Safe Datalog Queries with Linear Constraints*

Peter Z. Revesz

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

Peter Z. Revesz

Abstract. In this paper we consider Datalog queries with linear constraints. We identify several syntactical subcases of Datalog queries with linear constraints, called safe queries, and show that the least model of safe Datalog queries with linear constraints can be evaluated bottom-up in closed-form. These subcases include Datalog with only positive and upper-bound or only negative and lower bound constraints or only half-addition, upper and lower bound constraints. We also study other subcases where the recognition problem is decidable.

1 Introduction

Constraint databases is an active area of current research. In particular, linear constraint databases have been used for modeling geometric data and in other applications [3, 14, 15, 22, 23]. There are several proposals to define query languages for linear constraint databases.

Most query language proposals are based on first-order logic. However, it has been found that first-order languages even with real polynomial constraint databases are incapable of expressing many simple recursive queries like finding the transitive closure of an input graph [2].

Other query language proposals are based on fixpoint-logic [1, 29]. Unfortunately, the evaluation of fixpoint queries with linear constraint databases is not guaranteed. This is a major drawback for database use, where non-expert users should be allowed to express new queries without having to worry about termination problems. For example, Kuijpers et al. [22] prove for a five-rule stratified Datalog program that defines topological connectivity that it terminates for any rational linear constraint database input. However, the proof is quite complicated and works only for that single program.

Recently, Grumbach and Kuper [12] have proposed a tractable language with a bounded inflationary fixpoint operator. This is advantageous from the point of guaranteed termination for any query expressible in the language. However, the query language has a syntax and semantics which is not particularly elegant.

In this paper we consider restricted subsets of fixpoint-logic with linear constraint databases. For these restricted cases, we show that termination of the

1ēnThis work was supported in part by NSF grants IRI-9625055 and IRI-9632871.

query evaluation can be guaranteed. In addition, the correctness of the syntax of the language is easy to check even by beginning users, while the semantics of the language is based on standard fixpoint-logic.

An important problem that occurs in database applications is the recognition problem. Given a query program, an input database, a defined relation name R, and a tuple t of constants (rationals or integers) the recognition problem asks whether t is in R within the least fixpoint of the query program and the input database.

The recognition problem is known to be undecidable in general for fixpoint queries and linear constraint databases. In this paper we identify several syntactical subcases of fixpoint queries for which the recognition problem can be solved in finite time. We call these syntactical subcases safe queries.

The primary language based on fixpoint logic is Datalog. Theorem 1 shows that the least fixpoint of any Datalog query with only positive and upper bound constraints or only negative and lower bound constraints is evaluable in PTIME. Theorems 2 shows that the least fixpoint of Datalog queries with integer half-addition constraints is also evaluable in finite time. These theorems extend the known cases of Datalog with constraint queries whose least fixpoints can be found in finite time.

In both cases the least fixpoints will be computed in a constraint form, where the output relations are a set of constraint tuples. Each constraint tuple is a shorthand description for the set of constant tuples that satisfy the constraint. Therefore, it is easy to test whether t is in relation R by testing whether t satisfies any constraint tuple of R. We also consider the complexity of the recognition problem for any fixed program on a variable size input database. This measure, often used in databases, is referred to as the *data complexity* of queries [6, 30]. Theorem 3 shows that Datalog with half-addition constraints has a DEXPTIME-complete data complexity. Theorems 4 and 6 show that the recognition problem is also decidable for two subcases of Datalog with addition and Datalog with linear equation constraints.

This paper is organized as follows. Section 2 lists some basic definitions, including various types of constraints and Datalog queries with constraints. Section 3 describes the evaluation procedure for Datalog queries for positive and upper bound, negative and lower bound and half-addition constraints and analyzes the data complexity of the recognition problem. Sections 4 and 5 study the recognition problem for subcases of Datalog with addition and Datalog with equation constraints. Finally, Section 6 discusses related work and conclusions.

2 Basic Concepts

2.1 Atomic Constraints

In this paper we consider several types of atomic constraints, which are all subcases of *linear constraints* of the form

$$c_1 x_1 + \ldots + c_k x_k \;\theta\; b$$

where c_i is a constant and x_i is a variable for each $1 \leq i \leq k$, the θ is either $\geq$ or $>$ and b is any constant. We call b the *gap-value* in each linear constraint.

We distinguish between two cases of linear constraints depending on the domain of the variables and constants: the domain of *rational linear constraints* is the set of rationals $\mathbf{Q}$ and the domain of *integer linear constraints* is the set of integers $\mathbf{Z}$.

We consider the following subcases of (rational or integer) linear constraints.

Positive Constraint: A positive constraint is an atomic constraint where each coefficient is non-negative.

Negative Constraint: A negative constraint is an atomic constraint where each coefficient is non-positive.

Lower Bound Constraint: A lower bound constraint is an atomic constraint of the form $x \ \theta \ b$.

