of interest because the sign of the polynomials does not change there. This means that the Venn diagram is *all* or *none* (depending on the sign of the polynomials) there.

$$R' \leftarrow \text{Venn}(S,\ B,\ R)$$

Input: S: a list of sets in $\mathbb{R}^m$, where $|S| = n$
B: a box in $\mathbb{IR}^m$
R: a Venn- diagram

Output: R': $R \cup \nu_B^S$

Local: P: a list of elements of $\{none, all, don't\ know\}$
B': a list of sets over the given universe

(1) [Recursion base.]
for $i \leftarrow 1, \ldots, |S|$ **do**
  $P_i \leftarrow \text{Analyze}(S_i,\ B)$
if $\neg(\exists i) P_i = don't\ know$ **then**
  $R \leftarrow \text{Projection}^{-1}(P)$
  return
(2) [Divide.]
$B' \leftarrow \text{Divide}(B)$
(3) [Recursively Conquer and Combine.]
$R' \leftarrow R$
$i \leftarrow 1$
while $i \leq |B'|$ **and** $R' \neq \Omega$ **do**
  $R' \leftarrow \text{Venn}(S,\ B'_i,\ R')$
  $i \leftarrow i + 1$

Figure 8: Venn

$R' \leftarrow \text{Venn}(S,\ B,\ R,\ P)$

Input: S: a list of sets in $\mathbb{R}^m$, where $|S| = n$
B: a box in $\mathbb{IR}^m$
R: a Venn- diagram
P: a list of subsets of $\{none, all, don't\ know\}$

Output: R': $R \cup \nu_B^S$

Local: B': a list of sets over the given universe

(1) [Recursion base.]
    for $i \leftarrow 1, \ldots, |S|$ **do**
        if $P_i \neq none$ **and** $P_i \neq all$ **then**
            $P_i \leftarrow$ Analyze(S, B)
    if $R\cup$ Projection$^{-1}(P')$ is unique for all substitutions P' of
            all or *none* for *don't know* in P **then**
        $R' \leftarrow R\cup$ Projection $^{-1}(P')$
        return
(2) [Divide.]
    $B' \leftarrow$ Divide(B)
(3) [Recursively Conquer and Combine.]
    $R' \leftarrow R$
    $i \leftarrow 1$
    while $i \leq |B'|$ **and** $R' \neq \Omega$ **do**
        $R' \leftarrow$ Venn(S, B'_i, R', P)
        $i \leftarrow i + 1$

Figure 9: Venn

$R' \leftarrow \mathrm{Venn}(S,\ B,\ R,\ P)$

Input: S: a list of sets in $\mathbb{R}^m$, where $|S| = n$
B: a box in $\mathbb{IR}^m$
R: a Venn- diagram
P: a list of subsets of $\{none, all, don't\ know\}$

Output: R': $R \cup \nu_B^S$

Local: B': a list of sets over the given universe

(1) [Recursion base.]
    for $i \leftarrow 1, \ldots, |S|$ **do**
        if $P_i \neq none$ **and** $P_i \neq all$ **then**
            $P_i \leftarrow$ Analyze$(S,\ B)$
            if $R\cup$ Projection$^{-1}(P')$ is unique for all substitutions P' of P **then**
                $R' \leftarrow R\cup$ Projection $^{-1}(P')$
            return
(2) [Divide.]
    $B' \leftarrow$ Divide(B)
(3) [Recursively Conquer and Combine.]
    $R' \leftarrow R$
    $i \leftarrow 1$
    while $i \leq |B'|$ **and** $R' \neq \Omega$ **do**
        $R' \leftarrow$ Venn$(S,\ B'_i,\ R',\ P)$
        $i \leftarrow i + 1$

Figure 10: Venn

$$R' \leftarrow \text{Venn}(S,\ B,\ R,\ P)$$

Input: S: a list of sets in $\mathbb{R}^m$, where $|S| = n$
B: a box in $\mathbb{IR}^m$
R: a Venn- diagram
P: a list of subsets of $\{none, all, don't\ know\}$

Output: R': $R \cup \nu_B^S$

Local: B': a list of sets over the given universe

(1) [Recursion base.]
  for $i \leftarrow 1, \ldots, |S|$ **do**
    if $P_i \neq none$ **and** $P_i \neq all$ **then**
      $P_i \leftarrow$ Analyze$(S,\ B)$
      if $R\cup$ Projection$^{-1}(P')$ is unique for all substitutions P' of P **then**
        $R' \leftarrow R\cup$ Projection $^{-1}(P')$
       return
(3) [Cross section inspection.]
  $R' \leftarrow$ Venn$(S,$ CrossSection$(B),\ R,\ P)$
(2) [Divide.]
  $B' \leftarrow$ Divide(B)
(3) [Recursively Conquer and Combine.]
  $i \leftarrow 1$
  while $i \leq |B'|$ **and** $R' \neq \Omega$ **do**
    $R' \leftarrow$ Venn$(S,\ B'_i,\ R',\ P)$
    $i \leftarrow i + 1$

Figure 11: Venn

From now on we assume that the sets are defined by only one polynomial. The method can be easily generalized to sets defined by more polynomials.

