Datalog with Integer Periodicity Constraints

David Toman, Jan Chomicki, and David S. Rogers
Dept. of Comp. and Info. Sci.
Kansas State University, Manhattan, KS–66506
{david,chomicki,rogers}@cis.ksu.edu

Abstract

This paper introduces a generalization of Datalog based on the notion of integer periodicity constraints. A closed form evaluation procedure running in PTIME for this class of constraints is developed. The periodicity constraints are then combined with integer (gap-)order constraints and an evaluation procedure for the combination is developed. A general method for combining different classes of constraints in the framework of Datalog is also discussed.

1 Introduction

Generalized databases [1, 10, 12] are infinite databases that can be represented using finite sets of *generalized* (or *constraint*) tuples. A number of query languages over such databases have been studied. The proposed query languages differ with respect to:

- the underlying inference mechanism (first order vs. deductive)
- the constraint language used.

In this paper we study generalized Datalog programs (function-free logic programs) that operate on constraint tuples in the place of ground atoms. This idea comes from constraint logic programming [9] and allows declarative specification of problems not solvable using the standard Datalog framework (e.g., reasoning about infinite periodic sets of integers).

Constraint tuples are a natural extension of relational tuples:

$R(1,2)$	tuple of arity 2 representing the set $\{(x,y) \mid x = 1 \wedge y = 2\}$
$R(x,y) :-\ x + 5 < y$	generalized tuple $\{(x,y) \mid x + 5 < y\}$ (order constraint)
$R(x,y) :-\ x \equiv_3 (y+1)$	$\{(x,y) \mid 3 \text{ divides } x - (y+1)\}$ (periodicity constraint)

The later two tuples represent infinite sets of integers. Constraint tuples can be combined using conjunction to produce tuples of higher arity. Also, periodicity constraints can be combined with order constraints, e.g.,

$$R(x) :-\ 0 < x, x \equiv_5 3, x < 1000000.$$

Note that this constraint represents the finite set $\{3, 8, \ldots, 999998\}$. Nevertheless the constraint representation is much more compact than the explicit representation of the same set and potentially amenable to a more efficient evaluation.

Applications of both periodicity and order constraints over integers can be found in several areas, including *temporal databases* (where the time component is usually isomorphic to integers). The constraints are used to represent possibly infinite sets of time points [1, 10, 11]. Periodicity and order constraints also serve to formulate general integrity constraints over temporal databases [8].

Example 1.1 A simple example of a Datalog program that uses integer constraints is a database of plane connections between cities. A natural query over such database is "Can I get from Paris to Toronto on Monday starting 8 a.m. or later, but to arrive no later than at 5 p.m.?" The example of such database takes advantage of both periodicity constraints and order constraints.

```
% Database of connections:
connection(paris, london, X, Y):-           % flight every day
        X = Mon10:00(mod 1day),
        Y = Mon11:30(mod 1day).
connection(paris, london, X, Y):-           % and weekdays
        X = Mon14:30(mod 1day),
        Y = X+1:30(mod 1day),
        Z = Mon00:00(mod 7days), W = Fri-23:59(mod 7days),
        Z < X, X < W.
connection(london, toronto, X, Y):-         % twice a week
        X = Tue14:30(mod 7days),            % on Tuesday
        Y = Tue16:50(mod 7days).
connection(london, toronto, X, Y):-         % and
        X = Fri14:30(mod 7days),            % on Friday
        Y = Fri16:50(mod 7days).

% Change times for airports
changetime(london, X, Y) :- X+30<Y.         % 30' to change planes

% Planning of trips:
trip(From, To, Start, End) :-
        connection(From, To, Dtime, Atime),
        Start+15<Dtime,                     % 15' check in
        Atime+20<End.                       % 20' baggage claim
trip(From, To, Start, End) :-
        trip(From, Change, Start, Atime),
        trip(Change, To, Dtime, End),
        changetime(Change,Atime,Dtime).     % specific for airport
```

Note that time units need to be converted to the smallest common unit (i.e., in our example the common unit can be one minute; `1day` is just a shorthand for 1440; similarly `Mon10:00` is 2040, etc. Thus `X = Mon10:00(mod 1day)` is syntactic sugar for $X \equiv_{1440} 2040$). The representation of inherently periodic information (like "flight every day") is very natural using periodicity constraints. The conditions of the type "you need at least 15 minutes to check in" are captured easily using order constraints.