Upper Bound Constraint: An upper bound constraint is an atomic constraint of the form $-x \ \theta \ b$.

Equality Constraint: An equality constraint is a conjunction of an upper and a lower bound constraint of the form $x \geq b$ and $-x \geq -b$. We abbreviate such a conjunction by $x = b$.

Half-Addition Constraint: A half-addition constraint is an atomic constraint of the form $x_1 + x_2 \geq b$ or $x_1 - x_2 \geq b$ or $-x_1 + x_2 \geq b$ or $-x_1 - x_2 \geq b$ where b is non-negative.

Addition Constraint: An addition constraint is a conjunction of two atomic constraints of the form $-x_1 + x_2 \geq b$ and $x_1 - x_2 \geq -b$. We abbreviate such a conjunction by $x_2 = x_1 + b$.

Equation Constraint: An equation constraint has the same form as an atomic constraint except θ is $=$. An equation constraint can be expressed by a conjunction of two atomic constraints.

Note: An addition constraint can be expressed as $\exists x_3 \quad x_2 + x_3 \geq b$ and $-x_2 - x_3 \geq -b$ and $x_1 + x_3 \geq 0$ and $-x_1 - x_3 \geq 0$. Notice that addition constraints cannot be expressed by half-addition constraints when $b \neq 0$, because either $-b$ or b is negative.

2.2 Datalog with Constraints

The following definition of the syntax and semantics of Datalog programs with constraints extends the definition of Datalog without constraints in [1, 29] and was also given in [18].

Facts: Each input database is a set of facts (also called constraint tuples) that have the form,

$$R_0(x_1, \ldots, x_k) :\!\!-\, \psi. \qquad (fact)$$

where ψ is a conjunction of atomic constraints on $x_1, \ldots, x_k$ which are not necessarily distinct variables or constants.

Rules: Each Datalog program is a set of rules that have the form,

$$R_0(x_1, \ldots, x_k) :\!\!— R_1(x_{1,1}, \ldots, x_{1,k_1}), \ldots, R_n(x_{n,1}, \ldots, x_{n,k_n}), \psi. \qquad (rule)$$

where $R_0, \ldots, R_n$ are not necessary distinct relation symbols and the xs are not necessarily distinct variables or constants and ψ is a conjunction of atomic constraints. We call the left hand side of :— the *head* and the right hand side of :— the *body* of a fact or rule. Several facts or several rules can have the same left-hand relation name. In the facts all variables in the body also appear in the head. In the rules some variables in the body may not appear in the head.

Query: Each Datalog query consists of a Datalog program and an input database.

Example 1 The following query checks whether at least k out of n formulas $f_1(x_1, \ldots, x_m), \ldots, f_n(x_1, \ldots, x_m)$ of atomic constraints can be simultaneously satisfied. We assume that the formulas are in disjunctive normal form and that $f_{i,j}(x_1, \ldots, x_m)$ is the jth disjunct of the ith formula.

Let $C(y, x_1, \ldots, x_m)$ be an input database relation which contains a constraint tuples of the form $y = i, f_{i,j}(x_1, \ldots, x_m)$ for each $f_{i,j}$.

Let $Next(x, y)$ be an input database relation that contains the constraint tuples $x = i, y = i + 1$ for each $0 \leq i \leq n$. Let $Need(x)$ and $Out\_Of(y)$ be the relations that contains $x = k$ and $y = n$, respectively. Then, the query can be expressed as follows.

$$\begin{aligned}
&Sat(x_1, \ldots, x_m) &&:\!\!— Test(x_1, \ldots, x_m, n, k),\ Out\_Of(n)\ Need(k).\\
&Test(x_1, \ldots, x_m, i_1, j_1) &&:\!\!— Test(x_1, \ldots, x_m, i, j),\ Next(i, i_1),\\
& && \qquad C(i_1, x_1, \ldots, x_m),\ Next(j, j_1).\\
&Test(x_1, \ldots, x_m, i_1, j) &&:\!\!— Test(x_1, \ldots, x_m, i, j),\ Next(i, i_1).\\
&Test(x_1, \ldots, x_m, 0, 0).
\end{aligned}$$

The query defines the relation $Test$ such that $Test(x_1, \ldots, x_m, i, j)$ is true for some values of $x_1, \ldots, x_m, i, j$ if and only if out of the first i formulas at least j can be simultaneously satisfied.

Semantics: Let Q be any Datalog query with constraints. We call an *interpretation* of Q any assignment I of a finite or infinite number of tuples over $\delta^{\alpha(R_i)}$ to each R_i that occurs in Q, where δ is the domain of the attribute variables and $\alpha(R_i)$ is the arity of relation R_i.

