Quantifier-Elimination for the First-Order Theory of Boolean Algebras with Linear Cardinality Constraints*

Peter Revesz

Department of Computer Science and Engineering
University of Nebraska-Lincoln, Lincoln, NE 68588, USA
{revesz}@cse.unl.edu

Abstract. We present for the first-order theory of atomic Boolean algebras of sets with linear cardinality constraints a quantifier elimination algorithm. In the case of atomic Boolean algebras of sets, this is a new generalization of Boole's well-known variable elimination method for conjunctions of Boolean equality constraints. We also explain the connection of this new logical result with the evaluation of relational calculus queries on constraint databases that contain Boolean linear cardinality constraints.

1 Introduction

Constraint relations, usually in the form of semi-algebraic or semi-linear sets, provide a natural description of data in many problems. What programming language could be designed to incorporate such constraint relations? Jaffar and Lassez [27] proposed in a landmark paper *constraint logic programming*, with the idea of extending Prolog, and its usual top-down evaluation style, with constraint solving (which replaces unification in Prolog). That is, in each rule application after the substitution of subgoals by constraint tuples, the evaluation needs to test the satisfiability of the constraints, and proceed forward or backtrack according to the result.

As an alternative way of incorporating constraint relations, in a database framework, Kanellakis, Kuper, and Revesz [29, 30] proposed *constraint query languages* as an extension of relational calculus with the basic insight that the evaluation of relational calculus queries on X-type constraint relations reduces to a quantifier elimination [15, 16] in the first-order theory of X.[1] As an example from [29, 30], if the constraint relations are semi-algebraic relations, then the

* This work was supported in part by USA National Science Foundation grant EIA-0091530.

[1] They also considered various bottom-up methods of evaluating Datalog queries and the computational complexity and expressive power of constraint queries. Since relational calculus is closely tied with practical query languages like SQL, it has captured the most attention in the database area.

quantifier elimination for real closed fields [2, 3, 12, 13, 40, 54] can be used to evaluate queries.

There are advantages and disadvantages of both styles of program/query evaluations. Constraint logic programs have the advantage that their implementation can be based on only constraint satisfiability testing, which is usually easier and faster than quantifier elimination required by constraint relational calculus. On the other hand, the termination of constraint logic programs is not guaranteed, except in cases with a limited expressive power. For example, for negation-free Datalog queries with integer (gap)-order constraints the termination of both the tuple-recognition problem [14] and the least fixed point query evaluation [41, 42] can be guaranteed. When either negation or addition constraints are also allowed, then termination cannot be guaranteed. In contrast, the evaluation of constraint relational calculus queries have a guaranteed termination, provided there is a suitable effective quantifier elimination method.

While many other comparisons can be made (see the surveys [28, 43] and the books [31, 33, 47]), these seem to be the most important. Their importance becomes clear when we consider the expected users. Professional programmers can write any software in any programming language and everything could be neatly hidden (usually under some kind of options menu) from the users. In contrast, database systems provide for the users *not ready-made programs but a easy-to-use high-level programming language*, in which they can write their own simple programs. It is unthinkable that this programming language not terminate, and, in fact, run efficiently. Therefore constraint database research focused on the efficient evaluation of simple non-recursive query languages.[2]

The constraint database field made initially a rapid progress by taking off-the-shelf some quantifier elimination methods. Semi-linear sets as constraint relations are allowed in several prototype constraint database systems [8, 21, 24, 25, 48, 49] that use Fourier-Motzkin quantifier elimination for linear inequality constraints [18]. The latest version of the DISCO system [9, 50] implements Boolean equality constraints using Boole's existential quantifier elimination method for conjunctions of Boolean equality constraints.

Relational algebra queries were considered in [20, 42, 46]. As in relational databases, the algebraic operators are essential for the efficient evaluation of queries. In fact, in the above systems logical expressions in the form of relational calculus, SQL, and Datalog rules are translated into relational algebra.

There were also deep and interesting questions about the relative expressive power and computational complexity of relational versus constraint query languages. Some results in this area include [4, 23, 37, 45] and a nice survey of these can be found in Chapters 3 and 4 of [31].

[2] Of course, many database-based products also provide menus to the users. However, the users of database-based products are only indirect users of database systems. The direct users of database systems are application developers, who routinely embed SQL expressions into their programs. Thanks to today's database systems, they need to worry less about termination, efficiency, and many other issues than yesterday's programmers needed while developing software products.

After these initial successes, it became clear that further progress may be possible only by extending the quantifier elimination methods. Hence researchers who happily got their hands dirty doing implementations found themselves back at the mathematical drawing table.

The limitations of quantifier elimination seemed to be most poignant for Boolean algebras. It turns out that for conjunctions of Boolean equality *and inequality* constraints (which seems to require just a slight extension of Boole's method) no quantifier elimination is possible. Let us see an example, phrased as a lemma.

Lemma 1. *There is no quantifier-free formula of Boolean equality and inequality constraints that is equivalent in every Boolean algebra to the following formula:*

$$\exists d\ (d \sqcap g \neq \bot) \wedge (\overline{d} \sqcap g \neq \bot)$$

where d and g are variables and $\bot$ is the zero element of the Boolean algebra. □

Consider the Boolean algebra of sets, with the one element being the names of all persons, the zero element being the empty set, the domain being the powerset (set of all subset of the one element).

In the formula variable d may be the set of students who took a database systems class, variable g may be the set of students who graduate this semester. Then the formula expresses the statement that "some graduating student took a database systems class, and some graduating student did not take a database systems class." This formula implies that g has at least two elements, that is, the cardinality of g is at least two, denoted as:

$$|g| \geq 2$$

But this fact can not be expressed by any quantifier-free formula with Boolean equality and inequality constraints and g as the only variable.

Lemma 1 implies that there is no general quantifier elimination method for formulas of Boolean equality and inequality constraints. This negative result was noted by several researchers, who then advocated approximations. For example, Helm et al. [26] approximate the result by a formula of Boolean equality and inequality constraints. Can we do better than just an approximation?

The only hopeful development in the quantifier elimination area was by Marriott and Odersky [32] who showed that formulas with equality and inequality constraints admit quantifier elimination for the special case of *atomless* Boolean algebras. However, many Boolean algebras are not atomless but atomic. How can we deal with those Boolean algebras? Could any subset of atomic Boolean algebras also admit quantifier elimination? In this paper we show that the atomic Boolean algebras of sets, i.e., Boolean algebras where the Boolean algebra operators are interpreted as the usual set operators of union, intersection and complement with respect to the one element, also admit quantifier elimination, in spite of the pessimistic looking result of Lemma 1.

Let us take a closer look at the Lemma. Surprisingly, the condition $|g| \geq 2$ is not only necessary, but it is also sufficient. That is, for any Boolean algebra of sets if G is any set with at least two elements, then we can find a set D such that the above formula holds. Therefore, $|g| \geq 2$ is exactly the quantifier-free formula that we would like to have as a result of the quantifier elimination. However, quantifier elimination techniques are normally required to give back equivalent quantifier-free formulas with the same type of constraints as the input. This condition is commonly called being *closed* under the set of constraints. This raises the interesting question of what happens if we allow cardinality constraints in our formulas.

While cardinality constraints on sets are considered by many authors, and interesting algorithms are developed for testing the satisfiability of a conjunction of cardinality constraints, there were, to our knowledge, no algorithms given for quantifier elimination for atomic Boolean algebras of sets with cardinality constraints.