The main contribution of this paper is the definition of a closed form bottom-up evaluation procedure for Datalog programs with integer periodicity and order constraints. This language can be used as a simple but very expressive query language over generalized databases. The evaluation for order constraints is based on [13, 14].

Also a general method for incorporating various classes of constraints into Datalog is studied.

A generally accepted measure of performance of a bottom-up evaluation procedure is *data complexity* [3, 15], which measures the complexity of answering a fixed query w.r.t. the size of the database (the size of the underlying database usually dominates the size of the query by several orders of magnitude). The proposed bottom-up evaluation procedure has *polynomial* data complexity.

There are other proposals that introduce classes of infinite integer relations to query languages. Datalog with (gap-)order constraints was introduced in [13]. In [10, 11], relational calculus over generalized relational databases with order constraints and linear repeating points was studied. (A linear repeating point has the form $\{c + kn | n \in Z\}$ for some fixed integers c and k and is just a different notation for a periodicity constraint.) But the proposed approach was limited to first order queries and thus simple inductive queries like transitive closure were not expressible. It is not clear how to adapt this approach to Datalog because of no obvious termination argument. Moreover, our solution tries to keep the different classes of constraints separate as long as possible (the only 'meeting' points are the quantifier elimination and the consistency checking procedures). This arrangement provides a potentially more general framework which could be adapted to combinations of other constraint languages.

Another approach comes from the area of temporal databases. In [4, 5] Datalog was extended with a limited use of a successor function (the use was restricted to a single distinguished argument in each literal). This made it possible to represent infinite periodic sets of integers using Horn rules. But it is not clear how to add constraints to this language and maintain its computational properties. In particular, order constraints are not expressible in it. Also, the unary successor symbol is used in the deductive layer on top of the database. Our approach allows a representation of infinite relations to be stored directly into the database. Another extension of Datalog was proposed in [2]. This extension combines linear repeating points, order constraints, and unlimited use of successor function. The resulting language is very expressive—termination of query evaluation in this language cannot be guaranteed.

Summarizing: [13, 14] can't handle periodicity constraints, [10, 11] can't handle recursion, [4, 5] can't handle ordering, and [2] does not guarantee termination. Our language is thus another step towards a tractable and expressive query language for temporal databases.

The rest of the paper is organized as follows. Section 2 gives the definition of a finite representation of periodicity constraints together with the operations needed for bottom-up evaluation. Section 3 describes the bottom-up evaluation procedure itself and gives proofs of correctness and termination. Also the complexity of the evaluation procedure is analyzed. Section 4 deals with combining periodicity constraints with order constraints. Again, correctness, termination, and complexity results are presented. Section 5 studies the possibility of combining different classes of constraints in the framework of Datalog. The paper is concluded with a few open problems.

2 Periodicity Graphs

This section describes a graph representation of periodicity (congruence) constraints. The representation together with operations defined over it serve as a basis for closed form bottom-up evaluation of Datalog programs with periodicity constraints.

Definition 2.1 (Periodicity Constraint) *Let K be a finite set of natural numbers. Let $\mathcal{C}_K$ be a set of formulas defined inductively as follows:*

1. $x_i \equiv_k (x_j + c)$ *where x_i, x_j are variables, $k \in K$, and $c \in \{0, \ldots, k-1\}$*
2. $x_i \equiv_k c$ *where x_i is a variable, $k \in K$, and $c \in \{0, \ldots, k-1\}$*
3. *if $C_1, C_2 \in \mathcal{C}_K$ then $C_1 \wedge C_2 \in \mathcal{C}_K$*

Let C be an element of $\mathcal{C}_K$ and let X be the set of free variables in C. Then we call C a periodicity constraint over variables X. *A constraint built only using rules (2) and (3) is called a* simple (periodicity) constraint.

