The Evaluation and the Computational Complexity of Datalog Queries of Boolean Constraint Databases*

Peter Z. Revesz
Department of Computer Science and Engineering
University of Nebraska–Lincoln, Lincoln, NE 68588-0115, USA

Abstract

In the database framework of Kanellakis et al. it was argued that constraint query languages should meet the closed-form requirement, that is, they should take as input constraint databases and give as output constraint databases that use the same type of constraints. This paper shows that the closed-form requirement can be met for Datalog queries with Boolean equality constraints with double exponential time-complete data complexity, for Datalog queries with precedence and monotone inequality constraints in triple exponential-time data complexity. A closed-form evaluation is also shown for (Stratified) Datalog queries with equality and inequality constraints in atomless Boolean algebras in triple exponential-time data complexity.

1 Introduction

Constraint databases describe extensional database relations as quantifier-free first-order formulas. Constraint databases in the form of non-ground facts have been used in constraint logic programming [9, 10, 19] for almost ten years. Constraint databases also generated much interest recently in the database community [11]. In the database framework of [12] it was argued that the closed-form requirement for relational query languages should extend to constraint query languages. That is, constraint query languages should take as input constraint databases and give as output constraint databases that use the same type of constraints.

Boolean terms within logic programs were first considered by Büttner and Simonis [4]. Their method relies on Boolean term unification [15]. Boolean constraints within constraint logic programs were first considered in [12] where querying Boolean constraint databases with Boolean equality constraints over free algebras was analyzed.

[12] proved that Datalog queries with Boolean equality constraint databases can be evaluated in closed-form with Π_2^p-hard data complexity. (Data complexity is the measure of the computational complexity of fixed queries as the size of the input database grows [5, 21]. The rationale behind this measure is that in practice the size of the database typically dominates by several orders of magnitude the size of the query.)

In Section 2 we review Boolean algebras and two quantifier elimination methods in Boolean algebras. We also describe a new quantifier elimination method. In Section 3 we consider Datalog queries. We first describe a parametric fixpoint evaluation method for Datalog queries. Then we show that Datalog queries with equality constraints can be evaluated in double exponential-time, with precedence and monotone inequality constraints in triple exponential time, and with equality and inequality constraints in atomless Boolean algebras in triple exponential time data complexity. We also show that all of these cases have a double exponential time-hard data complexity lower bound. In Section 4 we consider Stratified Datalog

*This work was supported in part by NSF grants IRI-9625055 and IRI-9632871.

queries. We show that Stratified Datalog queries with equality and inequality constraints in atomless Boolean algebras can be evaluated in closed-form in triple exponential time data complexity. Finally Section 5 discusses other related work and gives some concluding remarks.

2 Basic Concepts

In this section we review the basic concepts of Boolean algebras, Boolean constraints, and quantifier elimination in Boolean algebras. We also present one new quantifier elimination method.

2.1 Boolean Algebras

A *Boolean algebra* B is a sextuple $\langle \delta, \wedge, \vee, ', 0, 1 \rangle$, where δ is a set called the domain, $\wedge$, $\vee$ are binary functions, $'$ is a unary function and 0, 1 are two specific elements of δ (called the *zero* and the *identity* element, respectively) such that for any elements x, y, and z in δ the following axioms hold:

$$\begin{array}{rclrcl}
x \vee y & = & y \vee x & x \wedge y & = & y \wedge x \\
x \vee (y \wedge z) & = & (x \vee y) \wedge (x \vee z) & x \wedge (y \vee z) & = & (x \wedge y) \vee (x \wedge z) \\
x \vee x' & = & 1 & x \wedge x' & = & 0 \\
x \vee 0 & = & x & x \wedge 1 & = & x \\
0 & \neq & 1 & & &
\end{array}$$

For Boolean algebras there is a representation theorem, known as *Stone's theorem* [3, 7]: "Every Boolean algebra is isomorphic to Boolean algebra of sets (where $\wedge, \vee, '$ are interpreted as $\cap, \cup$, and set difference from 1, respectively) and every finite Boolean algebra is isomorphic to the power set of a finite set". Thus, there is a unique (up to isomorphism) finite Boolean algebra for every cardinality 2^m. The Boolean algebra of cardinality 2^{2^m} is *the one freely generated by m generators* and is denoted by B_m. For $m = 0$, we have B_0=$\langle \{0, 1\}, \wedge, \vee, ', 0, 1 \rangle$.

Boolean Terms: We use $T(F, V \cup C)$ for the set of *terms* built in the usual way from F, the set of function symbols and the zero and the identity elements $\{\wedge, \vee, ', 0, 1\}$, V a set of variable symbols, and C a set of constant symbols distinct from $0, 1$. *Ground terms* are those terms which do not have any variable symbols appearing in them. A (B, σ)*-interpretation* is a pair, where B is a Boolean algebra and σ is a mapping of the constant symbols C to the elements of B. For each t in $T(F, V \cup C)$, given a (B, σ)-interpretation and an element of B for each variable symbol appearing in t, we can evaluate t in the usual way and have it denote one element of B.

A *Boolean equality constraint* between terms t_1 and t_2 in $T(F, V \cup C)$ is a statement of the form $t_1 =_{B,\sigma} t_2$ for a (B, σ)-interpretation. A *Boolean inequality constraint* is a statement of the form $t_1 \neq_{B,\sigma} t_2$. A *solution* of an equality (or inequality) constraint is a substitution of elements of B for the variable symbols that makes t_1 and t_2 denote the same element (or distinct elements) of B when σ is applied to the constant symbols. We say the equality (or inequality) constraint is true if every substitution of elements of B for the variable symbols is a solution. (We will sometimes omit B or σ when it is obvious from the context.)

Lemma 2.1 Let $t(z_1, z_2, \ldots, z_n)$ be a term, where the z's are the distinct variable or constant symbols occurring in it. Then the following equation is true:

$$t(z_1, z_2, \ldots, z_n) = (t(0, z_2, \ldots, z_n) \wedge z_1') \vee (t(1, z_2, \ldots, z_n) \wedge z_1).\Box$$

Disjunctive Normal Form: We use the convention that z^0 means z' and z^1 means z, and that the z's are ordered from z_1 to z_n. Then, we also may write the equation in the previous lemma as

$t(z_1, z_2, \ldots, z_n) = \bigvee_{a_1 \in \{0,1\}} (t(a_1, z_2, \ldots, z_n) \wedge z_1^{a_1})$. By repeatedly using the above lemma and the nine Boolean algebra axioms, it is possible to transform each term into the following *disjunctive normal form*:

$$t(z_1, \ldots, z_n) = \bigvee_{\overline{a} \in \{0,1\}^n} (t(a_1, \ldots, a_n) \wedge z_1^{a_1} \wedge z_2^{a_2} \wedge \ldots \wedge z_n^{a_n})$$

where $\bigvee_{\overline{a} \in \{0,1\}^n}$ denotes the disjunction of all 0, 1 substitutions for $a_1, \ldots, a_n$. The function determined by $t(z_1, \ldots, z_n)$ depends only on the values of the 2^n expressions $t(a_1, \ldots, a_n)$, where each a_i is either 0 or 1. One can see that each of these 2^n expressions has value either 0 or 1. Hence, it is possible to see that there are 2^{2^n} disjunctive normal forms with n variable and constant symbols.

Constructing B_m: We give a simple example of how to construct the free Boolean algebra B_m out of a set of m constant symbols $C = \{c_1, \ldots, c_m\}$. First we build all possible ground terms. Next we find all the equivalence classes of ground terms under the Boolean algebra axioms. Each equivalence class is an element of B_m and corresponds to a disjunctive normal form. There are 2^{2^m} distinct equivalence classes or elements of B_m. We call the constant symbols the *generators*, all possible conjunctions of the generators the *minterms* of the algebra. All elements of the Boolean algebra can be expressed as a disjunction of some minterms.