The *immediate consequence operator* of a Datalog query Q, denoted T_Q, is a mapping from interpretations to interpretations as follows. For each interpretation I:
$R_0(a_1, \ldots, a_k) \in T_Q(I)$ iff there is an instantiation σ of all variables by constants from δ, including variables $x_1, \ldots, x_k$ by constants $a_1, \ldots, a_k$, in either a fact of

the form $(fact)$ such that $\sigma(\psi)$ is true, or a rule of the form $(rule)$ such that $R_i(\sigma(x_{i,1},\ldots,x_{i,k_i})) \in I$ for each $1 \leq i \leq n$ and $\sigma(\psi)$ is true.

Let $T^0_Q(I) = T_Q(I)$. Also let $T^{i+1}_Q(I) = T^i_Q(I) \cup T_Q(T^i_Q(I))$. An interpretation I is called a *least fixpoint* of a query Q iff $I = \bigcup_i T^i_Q(\emptyset)$.

The above is a general definition of the syntax and the semantics of Datalog programs with constraints. In this paper, we will be interested in particular with the following cases of Datalog with constraints:

Datalog^{pos}: positive and upper bound constraints.

Datalog^{neg}: negative and lower bound constraints.

Datalog^{ha}: half-addition, upper and lower bound constraints.

Datalog^{VA}: addition constraints defining vector addition.

Datalog^{MM}: equation constraints defining matrix multiplication.

For the last two types of queries some special restrictions apply that are detailed in Sections 4 and 5.

2.3 Stratified Datalog with Constraints

Semipositive Datalog queries [1, 29] extend Datalog with negation. Syntactically, they are composed of facts of the form $(fact)$ and rules of the form $(rule)$ where a negation symbol may occur before any relation symbol R_i that is the head of some fact.

Semantically, each semipositive Datalog program is a mapping from interpretations to interpretations similarly to Datalog programs except if R_i is negated in a rule, then the consequence operator requires that $R_i(\sigma(x_{i,1},\ldots,x_{i,k_i})) \notin I$.

Example 2 We can modify Example 1 to test whether exactly k formulas are satisfied by inserting $\neg\, C(i_1, x_1, \ldots, x_m)$ into the body of the third rule.

Another extension of Datalog is the class of *stratified Datalog* programs. Each stratified Datalog program Π is the union of semipositive programs $\Pi_1, \ldots, \Pi_k$ satisfying the following property: no relation symbol R that occurs negated in a Π_i is a head of a rule in any Π_j with $j \geq i$. We call P_i the ith stratum of the program.

Each stratified Datalog program is a mapping from interpretations to interpretations. In particular, if Π is the union of the semipositive programs $\Pi_1, \ldots, \Pi_k$ with the above property, then the composition $\Pi_k(\ldots\Pi_1()\ldots)$ is its semantics.

The above is a general definition for semipositive and stratified Datalog programs. In this paper, we will be interested in the following:

Stratified Datalog$^{pos/neg}$: that is, stratified Datalog programs in which:

- Each input database relation is either positive, that is, it contains only positive or upper bound constraints, or negative, that is, it contains only negative or lower bound constraints.

- Each odd stratum contains only unnegated positive input database relations or relations defined in earlier odd strata and negated negative input database relations or relations defined in earlier even strata.
- Each even stratum contains only unnegated negative input database relations or relations defined in earlier even strata and negated positive input database relations or relations defined in earlier odd strata.

3 Evaluation of Datalog with Constraints

In this section we show that the least fixpoint of Datalog^{pos}, Datalog^{neg} and Datalog^{ha} queries can be evaluated bottom-up.

3.1 Constraint Least Fixpoints and Least Models

Constraint Rule Application: Let us assume that we have a rule of the form (*rule*) and we also have given or derived facts for each $1 \leq i \leq n$ of the form:

$$R_i(x_{i,1}, \ldots, x_{i,k_i}) :\!\!- \psi_i(x_{i,1}, \ldots, x_{i,k_i}).$$

where formula ψ_i is a conjunction of constraints. A *constraint rule application* of this rule given these facts as input produces the following derived fact:

$$R_0(x_1, \ldots, x_k) :\!\!- \phi(x_1, \ldots, x_k).$$

where ϕ is a quantifier-free formula that is equivalent to

$$\exists * \psi_1(x_{1,1}, \ldots, x_{1,k_1}), \ldots, \psi_n(x_{n,1}, \ldots, x_{n,k_n}), \psi.$$

where $*$ is the list of the variables in the body of the rule which do not occur in the head of the rule.

The *bottom-up constraint fixpoint evaluation* of Datalog queries starts from the input facts and rules and repeatedly applies one of the rules until no new facts can be derived and added to the database. We call the set of input and derived facts the constraint least fixpoint of the query.

Remember that a constraint tuple is equivalent to a possibly infinite number of regular tuples of constants from the domain. Hence a finite number of constraint tuples could represent an infinite least fixpoint. Proposition 1, which relies on this observation, was proven in many instances in constraint logic programming and constraint databases [16–18].