So when tightening is applied to every polynomial defining a set then we get intervals enclosing all the zeros of each polynomial as the result. But these intervals are different for each polynomial and they have to be merged into one result (see figure 12).

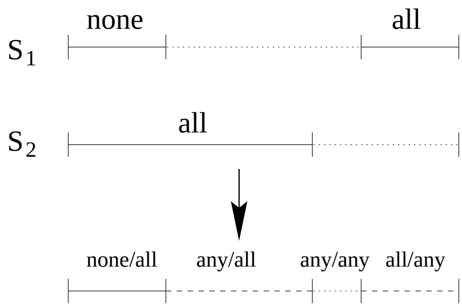

Figure 12: Merging Tightening Result

Special care has to be taken because the result of tightening consists of closed intervals while their complements are open intervals. This has to be preserved when merging the results.

6 Quantifiers, Cardinality

In this chapter we will investigate the extension of the query language by quantifiers or a cardinality operator. We will show that both extensions yield languages of the same expressiveness. Some of these results can be inferred from results on quantifier elimination in measure algebras (i.e. generalized boolean algebras with a finitely additive measure assuming values in an ordered field), see for example [11]. However our presentation is based on different point of view.

A query using quantifiers might look like this: $(\exists X)[X \subset S_1]$ meaning that S_1 has a proper subset. An equivalent formula using a cardinality operator might look like this: $|S_1| \geq 1$.

Definition 4 *Let $\mathcal{L}_Q$ be the extension of the query language by quantifiers ranging over all subsets of $\mathbb{R}^m$. Let $\mathcal{L}_C$ be the extension of the query language by atomic formulas of the form $|T| \leq c$, $|T| \geq c$ and $|T| = c$, where T is an arbitrary term of the query language and c is a non- negative integer.*

These language extensions have their natural semantics. We will call queries without the above extensions basic queries. Now we will introduce some definitions and theorems which will enable us to deal with extended queries in a similar way as with basic queries.

Definition 5 *A **generalized Venn- diagram** of a tuple $S = (S_1, \ldots, S_n)$ of sets is the tuple of region cardinalities of all regions where a **region cardinality** of a region R is a formula $|R| = c$, where c is the cardinality of the set denoted by the region (either a non-negative integer or ∞).*

We will depict generalized Venn- diagrams not by marking regions with dots but by marking each region with the constant on the right side of its region cardinality (see 13).

Theorem 7 *The truth of any query in $\mathcal{L}_C$ over sets $S_1, \ldots, S_n$ can be decided from the generalized Venn- diagram of these sets.*

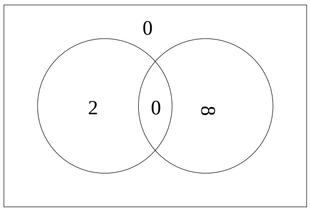

Figure 13: Generalized Venn- diagram

$T \leftarrow$ EvalQuery(ϕ, V)

Input: ϕ: a query in $\mathcal{L}_C$
V: the generalized Venn- diagram of a tuple of sets S.

Output: T: *true* iff ϕ is true for S.

(1) $V' \leftarrow$ the Venn- diagram of all regions which have non-zero region- cardinality in V.
(2) **if** ϕ **is** $|\psi| = n$ **then** $T \leftarrow$ Card(EvalTerm(ψ, V'), V)=n
(3) **if** ϕ **is** $\psi_1 = \psi_2$ **then** $T \leftarrow$ EvalTerm(ψ_1, V') = EvalTerm(ψ_2, V')
(4) **if** ϕ **is** $\psi_1 \subseteq \psi_2$ **then** $T \leftarrow$ EvalTerm(ψ_1, V') $\subseteq$ EvalTerm(ψ_2, V')
...
(5) **if** ϕ **is** $\neg\phi_1$ **then** $T \leftarrow$ **not** EvalQuery(ϕ_1, V')
(6) **if** ϕ **is** $\phi_1 \wedge \phi_2$ **then** $T \leftarrow$ EvalQuery(ϕ_1, V') **and** EvalQuery(ϕ_2, V')
(7) **if** ϕ **is** $\phi_1 \vee \phi_2$ **then** $T \leftarrow$ EvalQuery(ϕ_1, V') **or** EvalQuery(ϕ_2, V')

$n \leftarrow$ Card(V, C)

Input: V: a Venn- diagram
C: a generalized Venn- diagram

Output: n: the added cardinalities of the regions of V in C (where $\infty + n = n + \infty = \infty$).