Example 2.1 Consider the free Boolean algebra B_3 generated by the constant symbols a, b, c. B_3 has 2^3 minterms. These minterms in descending order of binary superscript values are:

$a \wedge b \wedge c$, $a \wedge b \wedge c'$, $a \wedge b' \wedge c$, $a \wedge b' \wedge c'$, $a' \wedge b \wedge c$, $a' \wedge b \wedge c'$, $a' \wedge b' \wedge c$, $a' \wedge b' \wedge c'$

By Lemma 2.1 each element of B_3 can be written in a disjunctive normal form in which each minterm either occurs as a disjunct or not. Hence there are 2^{2^3} elements of the free Boolean algebra B_3. □

Example 2.2 In a free Boolean algebra the generators don't come with an interpretation of the Boolean algebra operators. We give an example of a (B, σ)-interpretation. The symbol $\wedge$ is interpreted as set intersection, $\vee$ as set union, and $'$ as set difference from $\{1, 2, 3, 4, 5, 6, 7, 8\}$. Also,

$\sigma(a) = \{1, 2, 3, 4\}$
$\sigma(b) = \{1, 2, 5, 6\}$
$\sigma(c) = \{1, 3, 5, 7\}$

The interpreted Boolean algebra B is $\langle \delta, \wedge, \vee, ', \emptyset, \{1, 2, 3, 4, 5, 6, 7, 8\} \rangle$ where δ is the powerset of the set $\{1, 2, 3, 4, 5, 6, 7, 8\}$. In B the minterms are all different elements:

$a \wedge b \wedge c =_{B,\sigma} \{1\}, a \wedge b \wedge c' =_{B,\sigma} \{2\}, a \wedge b' \wedge c =_{B,\sigma} \{3\}, a \wedge b' \wedge c' =_{B,\sigma} \{4\}$,
$a' \wedge b \wedge c =_{B,\sigma} \{5\}, a' \wedge b \wedge c' =_{B,\sigma} \{6\}, a' \wedge b' \wedge c =_{B,\sigma} \{7\}, a' \wedge b' \wedge c' =_{B,\sigma} \{8\}$

The free Boolean algebra B_3 under the interpretation (B, σ) became another specific Boolean algebra with 8 different minterms that are not equal to $\emptyset$, and a total of 2^8 elements. □

Example 2.3 Now let us look at a different interpretation (B, σ_2) for the free Boolean algebra B_3. In this interpretation assume the same meaning for the Boolean algebra operators as in Example 2.2, but assume that σ_2 is the following:

$\sigma_2(a) = \{1\}$
$\sigma_2(b) = \{2\}$
$\sigma_2(c) = \{3\}$

In this case some of the minterms are equivalent. In fact,

$a \wedge b \wedge c =_{B,\sigma_2} \emptyset, a \wedge b \wedge c' =_{B,\sigma_2} \emptyset, a \wedge b' \wedge c =_{B,\sigma_2} \emptyset, a \wedge b' \wedge c' =_{B,\sigma_2} \{1\}$,
$a' \wedge b \wedge c =_{B,\sigma_2} \emptyset, a' \wedge b \wedge c' =_{B,\sigma_2} \{2\}, a' \wedge b' \wedge c =_{B,\sigma_2} \{3\}, a' \wedge b' \wedge c' =_{B,\sigma_2} \{4, 5, 6, 7, 8\}$

Under this interpretation of the constants, only a subalgebra of the Boolean algebra in Example 2.2

is generated. This subalgebra has only 4 different minterms that are not equal to $\emptyset$ and only 2^4 elements. $\square$

Atomless Boolean algebras have the property that between 0 and any element there is always another element [3, 7, 14]. In Section 4 we will require that the interpretation yields an atomless Boolean algebra. (We do not care which one, just that it has the above property.)

Example 2.4 Let B_h be the Boolean algebra whose elements are finite unions of half open intervals over the rational numbers, including the emptyset and the set of all rational numbers $\mathcal{Q}$. In B_h let $\wedge$ mean intersection, $\vee$ mean union, and $'$ mean complement with respect to the set of rational numbers. Further let a, b, c be three constants allowed within the terms, and let their interpretation be defined by σ_3 as follows:

$\sigma_3(a) = [2, 10)$
$\sigma_3(b) = [3, 35)$
$\sigma_3(c) = [4, 21)$

The minterms of the free Boolean algebra B_3 under the interpretation (B_h, σ_3) are various elements of B_h. For example, the minterm $a \wedge b \wedge c$ would be the half open interval $[4, 10)$. It is not an atom because it contains the half open interval $[5, 8)$ which is also an element of B_h. The Boolean algebra B_h is a well-known example of an atomless Boolean algebra. $\square$

2.2 Quantifier Elimination

In this section we consider existentially quantified formulas, which we define in the standard way [6]. We say that two formulas are equivalent in a given Boolean interpretation (B, σ) if they have the same set of solutions, i.e., substitutions for the free variables that make them true. The purpose of quantifier elimination is to rewrite an existentially quantified formula into an equivalent quantifier-free formula [6].

One quantifier elimination method which originates with George Boole and which follows easily from Lemma 2.1 is the following.

Lemma 2.2 For any Boolean interpretation (B, σ), the formula $\exists x\ (f(x, y_1, \ldots, y_k) =_{B,\sigma} 0)$ is equivalent to $f(0, y_1, \ldots, y_k) \wedge f(1, y_1, \ldots, y_k) =_{B,\sigma} 0$. $\square$

Another quantifier elimination method for atomless Boolean algebras was proven recently by Marriott and Odersky [14]. This method eliminates an existentially quantified variable from a system of Boolean equation and inequation constraints and returns as a solution another system.

Lemma 2.3 For any Boolean interpretation (B, σ) where B is atomless, the system of formulas

$$\begin{aligned}\exists x \quad & [f(x, y_1, \ldots, y_k) =_{B,\sigma} 0, \\ & g_1(x, y_1, \ldots, y_k) \neq_{B,\sigma} 0, \\ & \vdots, \\ & g_l(x, y_1, \ldots, y_k) \neq_{B,\sigma} 0]\end{aligned}$$

is equivalent to

$$\begin{aligned}& f(0, y_1, \ldots, y_k) \wedge f(1, y_1, \ldots, y_k) =_{B,\sigma} 0, \\ & (f(1, y_1, \ldots, y_k)' \wedge g_1(1, y_1, \ldots, y_k)) \vee (f(0, y_1, \ldots, y_k)' \wedge g_1(0, y_1, \ldots, y_k)) \neq_{B,\sigma} 0, \\ & \vdots \\ & (f(1, y_1, \ldots, y_k)' \wedge g_l(1, y_1, \ldots, y_k)) \vee (f(0, y_1, \ldots, y_k)' \wedge g_l(0, y_1, \ldots, y_k)) \neq_{B,\sigma} 0. \ \square\end{aligned}$$

Boole's method is limited because it allows quantifier elimination only from formulas that have only equality constraints. Marriott and Odersky's method is more general syntactically, but it works only for atomless Boolean algebras. In the following, we present a new quantifier elimination method that is syntactically between the two other methods but also works for any Boolean algebra. First we need some more definitions.

We call a constraint of the form $x \wedge y' =_{B,\sigma} 0$, abbreviated as $x \leq_{B,\sigma} y$, a *precedence* constraint.