Calvanese and Lenzerini [11, 10] study cardinality constraints that occur in ER-diagrams and ISA hierarchies. They give a method to test the satisfiability of a schema. This is a special case of cardinality constraints, because the ER-diagrams do not contain inequality constraints.

Ohlbach and Koehler [35, 36] consider a simple description logic with cardinality constraints. They give methods to test subsumption and satisfiability of their formulas, but they do not consider quantifier elimination.

Seipel and Geske [51] use constraint logic programming to solve conjunctions of cardinality constraints. Their set of constraint logic programming [27] rules is sound but incomplete.

Surprisingly, in this paper, we show that the augmented formulas, called *Boolean linear cardinality constraint formulas*, admit quantifier elimination. It is surprising that by adding to the set of atomic constraints, the problem of quantifier elimination becomes easier, not harder. Indeed, the end result, which is our quantifier elimination method described in this paper, may strike the reader as simple. But the finding of the trick of adding cardinality constraints for the sake of performing quantifier elimination is not obvious as shown by the following history summarized in Figure 1. In the figure the arrows point from less to more expressive Boolean constraint theories, but the labels on them indicate that the Boolean algebra needs to be of a certain type. Let's describe Figure 1 in some detail (please see Section 2 for definitions of unfamiliar terms).

Precedence between variables: A naive elimination of variables from a set of Boolean precedence constraints between variables or constants (in the case of algebras of sets set containment constraints between sets) occurs in syllogistic reasoning. Namely, the syllogistic rule *if all x are y, and all y are z, then all x are z* yields a simple elimination of the y variable. Such syllogisms were described already by Aristotle and developed further in medieval times and can be used as the basis of eliminating variables. Srivastava, Ramakrishnan, and Revesz [52] gave an existential quantifier elimination method for a special subset of the Boolean algebra of sets. They considered existentially quantified formulas with

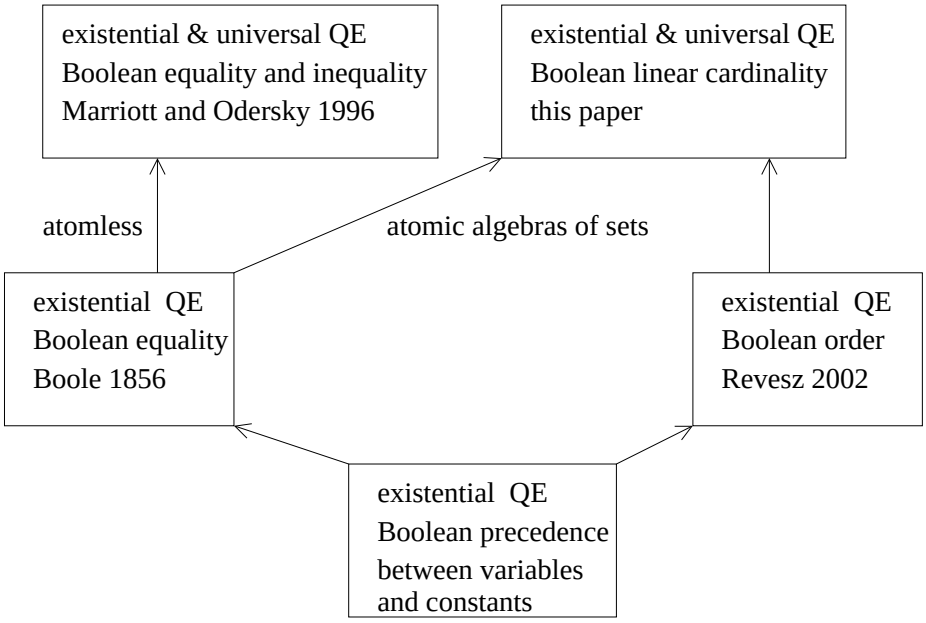

Fig. 1. Quantifier elimination methods for Boolean theories.

the quantifier-free part being only a conjunction of atomic constraints of the form $v \subseteq w$, where v and w are constants or variables ranging over the elements of a Boolean algebra of sets. Gervet [19] independently derived a similar method about the same time. This shows that the problem of quantifier elimination for sets arises naturally in different contexts.

Boolean equality: The quantifier elimination procedure for Boolean formulas with equality constraints was given by George Boole in 1856. His method allows the elimination of only existentially quantified variables.

Boolean order: Revesz [44, 47] shows that Boolean order constraints allow existential quantifier elimination. Boolean order constraints (see Section 2.2) are equality constraints of the form $x \sqcap t'_{mon} = \bot$ and inequality constraints of the form $t_{mon} \neq \bot$ where x is a variable and t is a *monotone Boolean term*, that is, a term that contains only the $\sqcap$ and $\sqcup$ operators. In Boolean algebras of sets, the first of these constraints can be written as $t_{mon} \supseteq x$. This is clearly a generalization of precedence constraints between variables. However, it is incomparable with Boolean equalities. This theory contains some inequality constraints (which cannot be expressed by equality constraints), namely when the left hand side is a monotone term, but it cannot express all kinds of equality constraints, but only those that have the form $t'_{mon} \wedge x = \bot$.

Boolean equality and inequality: Marriott and Odersky [32] show that atomless Boolean algebras admit both existential and universal quantifier elim-

ination. Clearly, their method is a generalization of Boole's method in the case of atomless Boolean algebras.

Boolean linear cardinality: The quantifier elimination for this is introduced in this paper. In the case of atomic Boolean algebras of sets, the new linear cardinality constraints quantifier elimination is another generalization of the quantifier elimination considered by Boole as shown by Lemma 3.

The outline of this paper is as follows. Section 2 gives some basic definitions regarding Boolean algebras, constraints, and theories. Section 3 describes a new quantifier elimination method for conjunctions of Boolean cardinality constraints. Section 4 uses the quantifier elimination method for the evaluation of relational calculus queries. Finally, Section 5 gives some conclusions and directions for further research.

2 Basic Definitions

We define Boolean algebras in Section 2.1, Boolean constraints in Section 2.2, and Boolean theories in Section 2.3.

2.1 Boolean Algebras

Definition 1. *A* Boolean algebra B *is a tuple* $\langle \delta, \sqcap, \sqcup, ', \bot, \top \rangle$*, where* δ *is a non-empty set called the* domain*;* $\sqcap$*,* $\sqcup$ *are binary functions from* $\delta \times \delta$ *to* δ*;* $'$ *is a unary function from* δ *to* δ*; and* $\bot$*,* $\top$ *are two specific elements of* δ *(called the* zero *element and the* one *element, respectively) such that for any elements* x*,* y*, and* z *in* δ *the following axioms hold:*

$$\begin{array}{ll} x \sqcup y = y \sqcup x & x \sqcap y = y \sqcap x \\ x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z) & x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z) \\ x \sqcup x' = \top & x \sqcap x' = \bot \\ x \sqcup \bot = x & x \sqcap \top = x \\ \bot \neq \top & \end{array}$$

For Boolean algebras we define the *precedence* relation, denoted as $\sqsupseteq$, by the following identity:

$$x \sqsupseteq y \quad \text{means} \quad x' \wedge y = \bot$$

We also write the above as $y \sqsubseteq x$ and say that y precedes x.