In the context of Datalog we can restrict our attention to the class of *simple* constraints without losing expressive power. This restriction allows more efficient evaluation. However, the general theory can be applied to other logic-based query languages over generalized databases with periodicity constraints where such simplification might not be possible (e.g., conjunctive queries).

Constraints of this form are used in the place of (ground) atoms during bottom-up evaluation. They are directly represented using graphs:

Definition 2.2 (Periodicity Graph) *Let $G = (N_G, E_G)$ be a complete graph over $n+1$ nodes such that $N_G = \{0, x_1, \ldots, x_n\}$ and all edges $e \in E_G$ are labeled with positive integers $\alpha(e)$ and $\beta(e)$ where $0 \leq \beta(e) < \alpha(e)$. We call G a* periodicity graph *over variables $x_1, \ldots, x_n$ (of arity n).*
Let $\alpha(x,y) := \alpha((x,y))$ *and* $\beta(x,y) := \beta((x,y))$ *for $x, y \in N_G$.*

More complex periodicity constraints can be represented as set of such graphs each representing a disjunct of the corresponding DNF.

Definition 2.3 (Assignment) *Let $\vartheta : \{x_1, \ldots, x_n\} \rightarrow Z$ be an assignment of integers to nodes $x_1, \ldots, x_n$. Let $\vartheta(0) = 0$. Let G be a periodicity graph over $x_1, \ldots, x_n$. We say that ϑ satisfies G (we write $\vartheta \models G$) iff*

$$\forall x, y \in N_G : \vartheta(x) \equiv_{\alpha(x,y)} \vartheta(y) + \beta(x,y)$$

We call G satisfiable *if there exists ϑ such that $\vartheta \models G$.*

To combine constraints over the same variables we use the following proposition [7]:

Proposition 2.4 (Chinese Remainder Theorem) *Let $x \equiv_{k_1} (y+c_1)$ and $x \equiv_{k_2} (y+c_2)$. If $\gcd(k_1,k_2)$ divides $|c_1 - c_2|$ then there exists a unique c such that*

$$x \equiv_{k_1} (y+c_1) \wedge x \equiv_{k_2} (y+c_2) \iff x \equiv_{\mathrm{lcm}(k_1,k_2)} (y+c)$$

where $0 \leq c < \mathrm{lcm}(k_1,k_2)$. Otherwise $x \equiv_{k_1} (y+c_1)$ and $x \equiv_{k_2} (y+c_2)$ are inconsistent.

This proposition allows to combine all periodicity constraints over two variables to a single constraint or to show inconsistency of the original constraints.

Definition 2.5 *Let G be a periodicity graph over $x_1, \ldots, x_n$ and C be a periodicity constraint over the same set of variables. We say that G is equivalent to $\mathcal{C}$ if for all assignments ϑ*

$$\vartheta \models C \iff \vartheta \models G$$

Each constraint in $\mathcal{C}_K$ will be shown to be equivalent to a periodicity graph. The graph is defined inductively w.r.t. the structure of the constraint using the following lemmas.

Lemma 2.6 *Let $C = (x_i \equiv_k x_j + c)$ be a periodicity constraint. Let G be a periodicity graph over $\{x_i, x_j\}$ where $\alpha(0, x_i) = \alpha(0, x_j) = 1$, $\beta(0, x_i) = \beta(0, x_j) = 0$, $\alpha(x_i, x_j) = k$, and $\beta(x_i, x_j) = c$. Then G is equivalent to C.*

Lemma 2.7 *Let $C = (x_i \equiv_k c)$ be a periodicity constraint. Let G be a periodicity graph over $\{x_i\}$ where $\alpha(0, x_i) = k$ and $\beta(0, x_i) = c$. Then G is equivalent to C.*

Algorithm 2.8 (Conjunction) *Let G_1 and G_2 be periodicity graphs over set of variables X and Y respectively. We construct a periodicity graph $G = G_1 \wedge G_2$ as follows:*