Proposition 1 For any Datalog with constraints query the bottom-up constraint least fixpoint is equivalent to the least fixpoint.

Observation 1: The evaluation of stratified Datalog queries can be reduced to the evaluation of Datalog queries. We evaluate each stratum by at first replacing in it each negated occurrence of a constraint relation R_i by its complement constraint relation co_R_i. This evaluation gives a constraint least model for the stratified Datalog query.

Proposition 1 gives some idea for computing even infinite least fixpoints in finite time. However, the termination of the constraint least fixpoint evaluation has to be proven for each particular case of constraints. For several cases of constraints termination is not possible. However, we can show termination for Datalog^{pos}, Datalog^{neg}, stratified Datalog$^{pos/neg}$ and Datalog^{ha} queries.

3.2 Termination Proof for Datalog^{pos}, Datalog^{neg} and Stratified Datalog$^{pos/neg}$

Observation 2: For integers we can rewrite each $>$ constraint with gap-value b into an equivalent $\geq$ constraint with gap-value $b+1$ and the same left hand sides. Hence we will assume that we have only $\geq$ constraints in the case of integers. Otherwise, the statements in this section apply to both rationals and integers with small differences that we point out as appropriate.

At first, we prove two quantifier elimination results, the first for positive and upper bound constraints, and the second for negative and lower bound constraints.

Lemma 1 Let S be any conjunction of positive and upper bound constraints over $x, y_1, \ldots, y_n$. Then we can rewrite $\exists x S$ into a logically equivalent conjunction S' of positive and upper bound constraints over $y_1, \ldots, y_n$.

The symmetric case of the above is also closed under quantifier-elimination.

Lemma 2 Let S be any conjunction of negative and lower bound constraints over $x, y_1, \ldots, y_n$. Then we can rewrite $\exists x S$ into a logically equivalent conjunction S' of negative and lower bound constraints over $y_1, \ldots, y_n$.

We can now show the following theorem in case of Datalog with positive or Datalog with negative constraints.

Theorem 1 The least fixpoint of any Datalog^{pos} or Datalog^{neg} query is evaluable in closed form in PTIME in the size of the input database.

By Observation 1 we can reduce the evaluation of stratified Datalog$^{pos/neg}$ queries to evaluating for each stratum either a Datalog^{pos} or Datalog^{neg} query by finding the complement relations before the evaluation of each stratum.

Lemma 3 Let R be any constraint relation with n number of tuples and at most m number of atomic constraints in each tuple. Then the complement relation of R can be found in PTIME in the size of the relation.

The proof of the above Lemma uses the fact that in any fixed k-dimension n number of hyperplanes cut the space into a polynomial in n number of k-dimensional polyhedra, each of which either belongs to R or to its complement. From Lemma 3 follows:

Corollary 1 The least model of any stratified fixed Datalog$^{pos/neg}$ program and variable input database is evaluable in closed form, where relations defined in odd strata will have positive and relations defined in even strata will have negative constraint forms, in PTIME in the size of the input database.

3.3 Termination Proof for Datalog^{ha}

Datalog^{ha} with Integer Domain: By Observation 2 we can again assume that each θ is $\geq$. We can transform any conjunction S of half-addition, lower bound and upper bound constraints over a set of variables $x_1, \ldots, x_{n-1}$ into a S' with only half-addition constraints over $x_1, \ldots, x_{n-1}, d$ where d is the largest absolute value of the gap-values in S. This is because we can replace each $x \geq b$ by $x + d \geq (b + d)$ and each $-x \geq b$ by $-x + d \geq (b + d)$.

We consider d as if it were an nth variable x_n. We could have any distinct pair of the n variables on the left hand side of a half-addition constraint. It does not matter which element of a pair is written first and which is written second. Without loss of generality we can insist that if $\pm x_i \pm x_j$ is on the left hand side, then $i < j$. It is easy to see that there can be only $4n(n-1)/2 = 2n(n-1)$ different left hand sides because there are four distinct cases considering whether x_i and x_j has positive or negative signs.

We further simplify S' so that it contains at most one half-addition constraint with each different left hand side. If S' has several half-addition constraints with the same left hand side all but the one with the highest gap-value is superfluous and is deleted. We call S' the *normal form* of S.

Lemma 4 Let S be any normal form conjunction of half-addition constraints over $x, y_1, \ldots, y_n$ and d. Then we can rewrite $\exists x S$ into a logically equivalent normal form S' of half-addition constraints over $y_1, \ldots, y_n$ and d.

Let us fix any ordering of the $2n(n-1)$ possible left hand sides. Using this fixed ordering, we can represent any S in normal form as a $2n(n-1)$-dimensional point in which the ith coordinate value will be $(b+1)$ if S contains a half-addition constraint with the ith left hand side and b is the gap-value in it, and 0 otherwise.