The above gives a formal definition of a Boolean algebra. It can be considered its syntax. The semantics of a Boolean algebra is given through *interpretations* for the elements of its structure. Without going into deep details about the numerous possible interpretations, we give one common interpretation of the domain and operators.

Definition 2. *A* Boolean algebra of sets *is any Boolean algebra, where:*
δ is a set of sets,
$\sqcup$ is interpreted as set union, denoted as $\cup$,
$\sqcap$ is interpreted as set intersection, denoted as $\cap$,
$'$ is interpreted as set complement with respect to $\top$, denoted as $\bar{}$, and
$\sqsubseteq$ (or $\sqsupseteq$) is interpreted as set containment, denoted as $\subseteq$ (or $\supseteq$).

Note: We call two Boolean algebras *isomorphic* if there is between them a bijection which preserves their respective Boolean operations. By Stone's representation theorem, any Boolean algebra is isomorphic to a Boolean algebra of sets [53]. Hence restricting our attention in this paper to Boolean algebras of sets is without any significant loss of generality.

An *atom* of a Boolean algebra is an element $x \neq \bot$ such that there is no other element $y \neq \bot$ with $y \sqsubseteq x$. It can happen that there are no atoms at all in a Boolean algebra. In that case, we call the Boolean algebra *atomless*; otherwise we call it *atomic*.

Let $\mathcal{N}$ denote the set of non-negative integer, $\mathcal{Z}$ denote the set of integer, and $\mathcal{Q}$ denote the set of rational numbers.

Example 1.

$$B_Z = \langle \mathcal{P}owerset(\mathcal{Z}), \cap, \cup, \bar{}, \emptyset, \mathcal{Z} \rangle$$

is a Boolean algebra of sets. In this algebra for each $i \in \mathcal{Z}$ the singleton set $\{i\}$ is an atom. This algebra is atomic. □

Example 2. Let H be the set of all finite unions of half-open intervals of the form $[a, b)$ over the rational numbers, where $[a, b)$ means all rational numbers that are greater than or equal to a and less than b, where a is a rational number or $-\infty$ and b is a rational number. Then:

$$B_H = \langle H, \cap, \cup, \bar{}, \emptyset, \mathcal{Q} \rangle$$

is another Boolean algebra of sets. This algebra is atomless. □

2.2 Boolean Constraints

In the following we consider only atomic Boolean algebras of sets with either a finite or a countably infinite number of atoms. For example, the Boolean algebra B_Z has a countably infinite number of atoms. We also assume that we can take the union or the intersection of an infinite number of elements of the Boolean algebra, i.e., our Boolean algebras are *complete*.

Cardinality can be defined as follows.

Definition 3. *Let x be any element of an atomic Boolean algebra of sets with a finite or countably infinite number of atoms. If x can be written as the finite union of $n \in \mathcal{N}$ number of distinct atoms, then the* cardinality *of x, denoted $|x|$, is n. Otherwise the cardinality of x is infinite, which is denoted as $+\infty$.*

The following lemma shows that the cardinality is a well-defined function from elements of an atomic Boolean algebra to $N \cup \{+\infty\}$.

Lemma 2. *Let x be any element of an atomic Boolean algebra of sets with a finite or countably infinite number of atoms. Then x can be written as the union of some $n \in \mathcal{N}$ or $+\infty$ number of distinct atoms in the Boolean algebra. Moreover, there is no other set of atoms whose union is equivalent to x.* □

For example, in the Boolean algebra B_Z we can write $\{1, 2\}$ as $\{1\} \cup \{2\}$ or as $\{2\} \cup \{1\}$. In either way, we use the same two distinct atoms, i.e., $\{1\}$ and $\{2\}$. It follows from Lemma 2 that it is enough to allow only atomic constant symbols in *Boolean terms* and *Boolean constraints*, which we define as follows.

Definition 4. Let $B = \langle \delta, \cap, \cup, \bar{}, \bot, \top \rangle$ be an atomic Boolean algebra of sets. A *Boolean term* over B is an expression built from variables ranging over δ, atomic constants denoting particular atoms of B, $\bot$ denoting the zero element, $\top$ denoting the one element, and the operators for intersection, union, and complement.

A Boolean term is *monotone* if it is built without the use of the complement operator.

Definition 5. Boolean constraints have the form:

Equality :	$t = \bot$
Inequality :	$t \neq \bot$
Order :	$t_{mon} \neq \bot$ or $t_{mon} \sqsupseteq x$
Linear Cardinality :	$c_1\|t_1\| + \ldots + c_k\|t_k\| \ \theta \ b$

where t and each t_i for $1 \leq i \leq k$ is a Boolean term, t_{mon} is a monotone Boolean term, x is a Boolean variable, each c_i for $1 \leq i \leq k$ and b is an integer constant and θ is:

$=$ for the equality relation,
$\geq$ for the greater than or equal comparison operator,
$\leq$ for the less than or equal comparison operator, or
$\equiv_n$ for the congruence relation modulus some positive integer constant n.

2.3 Boolean Theories

A *formula* of Boolean algebras of sets is a formula that is built in the usual way from the existential quantifier $\exists$, the universal quantifier $\forall$, the logical connectives $\wedge$ for *and* $\vee$ for *or*, the apostrophe $'$ for *negation*, and one of the above types of Boolean algebra constraints.

A solution or *model* of a Boolean algebra constraint (or formula) is an assignment of the (free) variables by elements of the Boolean algebra such that the constraint (or formula) is satisfied. A constraint (or formula) is *true* if every possible assignment is a solution.

Example 3. Consider the Boolean algebra B_Z of Example 1. Then the Boolean linear cardinality constraint:

$$3|x \cap y| - 2|z| = 4$$

has many solutions. For example $x = \{3, 6, 9\}$ and $y = \{3, 4, 5, 6\}$ and $z = \{1\}$ is a solution. The Boolean linear cardinality constraint:

$$|x \cup \{1\} \cup \{2\}| \geq 2$$

is true, because every assignment to x is a solution. □

Example 4. Suppose that we know the following facts about a company:

1. The number of salespersons is a multiple of 6.
2. The number of engineers is a multiple of 9.
3. There are seven employees who are both salespersons and engineers but not managers.
4. There are twice as many managers who are salespersons than managers who are engineers.
5. Each manager is either a salesperson or an engineer but not both.

Using variables x for managers, y for salespersons, and z for engineers, we can express the above using the following Boolean cardinality formula S:

$$\begin{aligned} |y| &\equiv_6 0 \wedge \\ |z| &\equiv_9 0 \wedge \\ |\overline{x} \cap y \cap z| &= 7 \wedge \\ |x \cap y| - 2|x \cap z| &= 0 \wedge \\ |x \cap y \cap z| + |x \cap \overline{y} \cap \overline{z}| &= 0 \end{aligned}$$

□

Example 5. Suppose that we know the following additional fact about the company in Example 4:

6. Person a is both a manager and an engineer but not a salesperson.

Here a is a constant symbol that denotes a particular atom of the Boolean algebra. We can express this by the following Boolean constraint:

$$|x \cap \overline{y} \cap z \cap a| = 1$$

□

In any given Boolean algebra two formulas are *equivalent* if they have the same set of models. The purpose of *quantifier elimination* is to rewrite a given formula with quantifiers into an equivalent quantifier-free formula [16]. A quantifier elimination method is *closed* or has a *closed-form* if the type of constraints in the formula with quantifiers and the quantifier-free formula are the same. (It

is sometimes possible to eliminate quantifiers at the expense of introducing more powerful constraints, but then the quantifier elimination will not be closed-form.)