1. *Let $N_G = N_{G_1} \cup N_{G_2}$ be nodes of the graph G.*
2. *If for any edge $(x_i, x_j) \in E_{G_1} \cap E_{G_2}$ the condition*

   $$\mathrm{lcm}(\alpha_{G_1}(x_i, x_j), \alpha_{G_2}(x_i, x_j)) \textit{ divides } |\beta_{G_1}(x_i, x_j) - \beta_{G_2}(x_i, x_j)|$$

   is violated, then the conjunction of G_1 and G_2 is not satisfiable and no graph is produced.
3. *Otherwise let the edges $(x_i, x_j) \in E_G$ be labeled as follows*

   $$\alpha_G(x_i, x_j) = \begin{cases} \mathrm{lcm}(\alpha_{G_1}(x_i, x_j), \alpha_{G_2}(x_i, x_j)) & (x_i, x_j) \in E_{G_1} \cap E_{G_2} \\ \alpha_{G_1}(x_i, x_j) & (x_i, x_j) \in E_{G_1} - E_{G_2} \\ \alpha_{G_2}(x_i, x_j) & (x_i, x_j) \in E_{G_2} - E_{G_1} \\ 1 & \textit{otherwise} \end{cases}$$

   $$\beta_G(x_i, x_j) = \begin{cases} \gamma_{i,j} & (x_i, x_j) \in E_{G_1} \cap E_{G_2} \\ \beta_{G_1}(x_i, x_j) & (x_i, x_j) \in E_{G_1} - E_{G_2} \\ \beta_{G_2}(x_i, x_j) & (x_i, x_j) \in E_{G_2} - E_{G_1} \\ 0 & \textit{otherwise} \end{cases}$$

   where $0 \leq \gamma_{i,j} < \alpha_G(x_i, x_j)$ is the unique solution of

   $$x_i \equiv_{\alpha_{G_1}(x_i,x_j)} x_j + \beta_{G_1}(x_i, x_j) \wedge x_i \equiv_{\alpha_{G_2}(x_i,x_j)} x_j + \beta_{G_2}(x_i, x_j)$$

   as in Proposition 2.4.

Lemma 2.9 *Let C_1 and C_2 be periodicity constraints over sets of variables X and Y respectively. Let G_1 and G_2 be periodicity graphs equivalent to C_1 and C_2. If $C_1 \wedge C_2$ is satisfiable then there exists periodicity graph $G = G_1 \wedge G_2$ over $X \cup Y$ equivalent to $C_1 \wedge C_2$.*

Theorem 2.10 *Let $C \in \mathcal{C}_K$ be a satisfiable periodicity constraint over $x_1, \ldots, x_n$. Then there exists a periodicity graph G over the same set of variables equivalent to this constraint.*
P r o o f: By induction on the structure of C.

When checking if an assignment of values to any two nodes of a periodicity graph satisfies all the constraints represented by the graph we need to check all paths between these two nodes in general. To avoid this problem we introduce a *normal form* of periodicity graphs where only single edges need to be checked for satisfaction.

Definition 2.11 (Normal Form) *Let G be a periodicity graph. We say that G is in* normal form *if for all $x_i, x_j, x_k \in N_G$*

$$x_i \equiv_{\alpha(x_i,x_j)} (x_j + \beta(x_i, x_j)) \Rightarrow x_i \equiv_{\gcd(\alpha(x_i,x_k),\alpha(x_k,x_j))} (x_j + (\beta(x_i, x_k) + \beta(x_k, x_j)))$$

where $\gcd(x, y)$ is the greatest common divisor of x and y.

Lemma 2.12 *All periodicity graphs in normal form are satisfiable.*
P r o o f: By induction on the size of the graph.