We say that g is a *monotone* Boolean function if $g(x_1, \ldots, x_n) \leq_{B,\sigma} g(y_1, \ldots, y_n)$ whenever $x_i \leq_{B,\sigma} y_i$ for each $1 \leq i \leq n$. We call a constraint of the form $g(x_1, \ldots, x_n) \neq_{B,\sigma} 0$, where g is a monotone Boolean function, a *monotone inequality constraint.*

The following are a some useful facts about the precedence relation [7]. For any elements x, y, z, w in any Boolean interpretation (B, σ),

$$x \leq_{B,\sigma} x \qquad \text{(reflexivity)}$$

$$x \leq_{B,\sigma} y \text{ and } y \leq_{B,\sigma} x \text{ imply } x =_{B,\sigma} y \qquad \text{(antisymmetry)}$$

$$x \leq_{B,\sigma} y \text{ and } y \leq_{B,\sigma} z \text{ imply } x \leq_{B,\sigma} z \qquad \text{(transitivity)}$$

$$0 \leq_{B,\sigma} x \qquad \text{(zero element)}$$

$$(x \wedge y) \leq_{B,\sigma} x \qquad \text{(augment)}$$

$$x \leq_{B,\sigma} y \text{ and } w \leq_{B,\sigma} z \text{ imply } (x \wedge w) \leq_{B,\sigma} (y \wedge z) \qquad \text{(merge)}$$

Now we can state and prove the new quantifier elimination method.

Lemma 2.4 For any Boolean interpretation (B, σ) and monotone Boolean functions $g_1, \ldots, g_l$ and y, z, v, w, u's that are constants or variables not necessarily distinct from each other but different from x, the system of formulas:

$$\begin{array}{ll} \exists x & [z_1 \leq_{B,\sigma} x, \ldots, z_m \leq_{B,\sigma} x, \\ & x \leq_{B,\sigma} y_1, \ldots, x \leq_{B,\sigma} y_k, \\ & w_1 \leq_{B,\sigma} u_1, \ldots, w_s \leq_{B,\sigma} u_s, \\ & g_1(x, v_1, \ldots, v_n) \neq_B 0, \\ & \vdots \\ & g_l(x, v_1, \ldots, v_n) \neq_B 0] \end{array}$$

is equivalent to

$$\begin{array}{l} z_1 \leq_{B,\sigma} y_1, \ldots, z_j \leq_{B,\sigma} y_i, \ldots, z_m \leq_{B,\sigma} y_k, \\ w_1 \leq_{B,\sigma} u_1, \ldots, w_s \leq_{B,\sigma} u_s, \\ g_1((y_1 \wedge \ldots \wedge y_k), v_1, \ldots, v_n) \neq_B 0, \\ \vdots \\ g_l((y_1 \wedge \ldots \wedge y_k), v_1, \ldots, v_n) \neq_B 0. \end{array}$$

Proof: Suppose that the first formula is true for some substitution for the variables other than x. Then there exists a substitution for x that makes the first formula without the quantifier true. With that substitution $z_j \leq_{B,\sigma} x$ and $x \leq_{B,\sigma} y_i$ are both true for all $1 \leq i \leq k$ and $1 \leq j \leq m$. Hence by transitivity $z_j \leq_{B,\sigma} y_i$ must be true. Further, by merge $x \leq_{B,\sigma} y_1, \ldots, x \leq_{B,\sigma} y_k$ imply that $x \leq_{B,\sigma} (y_1 \wedge \ldots \wedge y_k)$. This and the monotonicity of g_p imply that $g_p(x, v_1, \ldots, v_n) \leq_{B,\sigma} g_p((y_1 \wedge \ldots \wedge y_k), v_1, \ldots, v_n)$ for each $1 \leq p \leq l$.

Since the first formula is true, $g_p(x, v_1, \ldots, v_n) \neq_{B,\sigma} 0$ is true for each $1 \leq p \leq l$. Suppose now that for some p the constraint $g_p((y_1 \wedge \ldots \wedge y_k), v_1, \ldots, v_n) \neq_{B,\sigma} 0$ is false. Then $g_p((y_1 \wedge \ldots \wedge y_k), v_1, \ldots, v_n) =_{B,\sigma} 0$ would be true, and by reflexivity $g_p((y_1 \wedge \ldots \wedge y_k), v_1, \ldots, v_n) \leq_{B,\sigma} 0$ would be also true. Then by transitivity $g_p(x, v_1, \ldots, v_n) \leq_{B,\sigma} 0$ would be true. By the zero element rule $0 \leq_{B,\sigma} g_p(x, v_1, \ldots, v_n)$. The last two conditions and the antisymmetry rule imply that $g_p(x, v_1, \ldots, v_n) =_{B,\sigma} 0$ which is a contradiction. Hence $g_p((y_1 \wedge \ldots \wedge y_k), v_1, \ldots, v_n) \neq_{B,\sigma} 0$ must be true for each $1 \leq p \leq l$. Therefore each constraint in the second formula must be true using the same substitutions as in the first formula.

For the other direction, suppose that the second formula is true for some substitution for the variables in it. Then $z_j \leq_{B,\sigma} y_1, \ldots, z_j \leq_{B,\sigma} y_k$ imply using merge that $z_j \leq_{B,\sigma} (y_1 \wedge \ldots \wedge y_k)$ for each $1 \leq j \leq m$. The augmentation rule implies that $(y_1 \wedge \ldots \wedge y_k) \leq_{B,\sigma} y_i$ for each $1 \leq i \leq k$. Therefore, the same substitution and the choice of $y_1 \wedge \ldots \wedge y_k$ for x makes all the precedence constraints in the first formula true. Also, this choice for x makes each monotone inequality in the first formula equivalent to the corresponding monotone inequality in the second formula, which we assumed to be true. Hence the first formula must be also true using the same substitutions. □

3 Datalog Queries with Boolean Constraints

3.1 Syntax and Semantics

The following definition of the syntax and semantics of Datalog programs with Boolean Constraints extends the definition of Datalog without constraints in [20] and Datalog with only Boolean equality constraints in [12].

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

$$R_0(x_1, \ldots, x_k) :\!\!— f(x_1, \ldots, x_k) =_{B,\sigma} 0, g_1(x_1, \ldots, x_k) \neq_{B,\sigma} 0, \ldots, g_l(x_1, \ldots, x_k) \neq_{B,\sigma} 0.$$

where the f and the gs are Boolean terms.

Rules: Datalog programs consist of a set of rules that have the form,

$$R_0(x_1, \ldots, x_k) :\!\!— R_1(x_{1,1}, \ldots, x_{1,k_1}), \ldots, R_n(x_{n,1}, \ldots, x_{n,k_n}), f(\overline{x}) =_{B,\sigma} 0, g_1(\overline{x}) \neq_{B,\sigma} 0, \ldots, g_l(\overline{x}) \neq_{B,\sigma} 0.$$

where $R_0, \ldots, R_n$ are not necessary distinct relation symbols and the xs are either variables or constants, and $\overline{x}$ is the set of variables in the rule. We call the left hand side of :— the head and the right hand side of :— the body of the 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.

In general any conjunction of equality and inequality constraints could be allowed in the facts and rules. We have only defined a standard form for facts and rules. In the standard form each fact or rule has only one equality constraint of the form $f =_{B,\sigma} 0$. This is without loss of generality because each equality constraint of the form $f =_{B,\sigma} h$ is equivalent to the constraint $(f \wedge h') \vee (f' \wedge h) =_{B,\sigma} 0$ and similarly for inequality constraints. Further a set of equality constraints of the form $f_1 =_{B,\sigma} 0, \ldots, f_n =_{B,\sigma} 0$ is equivalent to the single equality constraint $f_1 \vee \ldots \vee f_n =_{B,\sigma} 0$. To easier readability, in the examples below we will sometimes use the general form instead of the standard form. The standard form will be an important simplification in describing the evaluation procedure and the proofs.