In arithmetic theories quantifier elimination is a well-studied problem. A well-known theorem due to Presburger ([38] and improvements in [6, 7, 17, 55, 56]) is that a closed-form quantifier elimination is possible for formulas with linear equations (including congruence equations modulus some fixed set of integers). We will use this powerful theorem in this paper.

The following lemma shows in the case of atomic Boolean algebras of sets a simple translation from formulas with only equality and inequality constraints into formulas with only linear cardinality constraints.

Lemma 3. *In any atomic Boolean algebra of sets, for every term t we have the following:*

$$\begin{array}{lll} t = \bot & \text{if and only if} & |t| \;=\; 0 \\ t \neq \bot & \text{if and only if} & |t| \;\geq\; 1 \end{array}$$

Moreover, using the above identities, any formula with only equality and inequality constraints can be written as a formula with only linear cardinality constraints. □

Finally, we introduce some useful technical definitions related to Boolean theories and formulas.

Let F be a formula that contains the variable and atomic constant symbols $z_1, \ldots, z_n$. Then the *minterms* of F, denoted $Minterm(F)$, are the set of expressions of the form $\zeta_1 \sqcap \ldots \sqcap \zeta_n$ where each ζ_i is either z_i or $\overline{z_i}$, that is, each minterm must contain each variable and atomic constant symbol either positively or negatively. Note that there are exactly 2^n minterms of F. We can order the minterms in a lexicographic order assuming that positive literals precede negative literals.

Example 6. Suppose that we use only the following variables and no constant symbols in a formula: x, y, z. Then we can form from these variables the following eight minterms in order:

$$\begin{array}{llll} x \cap y \cap z, & x \cap y \cap \overline{z}, & x \cap \overline{y} \cap z, & x \cap \overline{y} \cap \overline{z}, \\ \overline{x} \cap y \cap z, & \overline{x} \cap y \cap \overline{z}, & \overline{x} \cap \overline{y} \cap z, & \overline{x} \cap \overline{y} \cap \overline{z} \end{array}$$

If we also use the atomic constant symbol a, then we can form the following minterms:

$$\begin{array}{llll} x \cap y \cap z \cap a, & x \cap y \cap z \cap \overline{a}, & x \cap y \cap \overline{z} \cap a, & x \cap y \cap \overline{z} \cap \overline{a}, \\ x \cap \overline{y} \cap z \cap a, & x \cap \overline{y} \cap z \cap \overline{a}, & x \cap \overline{y} \cap \overline{z} \cap a, & x \cap \overline{y} \cap \overline{z} \cap \overline{a}, \\ \overline{x} \cap y \cap z \cap a, & \overline{x} \cap y \cap z \cap \overline{a}, & \overline{x} \cap y \cap \overline{z} \cap a, & \overline{x} \cap y \cap \overline{z} \cap \overline{a}, \\ \overline{x} \cap \overline{y} \cap z \cap a, & \overline{x} \cap \overline{y} \cap z \cap \overline{a}, & \overline{x} \cap \overline{y} \cap \overline{z} \cap a & \overline{x} \cap \overline{y} \cap \overline{z} \cap \overline{a} \end{array}$$

□

Note that any minterm that contains two or more atoms positively is equivalent to $\bot$. This allows some simplifications in certain cases.

Each Boolean cardinality constraint with n variables and atomic constants can be put into the *normal form*:

$$c_1|m_1| + \ldots + c_{2^n}|m_{2^n}| \;\; \theta \;\; b \tag{1}$$

where b and each c_i is an integer constant and each m_i is a minterm of the Boolean algebra for $1 \leq i \leq 2^n$, and θ is as in Definition 5.

Example 7. We rewrite S of Example 4 into the following normal form (omitting minterms with zero coefficients and the $\wedge$ symbol at the end of lines):

$$\begin{array}{ll}
|x \cap y \cap z| + |x \cap y \cap \overline{z}| + |\overline{x} \cap y \cap z| + |\overline{x} \cap y \cap \overline{z}| & \equiv_6 \; 0 \\
|x \cap y \cap z| + |x \cap \overline{y} \cap z| + |\overline{x} \cap y \cap z| + |\overline{x} \cap \overline{y} \cap z| & \equiv_9 \; 0 \\
|x \cap y \cap z| + |\overline{x} \cap y \cap z| & = \; 7 \\
|x \cap y \cap z| - |x \cap y \cap \overline{z}| + 2|x \cap \overline{y} \cap z| & = \; 0 \\
|x \cap y \cap z| + |x \cap \overline{y} \cap \overline{z}| & = \; 0
\end{array}$$

The constraint in Example 5 is already in normal form.

3 Quantifier Elimination Method

We give below a quantifier elimination algorithm for Boolean linear cardinality constraint formulas in atomic Boolean algebras of sets.

Theorem 1. [constant-free case] *Existentially quantified variables can be eliminated from Boolean linear cardinality constraint formulas. The quantifier elimination is closed, that is, yields a quantifier-free Boolean linear cardinality constraint formula.* □

Example 8. Let S be the Boolean cardinality formula in Example 4. A quantifier elimination problem would be to find a quantifier-free formula that is logically equivalent to the following:

$$\exists x \; S$$

First we put S into a normal form as shown in Example 7. Then let S^* be the conjunction of the normal form and the following:

$$\begin{array}{l}
|y \cap z| - |x \cap y \cap z| - |\overline{x} \cap y \cap z| = 0 \\
|y \cap \overline{z}| - |x \cap y \cap \overline{z}| - |\overline{x} \cap y \cap \overline{z}| = 0 \\
|\overline{y} \cap z| - |x \cap \overline{y} \cap z| - |\overline{x} \cap \overline{y} \cap z| = 0 \\
|\overline{y} \cap \overline{z}| - |x \cap \overline{y} \cap \overline{z}| - |\overline{x} \cap \overline{y} \cap \overline{z}| = 0
\end{array}$$

Second, we consider each expression that is the cardinality of a minterm (over x, y, z or over y, z) as an integer variable. Then by integer linear constraint

variable elimination we get:

$$\begin{array}{l} |y \cap z| + |y \cap \overline{z}| \equiv_6 0 \\ |y \cap z| + |\overline{y} \cap z| \equiv_9 0 \\ |y \cap z| \qquad\qquad\quad = \ 7 \end{array}$$

By Theorem 1, the above is equivalent to $\exists x\ S$. For instance, the following is one solution for the above:

$$\begin{array}{l} |y \cap z| = 7 \\ |y \cap \overline{z}| = 5 \\ |\overline{y} \cap z| = 2 \\ |\overline{y} \cap \overline{z}| = 0 \end{array}$$

Corresponding to this we can find the solution:

$$\begin{array}{ll} |x \cap y \cap z| = 0 & |\overline{x} \cap y \cap z| = 7 \\ |x \cap y \cap \overline{z}| = 2 & |\overline{x} \cap y \cap \overline{z}| = 3 \\ |x \cap \overline{y} \cap z| = 1 & |\overline{x} \cap \overline{y} \cap z| = 1 \\ |x \cap \overline{y} \cap \overline{z}| = 0 & |\overline{x} \cap \overline{y} \cap \overline{z}| = 0 \end{array}$$

Finally, given any assignment of sets to y and z such that the first group of equalities holds, then we can find an assignment to x such that the second set of equalities also holds. While this cannot be illustrated for all possible assignments, consider just one assignment shown in Figure 2 where each dot represents a distinct person. Given the sets y and z (shown with solid lines), we can find a set x (shown in Figure 2 with a dashed line) that satisfies the required constraints.