Algorithm 2.13 (Periodicity Graph Normalization) *Let $G = (N_G, E_G)$ be an arbitrary periodicity graph.*

repeat
  for $(x_i, x_j) \in E_G$ do
    for $x_k \in N_G - \{x_i, x_j\}$ do
      $M := \mathrm{lcm}(\alpha_G(x_i, x_j), \mathrm{gcd}(\alpha_G(x_i, x_k), \alpha_G(x_k, x_j))$
      if M divides $|\beta(x_i, x_j) - (\beta(x_i, x_k) + \beta(x_k, x_j))|$
      then
        $\alpha_G(x_i, x_j) := M$
        $\beta_G(x_i, x_j) := \gamma$
      else
        no graph can be produced and the algorithm fails
while $\alpha_G(x_i, x_j)$ changed for some i, j

where γ is the unique solution of the the pair of congruences $x_i \equiv_{\alpha_G(x_i,x_j)} x_j + \beta(x_i, x_j)$ and $x_i \equiv_{\mathrm{gcd}(\alpha_G(x_i,x_k),\alpha_G(x_k,x_j))} x_j + \beta(x_i, x_k) + \beta(x_k, x_j)$ as in Proposition 2.4.

This normalization procedure is also used to detect inconsistent (i.e. not satisfiable) periodicity graphs.

Lemma 2.14 *A periodicity graph is satisfiable if and only if it has an equivalent normal form.*
P r o o f: By induction on the size of the graph.

For closed-form bottom-up evaluation two more operations on the periodicity graphs are needed. The first one is a quantifier elimination procedure:

Algorithm 2.15 (Projection) *Let G be a periodicity graph over a set of variables X. Let $x \in X$. We will construct a periodicity graph $G' = \pi_x(G)$ over the set of variables $X - \{x\}$ as follows:*

1. *Let G'' be normal form of G.*
2. *Let G' be $(N_{G''} - \{x\}, E_{G''} - \{(x, y) | y \in E_{G''}\})$ and let the labeling of nodes and edges be a restriction of labeling of G''.*

Lemma 2.16 *Let C be a satisfiable periodicity constraint over a set of variables X. Let $x \in X$. Let G be a periodicity graph equivalent to C. Then the periodicity graph $G' = \pi_x(G)$ over $X - \{x\}$ is equivalent to $\exists x.C$.*

Note that the 'iff' is preserved because the graph G'' is normalized, i.e., all paths between two nodes are 'implied' by the edge between these two nodes. In the case of *simple* periodicity constraints step 1. in Algorithm 2.15 can be omited.

During bottom-up evaluation of Datalog only "new" tuples need to be added to the constructed model (because of monotonicity of TP). Here, only graphs not implied by already computed graphs are added.

Definition 2.17 (Subsumption) *Let G_1 and G_2 be periodicity graphs over the same set of variables. We say that G_1* subsumes *G_2 if for all assignments $\vartheta : \vartheta \models G_2$ implies $\vartheta \models G_1$.*

Note that the complexity of subsumption does not affect the overall data complexity of the bottom-up evaluation—it depends on the maximal arity of predicates but not on the size of the database.

3 Datalog with periodicity constraints

The previous section introduced a finite representation of periodicity constraints. This representation is used in the bottom-up evaluation of Datalog$^{\equiv Z}$ programs (the periodicity graphs will serve as elements of the interpretation). Datalog$^{\equiv Z}$ programs are defined naturally as follows:

Definition 3.1 (Datalog$^{\equiv Z}$ **program)** *Let P be a finite set of (function-free) clauses of the form*

$$A \leftarrow B_1, \ldots, B_k, C$$

where A and $B_1, \ldots, B_k$ are atoms, and C is a satisfiable periodicity constraint in $\mathcal{C}_K$. Then P is a Datalog$^{\equiv Z}$ *program.*

Note that the usual Datalog constants are not allowed here. This is not a problem; the constants will be added in Section 5.

In a Datalog$^{\equiv Z}$ program (with constraints from $\mathcal{C}_K$) every integer constant can be identified with its remainder class in $Z_{\mathrm{lcm}\,K}$[1]. Thus all operations on numbers are performed in a "modulo" arithmetic.