Example 3.1 Let the constant c describe the set of computer science majors, m the set of mathematics majors, and a, d, s the set of students who took an abstract algebra, a database systems, or a software engineering class at a university. Suppose that all non-mathematics majors who took an abstract algebra class are eligible for a mathematics minor, and all non-computer science students who took either a database systems or a software engineering class are eligible for a computer science minor. This can be expressed using the following facts.

$$\begin{array}{lll} math\_minor(X) & :\!- & X =_{B,\sigma} m' \wedge a. \\ cs\_minor(X) & :\!- & X =_{B,\sigma} c' \wedge (d \vee s). \end{array}$$

Example 3.2 Continuing Example 3.1, suppose further that all students who are either computer science majors or are eligible for a computer science minor can apply to the computer science master's program. This can be expressed by the rule.

$$can\_apply\_cs\_MS(X) \;:\!-\; cs\_minor(Y), X =_{B,\sigma} c \vee Y.$$

Example 3.3 Let relation $range(X, R)$ describe that animal X occurs in the range R within a national park. Suppose that animal a is infected with a virus which spreads by contact with other animals. The following Datalog program defines the set of animals which are in danger of being infected by animal a.

$$\begin{array}{lll} infection\_area(R) & :\!- & range(a, R). \\ infection\_area(R) & :\!- & infection\_area(R1), range(X, R2), R1 \wedge R2 \neq_{B,\sigma} 0, R =_{B,\sigma} R1 \vee R2. \\ in\_danger(X) & :\!- & infection\_area(R1), range(X, R2), R1 \wedge R2 \neq_{B,\sigma} 0. \end{array}$$

In this example a possible (B, σ) interpretation would be the subsets of the two dimensional real space $\mathcal{R}^2$, with the operator $\wedge$ meaning intersection, $\vee$ meaning union, and $'$ meaning complement with respect to the two dimensional real space. Animals could be identified by single points in $\mathcal{R}^2$, for example, the first coordinate value could be some identification number and the second the year of birth. $\square$

Each *Datalog query* consists of a set of facts (input database) and a set of rules (Datalog program). For example, the set of facts in Example 3.1 and the rule in Example 3.2 together express a Datalog query. The set of facts in Example 3.3 is a Datalog program, but it is not a complete query because we did not give any facts. (An empty set of facts is acceptable, if it is truely intented. However, here we just omitted to mention the facts.)

Semantics: Let Π be any Datalog query using (B, σ) Boolean constraints. We call an *interpretation* of Π any assignment I of a finite or infinite number of tuples over $\delta^{\alpha(R_i)}$ to each R_i that occurs in Π, where δ is the domain of B.

We call *valuation* any function from (tuples of) variables to (tuples of) elements of B satisfying the following: for all tuples t_1 and t_2, $\nu(t1, t2) = (\nu(t_1), \nu(t_2))$, and for all constants c, $\nu(c) = \sigma(c)$.

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:

$R_0(a_1, \ldots, a_k) \in T_Q(\emptyset)$ iff there is a valuation ν and a fact of the form
$R_0(x_1, \ldots, x_k) :\!- f(x_1, \ldots, x_k) =_{B,\sigma} 0, g_1(x_1, \ldots, x_k) \neq_{B,\sigma} 0, \ldots, g_l(x_1, \ldots, x_k) \neq_{B,\sigma} 0$ in Π such that
$\nu(x_1, \ldots, x_k) = (a_1, \ldots, a_k)$ and
$f(\nu(x_1, \ldots, x_k)) =_{B,\sigma} 0$ is true, and $g_j(\nu(x_1, \ldots, x_k)) \neq_{B,\sigma} 0$ is true for each $1 \leq j \leq l$.

$R_0(a_1, \ldots, a_n) \in T_Q(I)$ iff there is a valuation ν and a rule of the form
$R_0(x_1, \ldots, x_k) :\!- R_1(x_{1,1}, \ldots, x_{1,k_1}), \ldots, R_n(x_{n,1}, \ldots, x_{n,k_n}), f(\overline{x}) =_{B,\sigma} 0, g_1(\overline{x}) \neq_{B,\sigma} 0, \ldots, g_l(\overline{x}) \neq_{B,\sigma} 0$ in Π such that
$\nu(x_1, \ldots, x_n) = (a_1, \ldots, a_n)$ and $R_i(\nu(x_{i,1}, \ldots, x_{i,k_i})) \in I$ for each $1 \leq i \leq n$ and
$\nu(\overline{x}) = (\overline{a})$ and $f(\nu(\overline{x})) =_{B,\sigma} 0$ is true, and $g_j(\nu(\overline{x})) \neq_{B,\sigma} 0$ is true for each $1 \leq j \leq l$.

Let $T_Q^0(\emptyset) = T_Q(\emptyset)$. Also let $T_Q^{i+i}(I) = T_Q^i(I) \cup T_Q(T_Q^i(I))$.

An interpretation I is called a *fixpoint* of a query Q iff $I = \bigcup_i T_Q^i(\emptyset)$.

Example 3.4 For example, let us assume the interpretation B for the Boolean algebra operators as in Example 2.2, and for the constants σ_4 as follows:

$\sigma_4(a) = \{1, 2, 5, 6\}$
$\sigma_4(c) = \{5, 6, 7, 8\}$
$\sigma_4(d) = \{2, 4, 6, 8\}$
$\sigma_4(m) = \{1, 2, 3, 4\}$
$\sigma_4(s) = \{3, 6\}$

In this case we have the following:

$T_Q^0(\emptyset) = \{cs\_minor(\{2,3,4\}), math\_minor(\{5,6\})\}$.

$T_Q^1(\emptyset) = T_Q^0(\emptyset) \cup \{can\_apply\_cs\_MS(\{2,3,4,5,6,7,8\})\}$.

$T_Q^2(\emptyset) = T_Q^1(\emptyset) = \{cs\_minor(\{2,3,4\}), math\_minor(\{5,6\}), can\_apply\_cs\_MS(\{2,3,4,5,6,7,8\})\}$.

Therefore the fixpoint of the Datalog query is $T_Q^2(\emptyset)$. □

It is easy to see that we can do this evaluation for any Datalog query where (B, σ) is a finite Boolean algebra.

Proposition 3.1 Let Q be any Datalog query with equality and inequality constraints over some Boolean interpretation (B, σ) where B is a Boolean algebra with N number of elements. Then Q can be evaluated in $O(N^c)$-time for some constant c.

Proof: We evaluate the query using the consequence operator T_Q. In the evaluation we try to substitute all possible elements of the subalgebra into the variables of each fact to get $T_Q^0(\emptyset)$. Then we repeatedly try out all possible substitutions into the rules. We stop when the value of T_Q does not increase from one iteration to another.

In this evaluation, we add in each iteration except the last one at least one tuple to the current value of T_Q. Let r be the number of relation symbols and v be the maximum arity of any relation in Q. Then T_Q can have at most rN^v tuples in it. Therefore $O(rN^v)$ number of iterations are enough to evaluate Q. Further, in each iteration we have to try out all possible substitutions within each rule. The maximum number of variables in each rule is also some fixed constant d. That means we need to try out N^d substituitons for each rule. Checking whether a substitution of the variables satisfies a Boolean equality or inequality constraint takes a linear time in the size of the term on the left hand side. Hence the total evaluation time will be $O(N^c)$ for some constant c. □

We also need sometimes to consider infinite Boolean algebras. We can not do a fixpoint evaluation in an infinite Boolean algebra by trying out all possible substitutions. In this case we have to use some quantifier elimination based fixpoint evaluation, which we describe in the next section.