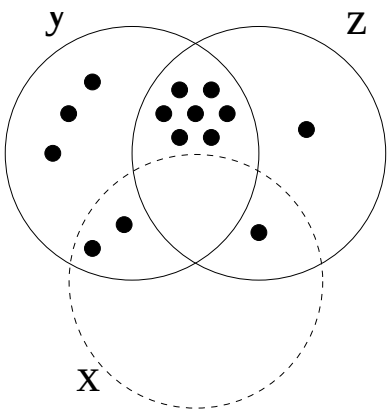

Fig. 2. Venn diagram for the employee example.

□

In the constant-free case, once we know the cardinality of each minterm, any assignment of the required number of arbitrary distinct atoms to the minterms yields an assignment to the variables that satisfies the Boolean formula.

Of course, when the minterms contain atoms, we cannot assign arbitrary atoms. We have to assign to a minterm that contains an atom a positively either the atom that a denotes or the bottom element $\bot$. There is no other choice that can be allowed.

We handle atomic constants as follows. We consider them as additional Boolean algebra variables that have cardinality one and the cardinality of their intersection is zero. That is, if we have atomic constants $a_1, \ldots, a_c$ in a formula, then we add to system (3) the following for each $1 \leq i \leq c$:

$$|a_i| = 1$$

and also for each $1 \leq i, j \leq c$ and $i \neq j$:

$$|a_i \cap a_j| = 0$$

Of course, both of the above conditions have to be put into a normal form before adding them to system (3). Then we solve these as in Theorem 1. Now we can prove the following.

Theorem 2. **[with atomic constants]** *Existentially quantified variables can be eliminated from Boolean linear cardinality constraint formulas. The quantifier elimination is closed, that is, yields a quantifier-free Boolean linear cardinality constraint formula.* □

Example 9. Let S be the Boolean cardinality formula that is the conjunction of the formula in Example 4 and the constraint in Example 5. A quantifier elimination problem would be to find a quantifier-free formula that is logically equivalent to the following:

$$\exists x\ S$$

First we put S into a normal form. Then let S^* be the conjunction of the normal form and the following:

$$\begin{array}{l}
|y \cap z \cap a| - |x \cap y \cap z \cap a| - |\overline{x} \cap y \cap z \cap a| = 0 \\
|y \cap z \cap \overline{a}| - |x \cap y \cap z \cap \overline{a}| - |\overline{x} \cap y \cap z \cap \overline{a}| = 0 \\
|y \cap \overline{z} \cap a| - |x \cap y \cap \overline{z} \cap a| - |\overline{x} \cap y \cap \overline{z} \cap a| = 0 \\
|y \cap \overline{z} \cap \overline{a}| - |x \cap y \cap \overline{z} \cap \overline{a}| - |\overline{x} \cap y \cap \overline{z} \cap \overline{a}| = 0 \\
|\overline{y} \cap z \cap a| - |x \cap \overline{y} \cap z \cap a| - |\overline{x} \cap \overline{y} \cap z \cap a| = 0 \\
|\overline{y} \cap z \cap \overline{a}| - |x \cap \overline{y} \cap z \cap \overline{a}| - |\overline{x} \cap \overline{y} \cap z \cap \overline{a}| = 0 \\
|\overline{y} \cap \overline{z} \cap a| - |x \cap \overline{y} \cap \overline{z} \cap a| - |\overline{x} \cap \overline{y} \cap \overline{z} \cap a| = 0 \\
|\overline{y} \cap \overline{z} \cap \overline{a}| - |x \cap \overline{y} \cap \overline{z} \cap \overline{a}| - |\overline{x} \cap \overline{y} \cap \overline{z} \cap \overline{a}| = 0
\end{array}$$

We also add the constraint $|a| = 1$ expressed using minterms as:

$$\begin{array}{c}
|x \cap y \cap z \cap a| + |x \cap y \cap \overline{z} \cap a| + |x \cap \overline{y} \cap z \cap a| + |x \cap \overline{y} \cap \overline{z} \cap a| \\
+ |\overline{x} \cap y \cap z \cap a| + |\overline{x} \cap y \cap \overline{z} \cap a| + |\overline{x} \cap \overline{y} \cap z \cap a| + |\overline{x} \cap \overline{y} \cap \overline{z} \cap a| = 1
\end{array}$$

Second, we consider each expression that is the cardinality of a minterm (over x, y, z, a or over y, z, a) as an integer variable. Then by integer linear constraint variable elimination we get:

$$\begin{array}{rl}
|y \cap z \cap a| + |y \cap z \cap \overline{a}| + |y \cap \overline{z} \cap a| + |y \cap \overline{z} \cap \overline{a}| & \equiv_6 0 \\
|y \cap z \cap a| + |y \cap z \cap \overline{a}| + |\overline{y} \cap z \cap a| + |\overline{y} \cap z \cap \overline{a}| & \equiv_9 0 \\
|y \cap z \cap a| + |y \cap z \cap \overline{a}| & = 7 \\
|\overline{y} \cap z \cap a| & = 1
\end{array}$$

The last linear cardinality constraint comes from the constraint in Example 5 and the constraint that $|a| = 1$. By Theorem 2, the above is equivalent to $\exists x\ S$. For instance, the following is one solution for the above:

$$\begin{array}{l}
|y \cap z \cap a| = 0 \\
|y \cap z \cap \overline{a}| = 7 \\
|y \cap \overline{z} \cap a| = 0 \\
|y \cap \overline{z} \cap \overline{a}| = 5 \\
|\overline{y} \cap z \cap a| = 1 \\
|\overline{y} \cap z \cap \overline{a}| = 1 \\
|\overline{y} \cap \overline{z} \cap a| = 0 \\
|\overline{y} \cap \overline{z} \cap \overline{a}| = 0
\end{array}$$

Corresponding to this we can find the solution:

$$\begin{array}{ll}
|x \cap y \cap z \cap a| = 0 & |\overline{x} \cap y \cap z \cap a| = 0 \\
|x \cap y \cap z \cap \overline{a}| = 0 & |\overline{x} \cap y \cap z \cap \overline{a}| = 7 \\
|x \cap y \cap \overline{z} \cap a| = 0 & |\overline{x} \cap y \cap \overline{z} \cap a| = 0 \\
|x \cap y \cap \overline{z} \cap \overline{a}| = 2 & |\overline{x} \cap y \cap \overline{z} \cap \overline{a}| = 3 \\
|x \cap \overline{y} \cap z \cap a| = 1 & |\overline{x} \cap \overline{y} \cap z \cap a| = 0 \\
|x \cap \overline{y} \cap z \cap \overline{a}| = 0 & |\overline{x} \cap \overline{y} \cap z \cap \overline{a}| = 1 \\
|x \cap \overline{y} \cap \overline{z} \cap a| = 0 & |\overline{x} \cap \overline{y} \cap \overline{z} \cap a| = 0 \\
|x \cap \overline{y} \cap \overline{z} \cap \overline{a}| = 0 & |\overline{x} \cap \overline{y} \cap \overline{z} \cap \overline{a}| = 0
\end{array}$$

Finally, given any assignment of sets to y, z, and a such that the first group of equalities holds and the correct atom is assigned to a, then we can find an assignment to x such that the second set of equalities also holds. Again, consider just one assignment shown in Figure 3 where each dot represents a distinct person. Given the sets y, z, and a (shown with solid lines), we can find a set x (shown in Figure 3 with a dashed line) that satisfies the required constraints.