Definition 3.2 (Named periodicity graph) *Let P be a* Datalog$^{\equiv Z}$ *program. Let R be a predicate symbol that occurs in P of arity k. Let G be a consistent periodicity graph over variables $\{x_1, \ldots, x_k\}$. We call the pair (R, G) a* named periodicity graph.

Named periodicity graphs take the role of ground atoms in the standard bottom-up evaluation of Datalog.

Definition 3.3 ($\mathrm{TP}_{\equiv}$) *Let P be a* Datalog$^{\equiv Z}$ *program. Let I be a set of named periodicity graphs* (an $\equiv$-interpretation)*. We define*

$$\begin{array}{rl}\mathrm{TP}_{\equiv}(I) = \{(A, \pi_{FV(A)}(G)) : & A \leftarrow B_1, \ldots, B_m, C \in P \\ & (B_i, G_i) \in I \textit{ for all } 0 < i \leq m \\ & G \textit{ is the normal form of } G_1 \wedge \ldots \wedge G_m \wedge G_C \\ & (A, \pi_{FV(A)}(G)) \textit{ is not subsumed by any atom in } I\}\end{array}$$

where $FV(A)$ is the set of (free) variables in the atom A. Let $\mathrm{TP}^0_{\equiv}(I) = I$ and $\mathrm{TP}^i_{\equiv}(I) = \mathrm{TP}_{\equiv}(\mathrm{TP}^{i-1}_{\equiv}(I))$ for $i > 0$.

Note that the nodes of the periodicity graphs are "renamed" using the variable names in the associated subgoals of the clause; the conjunction operation is performed w.r.t. those names (similarly to natural join).

Definition 3.4 (Upper bound) *Let P be a* Datalog$^{\equiv Z}$ *program and $\{C_1, \ldots, C_n\}$ be a set of all constraints in P. Let k_i be the modulo factor in the constraint C_i. Then we say that $\alpha(P) = \mathrm{lcm}\{k_i | 0 < i \leq n\}$ is the maximal modulo factor for P.*

We show that for a given Datalog$^{\equiv Z}$ program (i.e., w.r.t. a fixed $\mathcal{C}_K$) the labels of all the edges of periodicity graphs are bounded by a constant. This will be shown for periodicity constraints first:

Lemma 3.5 *Let C be a satisfiable periodicity constraint that occurs in a* Datalog$^{\equiv Z}$ *program P. Let G_C be a periodicity graph equivalent to C. Then for all $e \in E_G$: $\alpha_G(e) \leq \alpha(P)$.*

P r o o f: By structural induction on C.

[1] Z_n is an (additive) cyclic group with n elements.

The application of TP also preserves this bound:

Lemma 3.6 *Let P be a* Datalog$^{\equiv Z}$ *program. Let $G \in \mathrm{TP}^i_{\equiv}(\emptyset)$ Then for all $e \in E_G : \alpha(e) \leq \alpha(P)$.*
P r o o f: By induction on i.

Algorithm 3.7 (Bottom-up Evaluation of Datalog$^{\equiv Z}$ **Programs)** *Let P be a* Datalog$^{\equiv Z}$ *program and let I, J be $\equiv$-interpretations of P.*

> $I := \emptyset$
> *repeat*
> $J, I := I, \mathrm{TP}_{\equiv}(I)$
> *while* $J \neq I$

Algorithm 3.7 produces the same result as the standard bottom-up evaluation of the same program:

Theorem 3.8 *For any* Datalog$^{\equiv Z}$ *program P and any predicate symbol R in P*

$$R(c_1, \ldots, c_k) \in \mathrm{TP}^{\omega}(\emptyset) \iff (R, G) \in \mathrm{TP}^{\omega}_{\equiv}(\emptyset) \wedge [c_1/x_1, \ldots, c_k/x_k] \models G$$

where TP *is the standard consequence operator on Herbrand interpretations.*
P r o o f: By stage induction and continuity of TP and $\mathrm{TP}_{\equiv}$.

Note that $\mathrm{TP}^{\omega}(\emptyset)$ may be an infinite set. But the $\mathrm{TP}^{\omega}_{\equiv}(\emptyset)$ is always finite.