We say that a point *dominates* another point if it has the same dimension and all of its coordinate values are $\geq$ the corresponding coordinate values in the other point.

Suppose that relation $R(x_1, \ldots, x_{n-1})$ is defined in some Datalog^{ha} program. As the constraint fixpoint evaluation derives new constraint tuples for R, the right hand side of these constraint tuples will be conjunctions of half-addition constraints over the n variables, including d. The sequence of derived constraint tuples can be represented as described above using a point sequence:

$p_1, p_2, \ldots$

It is easy to see that if point p_i dominates point p_j, then p_i and p_j represent conjunctions S_i and S_j of half-addition constraints such that the set of solutions of S_i is included in the set of solutions of S_j. This shows that the fixpoint evaluation could be modified to add only points that do not dominate any earlier point in the sequence. By the geometric Lemma in [24], in any fixed dimension any sequence of distinct points with non-negative integer coordinates must be finite, if no point dominates any earlier point in the sequence. This shows that using a modified constraint fixpoint evaluation:

Theorem 2 The least fixpoint of any Datalog^{ha} query is evaluable in half-addition constraint form when the domain is the integer numbers.

For the recognition problem we can say the following.

Theorem 3 The recognition problem for any fixed Datalog^{ha} program and variable input database has a DEXPTIME-complete data complexity when the domain is the integer numbers.

Datalog^{ha} with Rational Domain: We can assume without loss of generality that the gap-values and the absolute value d are integer numbers, because if they are not, then we can multiply all gap-values by the least common multiple m of all the denumerators. Clearly, $(a_1, \ldots, a_k)$ satisfies a transformed conjunction of constraints if and only if $(\frac{a_1}{m}, \ldots, \frac{a_k}{m})$ satisfies the original constraint.

For dealing with rational numbers we will also treat as a special variable x_{n+1} the value $d + \frac{1}{2}$. Otherwise, the normal form will be defined as in the integer case except we allow both $\geq$ and $>$ comparisons within the half-addition constraints.

Lemma 5 Let S be any normal form conjunction of half-addition constraints over $x, y_1, \ldots, y_n, d, d + \frac{1}{2}$. Then we can rewrite $\exists x S$ into a logically equivalent normal form S' of half-addition constraints over $y_1, \ldots, y_n, d, d + \frac{1}{2}$.

For the rest of the section, the proof is similar to the integer case. Hence we have that:

Corollary 2 The least fixpoint of any Datalog^{ha} query is evaluable in half-addition constraint form when the domain is the rational numbers.

For the recognition problem we can say the following.

Corollary 3 The recognition problem for any fixed Datalog^{ha} program and variable input database has a DEXPTIME-complete data complexity when the domain is the rational numbers.

4 The Recognition Problem for Datalog^{VA} Queries

Datalog^{VA} queries are composed of regular relational database facts (sets of constant tuples) and rules of the form:

$$\begin{aligned} R(x_1, \ldots, x_m, y_1, \ldots, y_k) :\!\!- \; & F_1(y_{1,1}, \ldots, y_{1,k_1}), \ldots, F_n(y_{n,1}, \ldots, y_{n,k_n}), \\ & P(z_1, \ldots, z_m, y_{p,1}, \ldots, y_{p,k_p}), \\ & x_1 = z_1 + c_1, \; \ldots, \; x_m = z_m + c_m. \end{aligned}$$

where the F_is are input database relations, R and P are relation symbols occuring in the head of rules. Relation P and each relation F_i is optional in the rule. The ys are not necessarily distinct variables among themselves but they are all distinct from the x_is and z_is, which are all different.

The domain of the x_i and z_i variables is the set of non-negative integers numbers $\mathbf{N}$, but each c_i and any constant in the input database can be any integer.

The following theorem is proven by reduction of the recognition problem for Datalog^{VA} to the recognition problem in *vector addition systems with states*, VASS, which is shown to be decidable in [21]. VASS is a generalization the reachability problem in Petri nets, for which the containment problem is undecidable. This implies for Datalog^{VA} queries the following.

Theorem 4 The recognition problem for Datalog^{VA} queries is decidable and for semipositive Datalog^{VA} queries is undecidable.

Nevertheless, it is possible to prove the following.

Theorem 5 It can be decided whether a Datalog^{VA} query is safe, i.e., its output can be represented in constraint form for any valid input database.

5 The Recognition Problem for Datalog^{MM} Queries

Datalog^{MM} queries are composed of regular relational database facts and rules of the form:

$$\begin{aligned} R(x_1,\ldots,x_m) :\!\!-\; & R(y_1,\ldots,y_m), \\ & -x_1 + c_{1,1}y_1 + \ldots + c_{1,m}y_m = 0, \\ & \vdots \\ & -x_m + c_{m,1}y_1 + \ldots + c_{m,m}y_m = 0. \end{aligned}$$

or

$$R(x_1,\ldots,x_m) :\!\!-\; F(x_1,\ldots,x_m).$$

where F is a regular input relation (sets of constant tuples), the x_is and y_is are all different variables, and each $c_{i,j}$ is a rational constant. The domain of the variables is the set of rational numbers.