It is interesting to compare the above with Example 8. There we could choose for x an arbitrary atom of the set $\overline{y} \cap z$, but here we must choose the atom a. More precisely, we can choose an arbitrary atom of the set $\overline{y} \cap z \cap a$, which, of course, is the same. □

Theorem 3. *Universally quantified variables can be eliminated from Boolean linear cardinality constraint formulas. The quantifier elimination is closed, that is, yields a quantifier-free Boolean linear cardinality constraint formula.* □

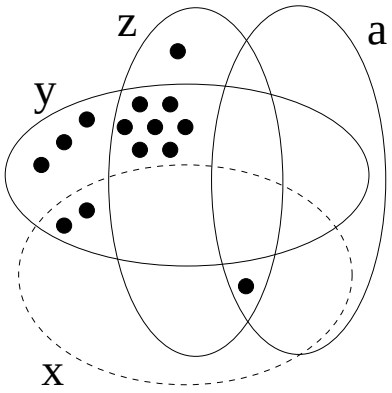

Fig. 3. Venn diagram for Example 9.

Given any formula, it can be decided whether it is true by successively eliminating all variables from it. Then it becomes easy to test whether the variable-free formula is true or false. Hence, this also shows that:

Corollary 1. *It can be decided whether a Boolean linear cardinality constraint formula is true or false.* □

4 Query Evaluation

It is well-known that several practical query languages, such as SQL without aggregation operators, can be translated into relational calculus [1, 39]. Hence while relational calculus is not used directly in major database systems, many theoretical results are stated in terms of it with clear implications for the more practical query languages. Hence we will do the same here.

A relational calculus formula is built from relation symbols R_i with variable and constant symbol arguments, the connectives $\wedge$ for *and*, $\vee$ for *or*, $\rightarrow$ for *implication*, and overline for *negation*, and the quantifiers $\exists$ and $\forall$ in the usual way. Each relation has a fixed *arity* or number of arguments. If R_i is a k-arity relation, then it always occurs in the formula in the form $R_i(z_1, \ldots, z_k)$, where each z_j for $1 \leq j \leq k$ is a variable or constant symbol.

A general framework for using constraint databases is presented in [29, 30]. The following three definitions are from that paper.

1. A *generalized k-tuple* is a quantifier-free conjunction of constraints on k ordered variables ranging over a domain δ. Each generalized k tuple represents in a finite way a possibly infinite set of regular k-tuples.
2. A *generalized relation of arity k* is a finite set of generalized k-tuples, with each k-tuple over the same variables.
3. A *generalized database* is a finite set of generalized relations.

Let r_i be the generalized relation assigned to R_i. We associate with each r_i a formula F_{r_i} that is the disjunction of the set of generalized k-tuples of r_i. According to the above definition, F_{r_i} is a quantifier-free *disjunctive normal form* (DNF) formula. This is not absolutely necessary, and other researchers allow non-DNF formulas too. The above the generalization is from finite relations found in relational database systems to *finitely representable* (via constraints) but possibly infinite relations.

Let ϕ be any relational calculus formula. Satisfaction with respect to a domain δ and database d, denoted $<\delta, d> \models$, is defined recursively as follows:

$$<\delta, d> \models R_i(a_1, \ldots, a_k) \text{ iff } F_{r_i}(a_1, \ldots, a_k) \text{ is true} \quad (2)$$

$$<\delta, d> \models \phi \wedge \psi \text{ iff } <\delta, d> \models \phi \text{ and } <\delta, d> \models \psi \quad (3)$$

$$<\delta, d> \models \phi \vee \psi \text{ iff } <\delta, d> \models \phi \text{ or } <\delta, d> \models \psi \quad (4)$$

$$<\delta, d> \models \phi \rightarrow \psi \text{ iff not } <\delta, d> \models \phi \text{ or } <\delta, d> \models \psi \quad (5)$$

$$<\delta, d> \models \phi \leftrightarrow \psi \text{ iff } <\delta, d> \models \phi \rightarrow \psi \text{ and } <\delta, d> \models \psi \rightarrow \phi \quad (6)$$

$$<\delta, d> \models \overline{\phi} \text{ iff not } <\delta, d> \models \phi \quad (7)$$

$$<\delta, d> \models \exists x_i \phi \text{ iff } <\delta, d> \models \phi[x_i/a_j] \text{ for some } a_j \in \delta \quad (8)$$

$$<\delta, d> \models \forall x_i \phi \text{ iff } <\delta, d> \models \phi[x_i/a_j] \text{ for each } a_j \in \delta \quad (9)$$

where $[x_i/a_j]$ means the instantiation of the free variable x_i by a_j.

The above semantics does not immediately suggest a query evaluation method.[3] In relational databases, the above would suggest a query evaluation, because in the last two rules we clearly need to consider a finite number of cases, but we cannot rely on this finiteness in constraint databases. However, the following alternative semantics, that is equivalent to the above, is discussed in [29, 30]:

Let $\phi(x_1, \ldots, x_m)$ be a relational calculus formula with free variables $x_1, \ldots, x_m$. Let relation symbols $R_1, \ldots, R_n$ in ϕ be assigned the generalized relations $r_1, \ldots, r_n$ respectively. Let $\phi_1 = \phi[R_1/F_{r_1}, \ldots, R_n/F_{r_n}]$ be the formula that is obtained by replacing in ϕ each relation symbol $R_i(z_1, \ldots, z_k)$ by the formula

$$F_{r_i}[x_1/z_1, \ldots, x_k/z_k]$$

where $F_{r_i}(x_1, \ldots, x_k)$ is the formula associated with r_i. Note that ϕ_1 is a simple first order formula of constraints, that is, it does not have any of the relation symbols R_i in it, hence d is no longer relevant in checking the satisfaction of ϕ_1. The output database of ϕ on input database $r_1, \ldots, r_n$ is the relation

$$r_{out} = \{(a_1, \ldots, a_m) \ : <\delta> \models \phi_1(a_1, \ldots, a_m)\}.$$

The above is a possibly infinite relation that needs to be also finitely represented.

[3] This semantic definition closely follows the usual definition of the semantics of relational calculus queries on relational databases. Indeed, the only difference is that in rule (4) the usual statement is that $(a_1, \ldots, a_k)$ is a tuple in the relation or "a row in the table" instead of the tuple satisfying a formula.

Such a finite representation can be found by quantifier elimination. For the goal of quantifier elimination is to find a quantifier-free formula F_1 that has the same models as ϕ_1 has [16]. That is,

$$r_{out} = \{(a_1, \ldots, a_m) \ : < \delta > \models \ F_1(a_1, \ldots, a_m)\}.$$

Hence the alternative semantic definition yields an effective method for query evaluation based on quantifier elimination. Moreover, F_1 can be put into a DNF to preserve the representation of constraint relations.