It is also possible that B or σ is not known. In this case a *parametric* constraint evaluation is used. In the parametric evaluation of Datalog queries we use the free Boolean algebra generated by some set of constants C which includes all the constant symbols in the query. The parametric evaluation will be equivalent to the fixpoint evaluation, for *any acceptable* (B, σ) interpretation. We will specify precisely in each case what class of interpretations are acceptable.

3.2 Parametric Evaluation of Datalog Queries

Parametric Rule Application: Let us assume that we have a rule of the form:

$$R_0(x_1, \ldots, x_k) :\!\!-\, R_1(x_{1,1}, \ldots, x_{1,k_1}), \ldots, R_n(x_{n,1}, \ldots, x_{n,k_n}), f(\overline{x}) =_{B,\sigma} 0, g_1(\overline{x}) \neq_{B,\sigma} 0, \ldots, g_l(\overline{x}) \neq_{B,\sigma} 0.$$

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 Boolean equality and inequality constraints.

A *parametric rule application* or *firing* of this rule given these facts as input will produce 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

$$\exists * \psi_1(x_{1,1},\ldots,x_{1,k_1}),\ldots,\psi_n(x_{n,1},\ldots,x_{n,k_n}), f(\overline{x}) =_{B,\sigma} 0, g_1(\overline{x}) \neq_{B,\sigma} 0,\ldots,g_l(\overline{x}) \neq_{B,\sigma} 0.$$

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 parametric evaluation* of Datalog queries consists of repeatedly firing rules starting from the input facts and rules, until no new facts can be derived. The set of derived facts will be a parametric description of the fixpoint of the Datalog query.

Example 3.5 Let C be any set of constants that includes the constants a, c, d, m, s. Let us consider a parametric evaluation of the Datalog query in Examples 3.1 and 3.2. We have to use the free Boolean algebra B_C generated by the constant symbols in C. In this case we have the following.

Let $\psi_{cs\_minor}(X)$ be equivalent to $X =_{B,\sigma} c' \wedge (d \vee s)$. The formula $\psi_{cs\_minor}(X)$ is the right hand side of a fact of *cs_minor*. Hence we can apply the rule by replacing *cs_minor*(Y) by ψ_{cs_minor} after renaming X by Y. Then we obtain,

$$can\_apply\_cs\_MS(X) :\!\!- Y =_{B,\sigma} c' \wedge (d \vee s), X =_{B,\sigma} c \vee Y.$$

Hence the derived fact for *can_apply_cs_MS* will be:

$$can\_apply\_cs\_MS(X) :\!\!- \phi(X).$$

where $\phi(X)$ is a quantifier-free formula that is equivalent to $\exists Y\; Y =_{B,\sigma} c' \wedge (d \vee s), X =_{B,\sigma} c \vee Y$. The latter formula can be rewritten using the identities mentioned above as:

$$\exists Y\; (Y' \wedge (c' \wedge (d \vee s))) \vee (Y \wedge (c' \wedge (d \vee s))') \vee (X' \wedge (c \vee Y)) \vee (X \wedge (c \vee Y)') =_{B,\sigma} 0.$$

Now we can apply Lemma 2.2, which will yield:

$$((1' \wedge (c' \wedge (d \vee s))) \vee (1 \wedge (c' \wedge (d \vee s))') \vee (X' \wedge (c \vee 1)) \vee (X \wedge (c \vee 1)'))\wedge$$
$$((0' \wedge (c' \wedge (d \vee s))) \vee (0 \wedge (c' \wedge (d \vee s))') \vee (X' \wedge (c \vee 0)) \vee (X \wedge (c \vee 0)')) =_{B,\sigma} 0.$$

In any Boolean algebra $1' = 0$ and $0' = 1$. Using this and the other Boolean algebra identities, the above can be simplified to:

$$((c' \wedge (d \vee s)') \vee X') \wedge ((c' \wedge (d \vee s))) \vee (X' \wedge c) \vee (X \wedge c')) =_{B,\sigma} 0.$$

Further simplifying, we get

$$((c' \wedge d' \wedge s') \vee X') \wedge ((c' \wedge d) \vee (c' \wedge s) \vee (X' \wedge c) \vee (X \wedge c')) =_{B,\sigma} 0.$$

Finally, putting the constraint into DNF, we get for $\phi(X)$ the following:

$$(X \wedge c' \wedge d' \wedge s') \vee (X' \wedge c' \wedge d) \vee (X' \wedge c' \wedge s) \vee (X' \wedge c) =_{B,\sigma} 0.$$

When we repeat the fixpoint iteration, no semantically new facts can be derived. Hence we the fixpoint will be the two input fact and the above derived fact. As a check at the correctness of the parametric evaluation, let's consider the (B, σ) interpretation used in Example 3.4. After substitution for the constants, we the above constraint becomes $(X \wedge \{1\}) \vee (X' \wedge \{2,4\}) \vee (X' \wedge \{3\}) \vee (X' \wedge \{5,6,7,8\}) =_{B,\sigma} 0$. In any solution of this constraint X has to contain $2,3,4,5,6,7,8$ and cannot contain 1. The only element of the Boolean algebra (B, σ) in Example 3.4 that satisfies all of these constraints is the element $\{2,3,4,5,6,7,8\}$. This is the same that we obtained in the non-parametric evaluation. □

This bottom-up parametric evaluation is easily seen to be both sound and complete because the quantifier elimination preserves the set of solutions. However, we still need to consider closed-form, termination and computational complexity. Closed form means that the derived facts will contain the same type of constraints that the input facts and rules.

It can be seen that using any of the three quantifier elimination methods preserves the closed-form. Boole's method rewrites formulas with only equality constraints into formulas with equality constraints. (Remember that several equality constraints can be rewritten into a single equality constraint in standard form. Hence Boole's method can be easily extended for systems of equality constraint formulas.) Marriott and Odersky's method rewrites standard form constraints into standard form constraints. Finally, the new quantifier elimination method rewrites formulas with precedence and monotone inequality constraints into quantifier-free formulas with precedence and monotone inequality constraints. The inequality constraints are monotone, because it can be easily shown using the merge rule that if $g(x, v_1, \ldots, v_n)$ is a monotone Boolean function, then $g((y_1 \wedge \ldots \wedge y_k), v_1, \ldots, v_n)$ is also a monotone Boolean function.

We can combine the same Datalog program with several different input database. It is interesting to look at the computational complexity of queries with fixed Datalog programs and variable input databases. This commonly used measure is called the *data complexity* of queries [5, 21]. Let us now consider the computational data complexity upper bound of Datalog queries.

Theorem 3.1 Let Q be any Datalog query which contains *only equality constraints* in some Boolean interpretation (B, σ). Then Q can be evaluated bottom-up in closed form in double exponential time data complexity.

Proof: Let v be the maximum arity of a relation in the given query program. Let m be the number of constant symbols in the query. Since the program is fixed, v is some constant, but m may vary with the size of the database. Our evaluation method will consist of a number of iterative steps. In each step we add all new facts that can be derived from the already known facts and rules. By the proper substitutions of database facts into the rules we get formulas on the right-hand side of the rules. From these formulas we eliminate existentially quantified variables using Lemma 2.2.