It can be seen that Datalog^{MM} queries can express sets of *Markov processes* when we make the restriction that for each j the $\sum_{1 \leq i \leq m} c_{i,j} = 1$ and use only a single recursive rule. We call this condition (1).

Further, it is known that the value of Markov processes approach a steady state when all the $c_{i,j}$s are positive. We call this condition (2).

Therefore, when conditions (1&2) hold, then after some finite number of rule applications we will only get $R(a_1,\ldots,a_m)$ tuples such that $\mid a_i - b_i \mid < \epsilon$ where $(b_1,\ldots,b_n)$ is the steady state value and ϵ is an arbitrarily small positive rational number.

We define the recognition problem with ϵ tolerance the task of deciding whether there is a tuple in the least fixpoint of the query such that each of its elements is within an ϵ distance from the corresponing element in the given tuple. Taking advantage of the steady state convergence of Markov processes [28], we can prove the following.

Theorem 6 The recognition problem with ϵ tolerance for Datalog^{MM} quereis satisfying conditions (1&2) is decidable.

6 Related Works and Conclusion

Datalog^{pos}, Datalog^{neg} and Datalog^{ha} queries are cases of constraint logic programs whose syntax and semantics was defined in a general way in [16]. Constraint bottom-up evaluations for constraint queries (both constraint logic programs and constraint relational calculus queries) were considered within a constraint database framework in [18] and many recent papers (see [17, 26] for surveys on constraint logic programming and constraint databases).

A *gap-order constraint* is a lower bound constraint, an upper bound constraint or a constraint of the form $x + b \leq y$ where $b \geq 0$. Note that all gap-order constraints are half-addition constraints, but some half-addition constraints are not gap-order constraints. For example, $x + y \geq 5$ is a half-addition constraint but it is not expressible by gap-order constraints. A least fixpoint evaluation for Datalog with gap-order constraints is described in [24]. The recognition problem is also studied in [8]. The DISCO system [5] implements Datalog queries with integer gap-order constraints. Adding negation in a safe way to Datalog with gap-order queries is studied in [25].

A *temporal constraint* is like a gap-order constraint but the gap-value can be any integer (both negative and non-negative). Temporal constraints can express addition constraints. Hence the recognition problem for Datalog with temporal constraints is undecidable. However, an evaluation of relational calculus queries with temporal constraints is possible and is considered by Koubarakis in [19, 20]. Efficient tests for temporal constraint satisfaction are described in [9] and for monotone two-variable constraints in [11].

Chomicki and Imielinski [7] consider the language Datalog$_{1S}$ which is like Datalog extended with an increment operator which may occur only in the first argument of relations. Linear recursive Datalog$_{1S}$ is a subcase of Datalog^{VA}. The least fixpoint is evaluable for Datalog$_{1S}$ queries [7].

Fribourg and Olsén [10] consider the connection between Petri nets and a subset of Datalog^{VA} programs. [10] shows that the least fixpoint of those queries that can be represented by a special case of Petri nets, called BPP-nets, is evaluable in finite time.

There is a growing number of implementations of first-order constraint queries with linear constraint databases, for example CCUBE [4], DEDALE [13] and MLPQ [27]. The query evaluation algorithms described in this paper could be useful extensions of these systems as well as some constraint logic programming systems, for example CLP(R), that implement linear constraints. We already started implementing safe recursive queries in MLPQ.