The above general approach to query evaluation also applies when the input relations are described by linear cardinality constraints in atomic Boolean algebras of sets. In particular, the existential quantifier elimination in Theorem 2 and the universal quantifier elimination in Theorem 3 can be used to evaluate relational calculus queries.

Theorem 4. *Relational calculus queries on constraint relations that are described using quantifier-free Boolean linear cardinality constraint formulas can be evaluated in closed-form.* □

Example 10. Each strand of a DNA can be considered as a string of four letters: A, C, G, and T. In bioinformatics, we often have a set of probes, which are small already sequenced fragments from different already known parts of a long DNA string. For example, one probe may be located from the first to the tenth location on the long DNA strand and be the following:

C A T C G A T C T C

Another may be located between the eighth and 20th and be the following:

C T C G G G A G G G A T C

and so on.

Each of these probes can be represented in B_Z as a tuple of sets (x_A, x_C, x_G, x_T), where x_A, x_C, x_G, and x_T are the positions where A, C, G, and T occur, respectively. Hence the first probe can be represented by:

$$(\{2, 6\},\ \{1, 4, 8, 10\},\ \{5\},\ \{3, 7, 9\})$$

while the second probe can be represented by:

$$(\{14, 18\},\ \{8, 10, 20\},\ \{11, 12, 13, 15, 16, 17\},\ \{9, 19\})$$

Suppose we have a large number of probe data in a relation $Probe(x_A, x_C, x_G, x_T)$, and we'd like to reconstruct the long DNA strand using the probe data. There may be some errors in the sequencing information, for example, a letter A on the DNA may be incorrectly indicated as C in the probe data. Suppose we are satisfied with a 95 percent accuracy regarding the probe data. Suppose also,

that we know that the long DNA sequence contains between 5000 and 6000 letters. The following relational calculus query finds possible long DNA sequences (y_A, y_C, y_G, y_T) that contain each probe with at least a 95 percent accuracy.

$$
\begin{array}{l}
(|y_A \cup y_C \cup y_G \cup y_T| - |y_A| - |y_C| - |y_G| - |y_T| = 0) \ \wedge \\
(|y_A \cup y_C \cup y_G \cup y_T| \ \geq \ 5000) \ \wedge \\
(|y_A \cup y_C \cup y_G \cup y_T| \ \leq \ 6000) \ \wedge \\
(\forall x_A, x_C, x_G, x_T \quad (Probe(x_A, x_C, x_G, x_T) \rightarrow \\
\quad |x_A \cap y_A| + |x_C \cap y_C| + |x_G \cap y_G| + |x_T \cap y_T| \ \geq \ 0.95\,|x_A \cup x_C \cup x_G \cup x_T|))
\end{array}
$$

In the above the solution is (y_A, y_C, y_G, y_T), which we can get by eliminating the universally quantified variables x_A, x_C, x_G, x_T. The first line ensures that the four sets y_A, y_C, y_G, y_T are disjoint. If they are disjoint, then the length of the solution is: $|y_A \cup y_C \cup y_G \cup y_T|$, which is restricted to be between 5000 and 6000 letters in the second and third lines of the relational calculus query. The fourth and fifth lines express that for each probe its overlap with the solution (left hand side of the cardinality constraint) must be greater than or equal to 95 percent of the length of the probe (right hand side).

We also need to check that the solution is a continuous sequence, that is, there are no gaps. We can do that by defining an input relation $Order(x, z)$, which will contain all tuples of the form $(\{i\}, \{j\})$ such that $1 \leq i \leq j \leq 6000$ and $i, j \in \mathcal{N}$. Then the following relational calculus formula tests whether the solution is a continuous sequence:

$$\exists z \, \forall x \ \ Order(x, z) \ \leftrightarrow \ |x \cap (y_A \cup y_C \cup y_G \ y_T)| \ = \ 1$$

The formula says that there must be a last element z in the solution, such that any x is an element of the solution, if and only if it is less than or equal to z. □

5 Conclusions and Future Work

Quantifier elimination from Boolean linear cardinality constraint formulas by reduction to quantifier elimination in Presburger arithmetic is a new approach. The complexity of the quantifier elimination needs to be investigated. Especially the handling of (atomic) constants may be simplified.

It is also interesting to look at more complex cardinality constraints. For example, one cannot express using only linear cardinality constraints that the cardinality of set A is always the square of the cardinality of set B. We avoided such constraints, because even existential quantifier elimination from integer polynomial equations is unsolvable in general [34]. However, with restrictions on the number of variables we may have an interesting solvable problem.

Example 10 shows that relational calculus with Boolean linear cardinality constraints can handle some string problems. It is interesting to compare this in expressive power and computation complexity with query languages for string databases. Grahne et al. [22] proposed an extension of relational calculus with alignment operators for string databases, but the evaluation of their query language is unsolvable in general [22]. Benedikt et al [5] proposed several other

extensions of relational calculus with various string operators. They show that the language S_{len} with only the prefix, the concatenation of a single character of the alphabet (at the end of a string), and the test-of-equal-length operators on strings does not admit quantifier elimination, although some weaker logics have quantifier elimination.

Finally, there are many practical implementation questions, for example, defining algebraic operators for query evaluation, data structures for representing the Boolean linear cardinality constraints, indexing for fast retrieval, and other issues of query optimization.

Acknowledgment: I thank the conference organizers, especially András Benczur, for encouraging submission of this paper.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. D.S. Arnon, G.E. Collins, and S. McCallum. Cylindrical algebraic decomposition, I: The basic algorithm. *SIAM Journal on Computing*, 13:865–877, 1984.
3. S. Basu. New results on quantifier elimination over real closed fields and applications to constraint databases. *Journal of the ACM*, 46(4):537–55, 1999.
4. M. Benedikt, M. Grohe, L. Libkin, and L. Segoufin. Reachability and connectivity queries in constraint databases. In *Proc. ACM Symposium on Principles of Database Systems*, pages 104–15, 2000.
5. M. Benedikt, L. Libkin, T. Schwentick, and L. Segoufin. Definable relations and first-order query languages over strings. *Journal of the ACM*, 50:694–751, 2003.
6. L. Berman. Precise bounds for Presburger arithmetic and the reals with addition. In *Proc. 18th IEEE FOCS*, pages 95–99, 1977.
7. A. Boudet and H. Comon. Diophantine equations, Presburger arithmetic and finite automata. In *Proceedings of CAAP'96*, number 1059 in Lecture Notes in Computer Science, pages 30–43. Springer-Verlag, 1996.
8. A. Brodsky, V. Segal, J. Chen, and P. Exarkhopoulo. The CCUBE constraint object-oriented database system. *Constraints*, 2(3–4):245–77, 1997.
9. J.-H. Byon and P. Revesz. DISCO: A constraint database system with sets. In G. Kuper and M. Wallace, editors, *Proc. Workshop on Constraint Databases and Applications*, volume 1034 of *Lecture Notes in Computer Science*, pages 68–83. Springer-Verlag, 1995.
10. D. Calvanese and M. Lenzerini. On the interaction between ISA and cardinality constraints. In *Proceedings of the Tenth International Conference on Data Engineering*, pages 204–213. IEEE Computer Society Press, 1994.
11. D. Calvanese, M. Lenzerini, and D. Nardi. A unified framework for class based representation formalisms. In *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning*, pages 109–120. Morgan Kaufmann, 1994.
12. B. F. Caviness and J. R. Johnson, editors. *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Springer-Verlag, 1998.
13. G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In H. Brakhage, editor, *Automata Theory and Formal Languages*,