We also have to show that the procedure terminates in $O(2^{2^m})$-time. To do that, we always keep every fact in disjunctive normal form. Note that Lemma 2.2 does not introduce any new constant symbols, hence the number of constant symbols m does not change during evaluation. The quantifier elimination yields constraints that have up to m constant symbols and v variables. That means that the constraints for each relation R can be represented by at most $2^{2^{v+m}}$ facts by counting only disjunctive normal forms. Hence, after each iteration step, every newly derived constraint can be compared easily with facts already present in the database. If all newly derived constraints are already present, then the iteration can stop, otherwise we add the new constraints. This procedure clearly must terminate because there are only

$O(2^{2^{v+m}})$ number of facts that can be added for each relation with v arity. Since v is a fixed constant for each fixed Q, the theorem holds. □

Theorem 3.2 Let Q be any Datalog query with constraints in some Boolean interpretation (B, σ) where B is an *atomless* algebra. Then Q can be evaluated bottom-up in closed form in triple exponential time data complexity.

Proof: We can argue similarly to Theorem 3.1. The difference is that we use now the quantifier elimination algorithm in Lemma 2.3 and we have to consider the inequality constraints. Clearly, this quantifier elimination algorithm also does not introduce new constants. All derived facts may contain only those constants that occur in the query. We have already argued that there can be $2^{2^{v+m}}$ different Boolean functions with v variables and m constant symbols. Now, each of these may occur within the equality or in one of the inequality constraints in the standard form of a fact. Hence there can be $2^{2^{2^{v+m+1}}}$ different facts. Hence we would need that many iterations at most for the parametric evaluation. □.

Theorem 3.3 Let Q be any Datalog query which contains *only precedence* and *monotone inequality* constraints in some Boolean interpretation (B, σ). Then Q can be evaluated bottom-up in closed form in triple exponential time data complexity.

Proof: The argument is similar to Theorem 3.2. Let v be the maximum arity of any relation in the given query program. Let m be the number of constant symbols in the program and the input database. We can express v^2 different precedence constraints between two variables, $2v2^{2^m}$ different precedence constraints between a variable and an element of the free Boolean algebra B_m.

We have seen in Lemma 3.1 that there are $2^{2^{v+m}}$ different Boolean functions. Hence there are also $O(2^{2^{v+m}})$ monotone Boolean functions. For any relation R in Q, in each fact of R some subset of these monotone Boolean functions will occur within the monotone inequality constraints. There are $O(2^{2^{2^{v+m}}})$ different subsets of these monotone Boolean functions. Therefore, considering the number of precedence and inequality constraints possible and their different conjunctions not counting order, there are $O(2^{2^{2^{cm}}})$ different facts of R for some constant c.

In the bottom-up evaluation we use Lemma 2.4. The bottom-up evaluation must terminate because for each R the evaluation can add only $O(2^{2^{2^{cm}}})$ number of facts. □

3.3 Lower Bounds

For the lower bound, we consider Datalog queries with only equality constraints using the interpretation (B, σ) where σ maps each constant to itself and B is B_m where m is the number of distinct constant symbols that occur in the query.

At first we show that given a binary input relation *next_gen* which describes an ordering on the generator symbols in the query, there is a Datalog query that gives as output a *next* relation which describes the ordering of the minterms formable from these generators, assuming that the minterms are ordered according to the binary value of the superscript $\overline{a}$.

Lemma 3.1 Let $g_1, \ldots, g_m$ be constant symbols. Suppose that we are given the relations $next\_gen(g_1, g_2)$, $\ldots, next\_gen(g_{m-1}, g_m)$ and $last\_gen(g_m)$. Then we can define in Datalog the relation $next(X, Y)$ that is true if and only if X and Y are minterms generated by $g_1, \ldots, g_m$ and the minterm number of X is one less than the minterm number of Y.

Proof: We define the $next(X, Y)$ relation as follows. If in binary the ith digit of a minterm X is 1 then g_i should be a factor in it, and if it is 0 then g_i' should be a factor in X. We can express these as $X \wedge g_i =_{B,\sigma} X$ and $X \wedge g_i' =_{B,\sigma} X$ respectively.

$$\begin{array}{lll}
next(X,Y) & :\!\!- & next\_minterm2(X,Y,M), last\_gen(M). \\
 & & \\
next\_minterm2(X,Y,I) & :\!\!- & next\_minterm2(X,Y,J), next\_gen(J,I), X \wedge I' =_{B,\sigma} X, Y \wedge I' =_{B,\sigma} Y. \\
next\_minterm2(X,Y,I) & :\!\!- & next\_minterm2(X,Y,J), next\_gen(J,I), X \wedge I =_{B,\sigma} X, Y \wedge I =_{B,\sigma} Y. \\
next\_minterm2(X,Y,g_1) & :\!\!- & X \wedge g_1' =_{B,\sigma} X, Y \wedge g_1 =_{B,\sigma} Y. \\
next\_minterm2(X,Y,I) & :\!\!- & next\_minterm3(X,Y,J), next\_gen(J,I), X \wedge I' =_{B,\sigma} X, Y \wedge I =_{B,\sigma} Y. \\
 & & \\
next\_minterm3(X,Y,I) & :\!\!- & next\_minterm3(X,Y,J), next\_gen(J,I), X \wedge I =_{B,\sigma} X, Y \wedge I' =_{B,\sigma} Y. \\
next\_minterm3(X,Y,g_1) & :\!\!- & X \wedge g_1 =_{B,\sigma} X, Y \wedge g_1' =_{B,\sigma} Y.
\end{array}$$

□

Next we show that given as input the *next* relation on the minterms, there is another Datalog query that gives as output a *succ* relation which describes an ordering on the elements of the free Boolean algebra, assuming that the elements are ordered according to the binary value of the superscript of the minterms. (Here a one (or zero) superscript means that the minterm is (or is not) in the disjunctive normal form of the element.)

Lemma 3.2 Suppose that we are given the relations $next(t_1,t_2),\ldots,next(t_{2^m-1},t_{2^m})$ and $first\_minterm(g_1' \wedge \ldots \wedge g_m')$ and $last\_minterm(g_1 \wedge \ldots \wedge g_m)$ that order the minterms of the free Boolean algebra B_m generated by $g_1,\ldots,g_m$. Then we can define in Datalog the relation $succ(N,M)$ that is true if and only if N and M are elements of the free Boolean algebra B_m and the number of M is one less than the number of N.

Proof: We encode each integer number from 0 to $2^{2^m}-1$ as some element of B_m. Each number n is represented by the element that contains the minterm t_i if and only if in the binary encoding of n the ith digit from the right is 1. For example, the number 9 is represented in the free Boolean algebra B_2 by the element $\{t_4,t_1\}$. In the rest of the paper let e_n denote the element of B_m that represents the integer number n.

We express the successor relation $succ(N,M)$ which is true if and only if M,N represent the numbers i,j respectively and $i=j+1$ for any $i,j<2^{2^m}$ by the following rules.

$$\begin{array}{lll}
succ(N,M) & :\!\!- & succ2(N,M,S), last\_minterm(S). \\
 & & \\
succ2(N,M,I) & :\!\!- & succ2(N,M,J), next(J,I), zero(N,I), zero(M,I). \\
succ2(N,M,I) & :\!\!- & succ2(N,M,J), next(J,I), one(N,I), one(M,I). \\
succ2(N,M,I) & :\!\!- & zero(N,I), one(M,I), first\_minterm(I). \\
succ2(N,M,I) & :\!\!- & succ3(N,M,J), next(J,I), zero(N,I), one(M,I). \\
 & & \\
succ3(N,M,I) & :\!\!- & succ3(N,M,J), next(J,I), one(N,I), zero(M,I). \\
succ3(N,M,I) & :\!\!- & one(N,I), zero(M,I), first\_minterm(I). \\
 & & \\
zero(N,I) & :\!\!- & N \wedge I =_{B,\sigma} 0. \\
one(N,I) & :\!\!- & N \wedge I =_{B,\sigma} I.
\end{array}$$

In the above $zero(N,I)$ and $one(N,I)$ relations define respectively that element N contains or does not contain minterm I. The rest of the proof is similar to that of the previous lemma. □.