(1) $n \leftarrow \sum_{r \in V} card(r, C)$ where $card(r, C)$ is s.t. $|r| = card(r, C) \in C$

Figure 14: Deciding an extended query from a generalized Venn- diagram

The proof is provided by an extension of the algorithm of figure 2, shown in figure 14

Now, similar to basic queries, any two tuples of sets make the same queries in $\mathcal{L}_C$ true iff they have the same generalized Venn- diagrams. Exactly the elements of an equivalence class wrt. truth of queries have the same generalized Venn- diagram. So we can take this generalized Venn- diagrams as the representative of the equivalence classes. Then we can say that a query in $\mathcal{L}_C$ is true for a generalized Venn- diagram V iff the algorithm of figure 14 gives true, in which case the query is true for all tuples of sets that have the generalized Venn- diagram V.

Definition 6 *For a generalized Venn- diagram W of n sets,* $\mathbf{Projection_{p_1,\ldots,p_k}(W)}$ *is the Venn- diagram V of k sets s.t. $|P_{p_1} \cap \ldots \cap P_{p_k}| = c \in V$ iff c is the sum of the cardinalities of all regions in W which are contained in P_{p_i} if $P_{p_i} = S_{p_i}$ and are not contained in P_{p_i} if $P_{p_i} = S'_{p_i}$.*

Furthermore we define $Projection(W)$ as $Projection_{1,\ldots,n-1}(W)$ and for a set of generalized Venn- diagrams $\mathcal{V}$, $Projection(\mathcal{V}) = \{Projection(V) | V \in \mathcal{V}\}$.

From now on we will denote the generalized Venn- diagram of a tuple of sets S by $\overline{\nu}^S$.

Theorem 8

$$Projection_{p_1,\ldots,p_k}(\overline{\nu}^{(S_1,\ldots,S_n)}) = \overline{\nu}^{(S_{p_1},\ldots,S_{p_k})}$$

Proof. Let $|P_1 \cap \ldots \cap P_n| = c$ be a region cardinality in $\overline{\nu}^{(S_{p_1},\ldots,S_{p_k})}$. We have to prove that c is the sum of the cardinalities of all regions in $\overline{\nu}^{(S_1,\ldots,S_n)}$ which are contained in P_{p_i} if $P_{p_i} = S_{p_i}$ and are not contained in P_{p_i} if $P_{p_i} = S'_{p_i}$.

This is clearly true because these regions are the only subsets of $P_{p_1} \cap \ldots \cap P_{p_k}$ and they are disjoint. ■

Theorem 9 *Let $\phi \in \mathcal{L}_C$. Let $S = (S_1, \ldots, S_n)$ be a tuple of sets over $\mathbb{R}^m$. Then $(\exists X)\phi[X]$ is true for S iff there is a V s.t. $Projection(V) = \overline{\nu}^S$ and $\phi[S_{n+1}]$ is true for V. Furthermore $(\forall X)\phi[X]$ is true for S iff for all V s.t. $Projection(V) = \overline{\nu}^S$ and $\phi[S_{n+1}]$ is true for V.*

Proof. We will give the proof only for existential quantification, it is similar for universal quantification.

$\Rightarrow$: Assume that $(\exists X)\phi[X]$ is true for S. Then there is a set S_{n+1} s.t. $\phi[S_{n+1}]$ is true for $(S_1, \ldots, S_{n+1})$. We choose V as $\overline{\nu}^{(S_1,\ldots,S_{n+1})}$. Then $Projection(V) = \overline{\nu}^S$ by theorem 8 and $\phi[S_{n+1}]$ is true for $\overline{\nu}^{(S_1,\ldots,S_n)}$ by theorem 7.

$\Leftarrow$: Assume that there is a V s.t. $\phi[S_{n+1}]$ is true for V and $Projection(V) = \overline{\nu}^S$. Then we have to find a set S_{n+1} s.t. $\phi[S_{n+1}]$ is true for $(S_1, \ldots, S_{n+1})$. Now we choose S_{n+1} s.t. $\overline{\nu}^{(S_1,\ldots,S_{n+1})} = V$. Such a set exists because $Projection(V) = \overline{\nu}^S$ which means that the elements of $S_1, \ldots, S_n$ can be distributed inside and outside of S_{n+1} according to V. Now $\phi[S_{n+1}]$ is true for $(S_1, \ldots, S_{n+1})$ because it is true for $V = \overline{\nu}^{(S_1,\ldots,S_{n+1})}$.