Theorem 3.9 *Algorithm 3.7 terminates for any* Datalog$^{\equiv Z}$ *program P.*
P r o o f: Immediate from Lemma 3.6.

3.1 Complexity of Bottom-up Evaluation for Datalog$^{\equiv Z}$

The number of applications of the $\mathrm{TP}_{\equiv}$ operator in the Algorithm 3.7 can be bounded for each Datalog$^{\equiv Z}$ program as follows:

Lemma 3.10 *Let P be a* Datalog$^{\equiv Z}$ *program and let R be a predicate symbol in P. Let a be the arity of R. Then the number of different elements of $\mathrm{TP}^{\omega}_{\equiv}(\emptyset)$ of the form (R, G) is at most $\alpha(P)^{a(a+1)}$*
P r o o f: For every edge e in G we have only $\alpha(P)$ different labels $\alpha(e)$ by Lemma 3.6 and also only $\alpha(P)$ different labels $\beta(e)$ by definition of the periodicity graph. All the graph components of the elements $(R, G) \in \mathrm{TP}^{\omega}_{\equiv}(\emptyset)$ have exactly $\frac{a(a+1)}{2}$ edges.

The actual number of graphs in the model will usually be smaller due to subsumption checking (Lemma 2.17). For *simple* constraints the bound is $\alpha(P)^{2a}$ (there are only a nontrivial edges in the periodicity graph).

Corollary 3.11 *Let P be a* Datalog$^{\equiv Z}$ *program where all the atomic constraints are from $\mathcal{C}_K$. Let a be the maximal arity of an atom in P, v be the number of distinct variables in a single clause of P, p be number of predicate symbols in P, and n number of clauses in P (including facts). Let m be the maximal number of goals in any clause in P. Then Algorithm 3.7 terminates in $O(n)$ steps.*
P r o o f: Application of the $\mathrm{TP}_{\equiv}$ takes at most $mO(v^2)+O(v^4)$ time for each clause in P. Because there are at most $p\alpha(P)^{a(a+1)}$ different elements in the $\mathrm{TP}^{\omega}_{\equiv}(\emptyset)$ (by Lemma 3.10) the total time needed to compute the model is at most:

$$n \cdot \left[\left(p\alpha(P)^{a(a+1)} \right)^m \cdot \left(mO(v^2) + O(v^4) \right) \right] \cdot p\alpha(P)^{a(a+1)} = O(n)$$

because a, m, p, v, and $\alpha(P)$ are constants for a given P w.r.t. $\mathcal{C}_K$.

The restriction of the modulo factors to a fixed set is crucial to achieve the data complexity bound. For *simple* constraints the application of $\mathrm{TP}_{\equiv}$ takes only $mO(v)$ time.

Also testing if a standard relational tuple belongs to the computed model can be done in PTIME:

Theorem 3.12 (Tuple recognition) *Let P be a* Datalog$^{\equiv Z}$ *program where all the atomic constraints are from $\mathcal{C}_K$. Let $A(c_1, \ldots, c_n)$ be a ground atom. Then $P \models A$ is decidable in $O(n)$.*

4 Combining Classes of Constraints over Integers

The periodicity constraints over integers are clearly not the only interesting class of constraints. In this section we consider the possibilities of combining different classes of constraints over the same domain (in our case integers).

4.1 Gap-order constraints

First we combine the constraint language developed so far with Datalog$^{<Z}$ (Datalog with gap-order constraints) [13, 14].

Definition 4.1 (Gap-order constraint) *Let u, l be integers, c be a nonnegative integer, and $x, y, \ldots$ be variables over integers. Then a finite conjunction of formulas of the form $l < x$, $x < u$, and $x + c < y$ is called a* gap-order constraint.

Conjunctions of gap-order constraints can be efficiently represented using *gap graphs* [13] (directed acyclic graphs where the nodes represent the variables and the lower and the upper bound of the constraints, and the edges represent the gaps, i.e., the minimal integer distances between the nodes). In this representation we also have all the needed operations for closed form bottom-up evaluation.