Now we can express a yes/no program Π in Datalog such that deciding whether $\Pi(d)$ is yes for variable database d is double exponential time-hard.

Theorem 3.4 There is a fixed yes/no query program Q in Datalog with Boolean equality constraints such that: If for each input database d, with constant symbols $c_1,\ldots,c_m$, we take the σ that maps these constant symbols to themselves and we interpret B as B_m then deciding whether $Q(d)$ is yes is double exponential time-hard.

Proof: This lower bound is by simulation of deterministic double exponential time bounded Turing machines. We can assume without loss of generality that the alphabet of the input tape is $b_0, b_1, \ldots, b_k$ where b_0 is the special tape symbol # denoting blank and the length of the input tape is m.

We will simulate a $2^{2^m}-1$-time bounded deterministic Turing machine using a Datalog with Boolean constraints over the free Boolean algebra B_m program.

Let the deterministic $2^{2^m}-1$-time bounded Turing machine be $\mathcal{T} = \langle K, \sigma, \delta, s_0, h \rangle$, where K is the set of states of the machine, σ is the alphabet, δ is the transition function, s_0 is the id number of the initial state, and h is the id number of the halting state.

First we use a relation T to describe the initial content of the tape. We record the length of the input tape into the *tape_size* relation:

$$tape\_size(e_m).$$

If $1 \leq i \leq m$ and the ith tape cell is b_j then we create a fact $T(e_i, e_j)$. If $i > m$, then the content of the ith tape cell will be blank. We express this by:

$$\begin{array}{lll} T(M, b_0) & :\!\!- & tape\_size(N), greater(N, M). \\ greater(I, J) & :\!\!- & succ(I, K), greater(K, J). \\ greater(I, J) & :\!\!- & succ(I, J). \end{array}$$

Note that the representation e_i of any number between 0 and m will need only at most $\log m$ minterms. Each minterm can be expressed as the product of the m generators or their complements. Each generator can be denoted in $\log m$ space. Hence each minterm can be expressed in $m \log m$ space and each e_i between 0 and m can be expressed in $m \log^2 m$ space.

Since the alphabet of the tapes are fixed, we can assume that $k < m$. Also note that only the first $2^{2^m} - 1$ tape cells will be given a value. That is enough because the simulation never needs to move beyond the $2^{2^m} - 1$st tape cell due to the time limit. We express the time limit as follows:

$$\begin{array}{lll} time\_bound(X) & :\!\!- & add\_minterm(X, K), last\_minterm(K). \\ add\_minterm(X, J) & :\!\!- & add\_minterm(X, I), next(I, J), X \wedge J =_{B,\sigma} J. \\ add\_minterm(X, I) & :\!\!- & first\_minterm(I), X \wedge I =_{B,\sigma} I. \end{array}$$

Second we use a relation D to describe the transition function δ of $\mathcal{T}$. We create for each possible machine input state s_1, output state s_2, tape symbols c and w, and movement indicator m (being $0, 1$ or 2), a fact $D(e_{s_1}, e_c, e_{s_2}, e_w, e_m)$ if according to δ when the machine is in state s_1 and pointing to c, then either (1) the machine may go to state s_2 and point to symbol w after writing it on the tape and $m = 0$ meaning no move on the tape, or (2) the machine may move one tape cell right and $m = 1$ or to the left and $m = 2$. (Note that if $m = 1$ or $m = 2$ then w can be any tape symbol; we will not use its value.)

Third we use a relation C to describe the configuration of the machine. The relation $C(e_t, e_i, e_s)$ describes that at time step t the machine is pointing to tape position i and is in state s. We can assume that the Turing machine is pointing at time zero to the first tape cell. Therefore we create a fact $C(e_0, e_1, e_{s_0})$.

Fourth we express the sequence of transitions of the machine by a relation $R(e_t, e_j, e_c)$ which is true if and only if at time t the j^{th} tape cell contains the tape symbol c. To initialize R we write the rule: $R(e_0, j, c) :\!\!- T(j, c)$.

We express the requirements for a valid deterministic computation of the machine as follows.

$$\begin{array}{lll}
C(T_2,O,S_2) & :- & succ(T,T_2), m(I,O,M), C(T,I,S_1), R(T,I,C_i), D(S_1,C_i,S_2,W,M) \\
 & & \\
m(I,I,e_0). & & \\
m(I,O,e_1) & :- & succ(I,O). \\
m(I,O,e_2) & :- & succ(O,I). \\
 & & \\
R(T_2,I,W) & :- & succ(T,T_2), C(T,I,S_1), R(T,I,C_i), D(S_1,C_i,S_2,W,e_0) \\
R(T_2,I,C_i) & :- & succ(T,T_2), C(T,I,S_1), R(T,I,C_i), D(S_1,C_i,S_2,W,e_1). \\
R(T_2,I,C_i) & :- & succ(T,T_2), C(T,I,S_1), R(T,I,C_i), D(S_1,C_i,S_2,W,e_2). \\
R(T_2,P,C_p) & :- & succ(T,T_2), C(T,I,S_1), R(T,P,C_p), greater(I,P). \\
R(T_2,P,C_p) & :- & succ(T,T_2), C(T,I,S_1), R(T,P,C_p), greater(P,I). \\
 & & \\
yes & :- & C(T,P,e_h), time\_bound(T_2), greater(T,T_2).
\end{array}$$

The last rule expresses that by time $2^{2^m} - 1$ the machine is in the halting state h. □

The next Corollary follows from Theorem 3.4 and Theorems 3.1, 3.2, 3.3.

Corollary 3.1 Datalog queries with only equality constraints have a double exponential time-complete data complexity. Datalog queries with only constraints in an atomless Boolean algebra and Datalog queries with only precedence and monotone inequality constraints have a double exponential time-hard data complexity and can be evaluted in triple exponential-time data complexity. □

4 Stratified Datalog Queries

4.1 Syntacs and Semantics

Semipositive Programs: We call *semipositive* those Datalog programs that allow negation of input relations [1, 5, 2]. Semantically each *semipositive* Datalog program Π is also a mapping from input databases to interpretations. On any input database d the output of the semipositive Datalog program is $\overline{\Pi}(d)$ where $\overline{\Pi}$ is the Datalog program in which each negated occurrence of an EDB relation R is replaced with the complement of R. The complement of a relation R with arity a when the domain is δ is the relation $\delta^a \setminus R$.

Stratified Datalog Programs: Each *stratified Datalog program* Π is a list of semipositive programs $\Pi_1, \ldots, \Pi_k$ satisfying the following property: no relation symbol R that occurs negated in a Π_i is an IDB in any Π_j with $j \geq i$. Each Π_i is called a stratum of Π.

Semantically each stratified Datalog program is a mapping from databases to interpretations. In particular, if Π is the list of the semipositive programs $\Pi_1, \ldots, \Pi_k$ with the above property, then the composition $\overline{\Pi}_k(\ldots\overline{\Pi}_1()\ldots)$ is the semantics of Π. The output of a stratified Datalog query is called the perfect model of the query [20].

Proposition 4.1 Any Stratified Datalog query Q with equality and inequality constraints over some interpretation (B, σ) where B is a finite Boolean algebra with N elements can be evaluated in $O(N^c)$-time for some constant c.