■

Corollary 1 *The truth of any query in $\mathcal{L}_Q$ over sets $S_1, \ldots, S_n$ can be decided from the generalized Venn- diagram of these sets.*

So queries of both language extension can be decided from generalized Venn- diagrams. It remains to be shown that they have the same expressiveness.

Theorem 10 *For any formula in $\mathcal{L}_C$ there is an equivalent formula in $\mathcal{L}_Q$.*

Proof. Any formula of the form $|T| = c$ is equivalent to a formula of the form $|T| \leq c \wedge |T| \geq c$ and any formula of the form $|T| \leq c$ is equivalent to a formula of the form $\neg|T| \geq c+1$. So it suffices to show that there is an equivalent formula in $\mathcal{L}_Q$ for any formula of the form $|T| \geq c$. It can be easily seen that a formula of the form $(\exists X_1)(\exists X_2)\ldots(\exists X_c)\ X_1 \subset X_2\ \wedge\ X_2 \subset X_3\ \wedge\ \ldots\ \wedge\ X_{c-1} \subset X_c\ \wedge\ X_c \subset T$ is equivalent. ■

In order to show the other direction we need the following theorem:

Theorem 11 *Let ϕ be an extended query in $\mathcal{L}_C$. Let $\mathcal{V}$ be the set of exactly the generalized Venn- diagrams for which ϕ is true. Then there is also a ϕ' s.t. ϕ' is true exactly for the elements of $Projection(\mathcal{V})$.*

Proof. Any extended query ϕ in $\mathcal{L}_C$ can be transformed into an equivalent normal form $\overline{\phi}$ s.t. $\overline{\phi}$ is in disjunctive normal form $P_1 \vee \ldots \vee P_k$, has no negations, only atomic queries of the form $|S_1 \cap \ldots \cap S_i| \sim c$ where $\sim \in \{=, \leq, \geq\}$ and in a P_i no left-hand side $|S_1 \cap \ldots \cap S_i|$ occurs twice. Then we get ϕ' by simply dropping all atomic formulas containing S_n from $\overline{\phi}$. ■

Theorem 12 *For any query in $\mathcal{L}_Q$ there is an equivalent query in $\mathcal{L}_C$.*

Proof. Without loss of generality we proof this for queries of the form $(Q_1X_1)(Q_2X_2)\ldots(Q_nX_n)\phi$ where Q_i is either $\exists$ or $\forall$ and ϕ is a basic query. Then ϕ is obviously in $\mathcal{L}_C$. Now it suffices that, given a query $(QX)\phi$ where $\phi \in \mathcal{L}_C$, $Q \in \{\exists, \forall\}$ there is an equivalent formula $\phi' \in \mathcal{L}_C$.

We will present the proof for $Q = \exists$ only. The $Q = \forall$ part is similar.

Let $\mathcal{V}$ be the set of all generalized Venn- diagrams which are true for $\phi[S_{n+1}]$. Then by theorem 9, $Projection(\mathcal{V})$ consists of exactly the generalized Venn- diagrams for which $(\exists X)\phi[X]$ is true. Now by theorem 11 there is a $\phi' \in \mathcal{L}_C$ which is true for exactly the elements of $Projection(\mathcal{V})$. This $\phi' \in \mathcal{L}_C$ is equivalent to $(\exists X)\phi$. ■

Corollary 2 *The languages $\mathcal{L}_Q$ and $\mathcal{L}_C$ have the same expressiveness.*

7 Conclusion

We have shown how queries about semi- algebraic sets can be efficiently answered. There is already a prototype implementation of the above algorithms. Furthermore they are already used in a working CLP system (see [4]).

Future work involves further efficiency improvements, specialication for consistency and redundancy checks, incrementality and the use of transcendental functions.

References

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

[2] A. Colmerauer. An introduction to prolog III. *Communications of the ACM*, 33(7):70–90, 1990.

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

[4] H. Hong and S. Ratschan. RISC-CLP(Tree(Δ)): A constraint logic programming system with parametric domain. Technical Report 25, Research Institute for Symbolic Computation, Johannes Kepler University, A-4040 Linz, Austria, 1995.

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

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

[7] J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[8] K. Sakai and A. Aiba. Cal: A theoretical background of constraint logic programming and its applications. *Journal of Symbolic Computation*, 8:589–603, 1989.

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

[10] J. Venn. On the diagrammatic and mechanical representation of propositions and reasonings. *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science*, 5(9):1–18, 1880.

[11] V. Weispfenning. Quantifier elimination for distributive lattices and measure algebras. *Zeitschrift für math. Logik und Grundlagen der Mathematik*, 31:249–261, 1985.