volume 33 of *Lecture Notes in Computer Science*, pages 134–83. Springer-Verlag, 1975.
14. J. Cox and K. McAloon. Decision procedures for constraint based extensions of Datalog. In *Constraint Logic Programming*, pages 17–32. MIT Press, 1993.
15. H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic*. Undergraduate Texts in Mathematics. Springer-Verlag, 2nd edition, 1994.
16. H.B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.
17. M.J. Fisher and M.O. Rabin. Super-exponential complexity of Presburger arithmetic. In *Proc. SIAM-AMS volume VII*. American Mathematical Society, 1974.
18. J. B. J. Fourier. Solution d'une question particuliére du calcul des inégalités. *Nouveau Bulletin des Sciences par la Société philomathique de Paris*, pages 99–100, 1826.
19. C. Gervet. Conjunto: Constraint logic programming with finite set domains. In *Proc. International Logic Programming Symposium*, pages 339–58, 1994.
20. D. Goldin and P. C. Kanellakis. Constraint query algebras. *Constraints*, 1:45–83, 1996.
21. D. Goldin, A. Kutlu, M. Song, and F. Yang. The constraint database framework: Lessons learned from CQA/CDB. In *Proc. International Conference on Data Engineering*, pages 735–737, 2003.
22. G. Grahne, M. Nykänen, and E. Ukkonen. Reasoning about strings in databases. *Journal of Computer and System Sciences*, 59:116–162, 1999.
23. S. Grumbach and Z. Lacroix. Computing queries on linear constraint databases. In *5th International Workshop on Database Programming Languages*, Electronic Workshops in Computing. Springer-Verlag, 1995.
24. S. Grumbach, P. Rigaux, and L. Segoufin. The DEDALE system for complex spatial queries. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 213–24, 1998.
25. S. Grumbach, P. Rigaux, and L. Segoufin. Spatio-temporal data handling with constraints. In *ACM Symposium on Geographic Information Systems*, pages 106–11, 1998.
26. R. Helm, K. Marriott, and M. Odersky. Spatial query optimization: From Boolean constraints to range queries. *Journal of Computer and System Sciences*, 51(2):197–201, 1995.
27. J. Jaffar and J. L. Lassez. Constraint logic programming. In *Proc. 14th ACM Symposium on Principles of Programming Languages*, pages 111–9, 1987.
28. J. Jaffar and M. Maher. Constraint logic programming: A survey. *J. Logic Programming*, 19/20:503–581, 1994.
29. P. C. Kanellakis, G. M. Kuper, and P. Revesz. Constraint query languages. In *Proc. ACM Symposium on Principles of Database Systems*, pages 299–313, 1990.
30. P. C. Kanellakis, G. M. Kuper, and P. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51(1):26–52, 1995.
31. G. M. Kuper, L. Libkin, and J. Paredaens, editors. *Constraint Databases*. Springer-Verlag, 2000.
32. K. Marriott and M. Odersky. Negative Boolean constraints. *Theoretical Computer Science*, 160(1–2):365–80, 1996.
33. K. Marriott and P. J. Stuckey. *Programming with Constraints: An Introduction*. MIT Press, 1998.
34. Y. Matiyasevich. *Hilbert's Tenth Problem*. MIT Press, 1993.
35. H. J. Ohlbach and J. Koehler. *How to extend a formal system with a Boolean algebra operator*, pages 57–75. Kluwer Academic Publisher, 1998.

36. H. J. Ohlbach and J. Koehler. Modal logics, description logics, and arithmetic reasoning. *Journal of Artificial Intelligence*, 109(1-2):1–31, 1999.
37. J. Paredaens, J. Van den Bussche, and D. Van Gucht. First-order queries on finite structures over the reals. *SIAM Journal of Computing*, 27(6):1747–63, 1998.
38. M. Presburger. Über die vollständigkeit eines gewissen systems der arithmetik ganzer zahlen, in welchem die addition als einzige operation hervortritt. In *Comptes Rendus, I. Congrès des Math. des Pays Slaves*, pages 192–201, 1929.
39. R. Ramakrishnan. *Database Management Systems*. McGraw-Hill, 1998.
40. J. Renegar. On the computational complexity and geometry of the first-order theory of the reals. *Journal of Symbolic Computation*, 13(3):255–352, 1992.
41. P. Revesz. A closed form for Datalog queries with integer order. In *International Conference on Database Theory*, volume 470 of *Lecture Notes in Computer Science*, pages 187–201. Springer-Verlag, 1990.
42. P. Revesz. A closed-form evaluation for Datalog queries with integer (gap)-order constraints. *Theoretical Computer Science*, 116(1):117–49, 1993.
43. P. Revesz. Constraint databases: A survey. In B. Thalheim and L. Libkin, editors, *Semantics in Databases*, volume 1358 of *Lecture Notes in Computer Science*, pages 209–46. Springer-Verlag, 1998.
44. P. Revesz. The evaluation and the computational complexity of Datalog queries of Boolean constraint databases. *International Journal of Algebra and Computation*, 8(5):472–98, 1998.
45. P. Revesz. Safe Datalog queries with linear constraints. In *Proc. 4th International Conference on Principles and Practice of Constraint Programming*, volume 1520 of *Lecture Notes in Computer Science*, pages 355–69. Springer-Verlag, 1998.
46. P. Revesz. Safe query languages for constraint databases. *ACM Transactions on Database Systems*, 23(1):58–99, 1998.
47. P. Revesz. *Introduction to Constraint Databases*. Springer-Verlag, New York, 2002.
48. P. Revesz and Y. Li. MLPQ: A linear constraint database system with aggregate operators. In *Proc. 1st International Database Engineering and Applications Symposium*, pages 132–7. IEEE Press, 1997.
49. P. Rigaux, M. Scholl, and A. Voisard. *Spatial Databases with Application to GIS*. Morgan Kaufmann, 2001.
50. A. Salamon. Implementation of a database system with Boolean algebra constraints. Master's thesis, University of Nebraska-Lincoln, May 1998.
51. D. Seipel and U. Geske. Solving cardinality constraints in (constraint) logic programming. In *Proceedings of the International Workshop on Functional and Logic Programming*, 2001.
52. D. Srivastava, R. Ramakrishnan, and P. Revesz. Constraint objects. In A. Borning, editor, *Proc. 2nd International Workshop on Principles and Practice of Constraint Programming*, volume 874 of *Lecture Notes in Computer Science*, pages 218–28. Springer-Verlag, 1994.
53. M. H. Stone. The theory of representations for Boolean algebras. *Transactions of the American Mathematical Society*, 40:37–111, 1936.
54. A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, Berkeley, 1951.
55. H. P. Williams. Fourier-Motzkin elimination extension to integer programming problems. *Journal of Combinatorial Theory (A)*, 21:118–23, 1976.
56. P. Wolper and B. Boigelot. An automata-theoretic approach to Presburger arithmetic constraints. In *Proc. Static Analysis Symposium*, volume 983 of *Lecture Notes in Computer Science*, pages 21–32. Springer-Verlag, 1995.