Proof: The evaluation of a stratified Datalog query that is composed of k semipositive Datalog queries reduces to the evaluation of k Datalog queries followed by the evaluationof the complement of the negated relations. By Proposition 3.1 each Datalog query can be evaluted in $O(N^c)$ time. Let R be any relation with arity a. Suppose that R contains a subset of the tuples in δ^a where δ is the domain of (B, σ). Then the complement of R can be found in $O(N^a \log(N^a))$ time. This is because we can sort the set of tuples in R and also find the set of tuples in order and then test for each whether it is already in the relation or not. Hence each complementation that is needed to go from one stratum to another can be also done

Theorem 5.1. If we have an efficient translation between any Boolean algebra and some Boolean algebra of sets, then the evalution of Datalog queries with $\leq$ constraints in the first can be reduced to an evalution of Datalog queries with $\leq$ constraints in the second. In all cases, the key reason for the time improvement is that the number of distinct minterms that can be built from m atoms is only 2^m, that is, an exponential factor lower than the normal number.

There has been little consideration of Boolean inequality constraints in the literature apart from the paper by Marriott and Odersky [14], and Helm, Marriott and Odersky [8]. Investigating other cases of Boolean inequality constraints that admit simple quantifier elimination remains an important topic.

The decidability of elementary Boolean algebras was proven by Tarski [18] and their complexity was analyzed by Kozen [13]. However, none of [13, 14, 18] considered the computational complexity of evaluating Datalog and Stratified Datalog queries. The known computational complexity results for Datalog and Stratified Datalog queries of other types of constraint databases can be found in [17].

References

[1] S. Abiteboul, R. Hull and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] K.R. Apt, H. Blair, A. Walker. Towards a Theory of Declarative Knowledge. In: *Foundations of Deductive Databases in Logic Programming*, Morgan-Kaufmann, 1988.

[3] S. Burris, H.P. Sankappanavar. *A Course in Universal Algebra*, Springer-Verlag, 1981.

[4] W. Büttner, H. Simonis. Embedding Boolean Expressions into Logic Programming. *Journal of Symbolic Computation*, 4:191–205, 1987.

[5] A.K. Chandra, D. Harel. Structure and Complexity of Relational Queries. *Journal of Computer and System Sciences*, 25:99-128, 1982.

[6] H.B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, 1972.

[7] P. Halmos. *Lectures on Boolean Algebras.* Springer-Verlag, 1974.

[8] R. Helm, K. Marriott, M. Odersky. Constraint-based Query Optimization for Spatial Databases. *Journal of Computer and System Sciences*, vol. 51, no. 2, pp. 197-210, 1995.

[9] J. Jaffar, J.L. Lassez. Constraint Logic Programming. *Proc. 14th ACM Symposium on Principles of Programming Languages*, 111–119, 1987.

[10] J. Jaffar, M.J. Maher. Constraint Logic Programming: A Survey. *J.Logic Programming*, 19 & 20, 503–581, 1994.

[11] P.C. Kanellakis. Constraint Programming and Database Languages: A Tutorial. *Proc. 14th ACM Symposium on Principles of Database Systems*, 46–53, 1995.

[12] P. C. Kanellakis, G. M. Kuper, P. Z. Revesz. Constraint Query Languages. *Journal of Computer and System Sciences*, vol. 51, no. 1, pp. 26-52, 1995.

[13] D. Kozen. Complexity of Boolean Algebras. *Theoretical Computer Science*, 10:221-247, 1980.

[14] K. Marriott, M. Odersky. Negative Boolean Constraints. *Theoretical Computer Science*, 160: 365–380, 1996.

[15] U. Martin, T. Nipkow. Boolean Unification – the Story so Far. *Journal of Symbolic Computation*, 7:275–293, 1989.

[16] P. Z. Revesz. Datalog Queries of Set Constraint Databases, In: *Proc. Fifth International Conference on Database Theory*, Springer-Verlag LNCS 893, pp. 425–438, January 1995.

[17] P. Z. Revesz. Constraint Databases: A Survey. In: *Semantics in Databases*, L. Libkin and B. Thalheim, eds., Springer-Verlag, to appear.

in $O(N^d)$ time for some constant d. Since we have a fixed number of semipositive queries within the stratified Datalog query, the entire evaluation can be done in $O(N^{dk})$ time. □

4.2 Parametric Evaluation of Stratified Datalog Queries

Lemma 4.1 Let Q be any semipositive query with constraints in some Boolean interpretation (B, σ) where B is an *atomless* Boolean algebra. Then Q can be evaluated bottom-up in closed form in triple exponential time data complexity.

Proof: Let v be the maximum arity of any relation in the given query program. Let m be the number of constant symbols occuring in the program and the input database. At first we need to find the complement of each negated relation.

Each negated relation R consists of a set of facts, where the right hand side of each fact is a conjunction of Boolean constraints. Let's take now the disjunction of these right hand sides as a formula ϕ. We put ϕ' again into DNF form. Then each disjunct of ϕ' will be the right hand side of a fact in the complement of R. Since this process does not introduce any new constants and we can delete semantically equivalent constraints, there would be at most $O(2^{2^{2^{c(v+m)}}})$ different facts in the complement of R, and each fact will have at most $O(2^{2^{c(v+m)}}$ size. After the set of facts for the negated relations is found, we can use the parametric evalution for Datalog queries. Hence the rest of the proof follows from Theorem 3.2. □

Theorem 4.1 Let Q be any Stratified Datalog query with constraints in some Boolean interpretation (B, σ) where B is an *atomless* Boolean algebra. Then Q can be evaluated bottom-up in closed form in triple exponential time data complexity.

Proof: By Lemma 4.1 each stratum can be evaluated in $O(2^{2^{2^{c(v+m)}}})$ time. Since no new constants are introduced by the evaluation in any semipositive query, the process can be repeated for each semipositive query, yielding an evaluation of Q in the required time. □

Unfortunately, relations with only equality constraints and relations with only precedence and monotone inequality constraints are not closed under complementation. Hence Stratified Datalog queries with these type of relations are not considered in this section.

5 Comparison with Related Works

Datalog queries with equality constraints were also considered in [12], where the data complexity was proven to be Π_2^p-hard, which is weaker than the double exponential time-complete result of this paper. In [16] Datalog queries with set variables and $\subseteq$ constraints were considered and a deterministic exponential-time complete data complexity was shown. This case can be viewed as a subcase of Datalog with equality constraints under the interpretation of intersection for $\wedge$, union for $\vee$, set complement for negation, and subset-equal for $\leq$, and where each constant represents an atom. The main theorem in [16] can be stated as follows.

Theorem 5.1 Let Q be any Datalog query with precedence constraints in a Boolean interpretation (B, σ) where B is the algebra of sets, that is $\langle \delta, \cap, \cup, ', \emptyset, 1 \rangle$ where the complement is interpreted as set difference from 1, δ is the powerset of all the atoms, and 1 is the set of atoms, and σ maps each constant to an atom. Then Q has a deterministic exponential time-complete data complexity. □

We note that mapping to atoms means mapping to singleton sets in a Boolean algebra of sets. There are other cases when an exponential factor of improvement in the evalution time may be possible when σ is restricted to mappings to atoms. For example, we can use Stone's representation theorem to generalize

[18] A. Tarski. Arithmetical classes and types of Boolean algebras. *Bulletin American Mathematical Society*, 55:1192, 1949.

[19] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, 1989.

[20] J.D. Ullman. *Principles of Database and Knowledge-Base Systems*, Vols 1&2, Computer Science Press, 1989.

[21] M. Vardi. The Complexity of Relational Query Languages. *Proc. 14th ACM Symposium on the Theory of Computing*, 137–145, 1982.