References

1. S. Abiteboul, R. Hull and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.
2. M. Benedikt, G. Dong, L. Libkin, L. Wong. Relational Expressive Power of Constraint Query Languages. *Journal of the ACM*, vol. 45, 1–34, 1998.
3. A. Brodsky, J. Jaffar, M.J. Maher. Toward Practical Query Evaluation for Constraint Databases. *Constraints*, vol. 2, no. 3&4, 279–304, 1997.
4. A, Brodsky, V.E. Segal, J. Chen, P.A. Exarkhopoulo. The CCUBE Constraint Object-Oriented Database System. *Constraints*, vol. 2., no. 3&4, 245–278, 1997.
5. J. Byon, P.Z. Revesz, DISCO: A Constraint Database System with Sets, *Proc. Workshop on Constraint Databases and Applications*, Springer-Verlag LNCS 1034, 68–83, 1995.
6. A.K. Chandra, D. Harel. Structure and Complexity of Relational Queries. *Journal of Computer and System Sciences*, vol. 25, 99–128, 1982.
7. J. Chomicki, T. Imielinski. Finite Representation of Infinite Query Answers. *ACM Transactions of Database Systems*, vol. 18, no. 2, 181–223, 1993.
8. J. Cox, K. McAloon. Decision Procedures for Constraint Based Extensions of Datalog. In: F. Benhamou, A. Colmerauer, eds., *Constraint Logic Programming*, MIT Press, 1993.
9. R. Dechter, I. Meiri, J. Pearl. Temporal Constraint Networks. *Artificial Intelligence*, vol. 49, 61–95, 1991.
10. L. Fribourg, H. Olsén. A Decompositional Approach for Computing Least Fixed-Points of Datalog Programs with $\mathcal{Z}$-Counters. *Constraints*, vol. 2, no. 3&4, 305–336, 1997.
11. D. Goldin, P.C. Kanellakis. Constraint Query Algebras. *Constraints*, vol. 1, no. 1&2, 54-83, 1996.
12. S. Grumbach, G. M. Kuper. Tractable Recursion over Geometric Data. *Proc. Third International Conference on Principles and Practice of Constraint Programming*, Springer-Verlag LNCS 1330, 450-462, 1997.
13. S. Grumbach, P. Rigaux, L. Segoufin. The DEDALE System for Complex Spatial Queries. *Proc. ACM SIGMOD International Conference on Management of Data*, ACM Press, 213–224, 1998.
14. S. Grumbach, J Su, C. Tollu. Linear Constraint Query Languages: Expressive Power and Complexity. *Proc. Logic and Computational Complexity*, Springer-Verlag LNCS 960, 1994.
15. M. Gyssens, J. Van den Bussche, D. Van Gucht. Complete Geometrical Query Languages. *Proc. 16th ACM Symposium on Principles of Database Systems*, 62–67, 1997.
16. J. Jaffar, J.L. Lassez. Constraint Logic Programming. *Proc. 14th ACM Symposium on Principles of Programming Languages*, 111–119, 1987.
17. J. Jaffar, M.J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, vol. 19 & 20, 503–581, 1994.
18. P. C. Kanellakis, G. M. Kuper, P. Z. Revesz. Constraint Query Languages. *Journal of Computer and System Sciences*, vol. 51, no. 1, 26-52, 1995.

19. M. Koubarakis. Database Models for Infinite and Indefinite Temporal Information. *Information Systems*, vol. 19, no. 2, 141–173, 1994.
20. M. Koubarakis. The Complexity of Query Evaluation in Indefinite Temporal Constraint Databases. *Theoretical Computer Science*, vol. 171, no. 1&2, 25–60, 1997.
21. S. R. Kosaraju. Decidability of Reachability in Vector Addition Systems. *Prof. 14th ACM Symposium on Theory of Computing*, 267-281, 1982.
22. B. Kuijpers, J. Paredaens, M. Smits, J. Van den Bussche. Termination Properties of Spatial Datalog Programs. In: D. Pedreschi and C. Zaniolo, eds., *Logic in Databases*, Springer-Verlag LNCS 1154, 101–116, 1997.
23. J. Paredaens, J.V.D. Bussche, D.V. Gucht. First-Order Queries on Finite Structures over the Reals. *Proc. Symp. on Logic in Computer Science*, 1995.
24. P. Z. Revesz. A Closed Form Evaluation for Datalog Queries with Integer (Gap)-Order Constraints. *Theoretical Computer Science*, vol. 116, no. 1, 117-149, 1993.
25. P. Z. Revesz. Safe Query Languages for Constraint Databases. *ACM Transactions on Database Systems.* vol. 23, no. 1, 1–43, 1998.
26. P. Z. Revesz. Constraint Databases: A Survey. In: L. Libkin and B. Thalheim, eds., *Semantics in Databases*, Springer-Verlag LNCS 1358, 209–246, 1998.
27. P. Z. Revesz, Y. Li. MLPQ: A Linear Constraint Database System with Aggregate Operators. *Proc. International Database Engineering and Applications Symposium*, IEEE Press, 132–137, 1997.
28. H. Schneider, G. P. Barker. *Matrices and Linear Algebra.* 2nd ed., Holt, Rinehart and Winston, 1973.
29. J.D. Ullman. *Principles of Database and Knowledge-Base Systems*, vols 1&2. Computer Science Press, 1989.
30. M. Vardi. The Complexity of Relational Query Languages. *Proc. 14th ACM Symposium on the Theory of Computing*, 137–145, 1982.

Proof of Lemma 1: First, simplify the positive constraints by deleting all variables with zero coefficients. Now, S' will contain the set of constraints that do not contain the variable x and all constraints that can be derived from some upper bound constraint of the form $-x\theta_1 b$ and a positive constraint of the form $c_0x + c_1y_1 + \ldots + c_ny_n\theta_2 a$ where c_i is a positive rational number for $0 \leq i \leq n$. The new constraint created will be $c_1y_1 + \ldots + c_ny_n\theta_3 a + c_0b$, where θ_3 is $\geq$ if θ_1 and θ_2 are both $\geq$ and $>$ otherwise. This is still a positive constraint and if a, b and c_0 are integers, then the new gap-value created will be also an integer. Therefore, upper bound and positive constraints are closed under existential quantifier elimination in the case of both rationals and integers. We can prove the soundness of the quantifier elimination similarly to the proof of Lemma 4.

Proof of Lemma 2: This case is symmetric to the case of Lemma 1. All constraints created will be between a lower bound constraint and a negative constraint.

Proof of Theorem 1: Let us consider any relation $R(x_1, \ldots, x_k)$. Each fact of R will contain a conjunction of positive and upper bound constraints. Note that the left hand side of each positive constraint will be the same as the left hand side of a positive constraint in the input database or one of the rules except that some coefficients can be changed to zero. Therefore, the number of different left hand sides of positive constraints is a finite number (if we have m variables in a positive constraint, then there could be 2^m different left hand sides that could be generated from it).

Let C be the set of positive coefficients in any positive constraint in the input database or the rules. Let B be the set of gap-values in any upper bound constraint. Let D denote the set of all possible products of an element in B and and element in C. Now, consider any positive constraint with m variables. As we eliminate any variable from it by adding it to an upper bound constraint we always add an element of D to the gap-value of the positive constraint. Further we can add only m times. Hence the number of possible gap-values that can be created is finite. In fact, if there are at most m variables in any positive constraint, then the set of possible gap-values that can be created are $S = \{b + d_1 + \ldots + d_m \ : \ b \in B, d_i \in D \cup \{0\}, 1 \leq i \leq m\}$. For any fixed program, m will be a constant equivalent to the maximum number of variables in any rule or fact. Hence the number of possible gap-values that can be created is polynomial in the size of B and C and hence also in the size of the input database.

Since both the set of left hand sides and the set of right hand sides that could occur in any constraint in any fact of R is a polynomial in the input database size, the number of possible constraints and the number of possible facts of R is also polynomial in it. The fixpoint evaluation needs to continue at most the number of different constraint tuples that could be added to the database. Since that is a polynomial number in the input database size, after that many iterations the fixpoint evaluation can stop. Hence each fixed Datalog^{pos} query can be evaluated in PTIME in the size of the input database.

A similar argument can show that Datalog^{neg} is also evaluable in PTIME data complexity.

Proof of Lemma 4: S' will be the conjunction of all the half-addition constraints in S that do not contain the variable x and all the half-addition constraints that can be derived from any pair of half-addition constraints in S using the implication table below.

	$x - z \geq b$	$-x + z \geq b$	$x + z \geq b$	$-x - z \geq b$
$x - y \geq a$		$-y + z \geq a + b$		$-y - z \geq a + b$
$-x + y \geq a$	$y - z \geq a + b$		$y + z \geq a + b$	
$x + y \geq a$		$y + z \geq a + b$		$y - z \geq a + b$
$-x - y \geq a$	$-y - z \geq a + b$		$-y + z \geq a + b$	

Given any two half-addition constraints with opposite signs for x, their sum is returned by the implication table. It is easy to see that if S consisted of half-addition constraints, then S' will contain only half-addition constraints because in each constraint created using the implication table the gap-value will be the sum of two gap-values already present in S. Therefore only non-negative gap-values will be created using the implication table. The only case that merits special mention is when y and z are the same variables. In that case we may obtain either $2y \geq a+b$ or $-2y \geq a+b$. These two cases can be rewritten into half-addition constraint form as $y+d \geq floor(\frac{a+b}{2})+d$ and $-y+d \geq floor(\frac{a+b}{2})+d$ respectively, where the floor function takes the smallest integer value that is greater than or equal to any given rational value.

For any instantiation, if two half-addition constraints are both true, then their sum also must be a true half-addition constraint. Hence if S is true, then S' must be also true for any instantiation of the variables $x, y_1, \ldots, y_n$.

For the other direction, suppose that S' is true for some instantiation of the variables $y_1, \ldots, y_n$. Then make the same instantiation into S. After the instantiation, x will be the only remaining variable in S. Wherever x occurs positively, the constraint implies a lower bound for x, and wherever x occurs negatively the constraint implies an upper bound for x.

Suppose that the largest lower bound l is implied by some constraint f and the smallest upper bound u is implied by some constraint g. Since the sum of f and g under the current instantiation is equivalent to $l \leq u$ and is in S', which is true, we can find a value between l and u inclusively for x that will make S also true.

Proof of Theorem 2: We can modify the basic fixpoint evaluation method by adding for each relation R only "points" that do not dominate any earlier point added to relation R. This shows that the number of points added to R must be finite. By reasoning similarly to R for each defined relation, we can see that the modified fixpoint evaluation must terminate. The correctness of the modification follows from the observation that points that are not added are not new (in the sense that there is no instantiation which makes them true but does not make any other input or already derived fact true).