We show that combination of these two approaches still has closed form evaluation and moreover the complexity bound does not increase.

Definition 4.2 (Datalog$^{\equiv Z, <Z}$ **programs)** *Let P be a finite set of clauses of the form*

$$A \leftarrow B_1, \ldots, B_k, C_1, C_2$$

where $A, B_1, \ldots, B_k$ are atoms, C_1 is a satisfiable periodicity constraint in $\mathcal{C}_K$, and C_2 is a satisfiable gap-order constraint. Then P is a Datalog$^{\equiv Z, <Z}$ *program.*

4.1.1 Complexity of consistency checking

Consistency (satisfiability) checking of periodicity graphs or gap graphs (separately) can be done in polynomial time (w.r.t. the size of the graph). However

Theorem 4.3 *Consistency checking of a conjunction of periodicity graph with a gap graph is NP-complete.*

P r o o f: By reduction of 3SAT to satisfiability of conjunctions of constraints of the form $x \equiv_k (y + c)$ and $x + c < y$.

Fortunately

Theorem 4.4 *Consistency checking of a conjunction of a periodicity graph constructed only from simple constraints and a gap graph can be computed in PTIME.*
P r o o f: By generalization of the consistency checking procedure given in [13]; the algorithm computes the length of every path from the lower to the upper bound of the gap-graph by adding the gap sizes on the edges of the path. Moreover for every node on the path the length of the partial path from the lower bound to this node is rounded w.r.t. the periodicity constraint associated with this node. The graph is consistent iff the length of every path is less than the difference between the lower and upper bounds.

Limiting the congruence constraints to the form $x \equiv_k c$ does not reduce the expressive power of the language. All the constraints of the form $x \equiv_k (y + c)$ can be defined using the following Datalog$^{\equiv Z}$ clauses:

$$\begin{array}{lcll}
x \equiv_k (y+c) & :- & x \equiv_k 0 & \wedge \quad y \equiv_k c \\
x \equiv_k (y+c) & :- & x \equiv_k 1 & \wedge \quad y \equiv_k (c+1) \\
 & \vdots & & \\
x \equiv_k (y+c) & :- & x \equiv_k (k-1) & \wedge \quad y \equiv_k (c-1)
\end{array}$$

In similar way we can also represent constraints of the form $x \not\equiv_k y + c$. The periodicity graphs become much simpler (and smaller) in this limited case; especially the normalization is not needed any more. The disadvantage of this solution is that we need to store more facts in the database.

Moreover, the overall complexity of bottom-up evaluation is exponential in the *arity* of predicate symbols in the program. Thus even using a NP-complete consistency checking procedure doesn't have an impact on the overall data complexity. We shall consider only simple periodicity constraints in the rest of the paper.

4.1.2 Bottom-up evaluation

A $(\equiv, <)$-interpretation is a set of triples (A, G, H) where A is a predicate symbol, G is a periodicity graph, and H is a gap graph respectively. A, G, and H must have the same arity.

In the definition of the consequence operator $\mathrm{TP}_{\equiv,<}$ we can reuse the conjunction and subsumption procedures defined for the component classes of constraints. We also use the consistency check from Theorem 4.4. But the most complex operation—the quantifier elimination—has to take care of the interactions of periodicity constraints with gap-order constraints [16].

Example 4.5 Assume, that we want to eliminate quantifier from the following constraint:

$$\exists y. x + c_1 < y \wedge y + c_2 < z \wedge y \equiv_k d$$

Clearly we can't replace it simply by $x + c_1 + c_2 + 1 < z$ as in the case of gap-order constraints only: we need to take into account the congruence $y \equiv_k d$, i.e., we need to make sure that there is at least one integer of the form $\{d + nk\}$ between $x + c_1$ and $z - c_2$. Thus the equivalent reformulation is

$$\begin{array}{lcll}
(x + c_1 + 1) \equiv_k d & \wedge & x + c_1 + c_2 + 1 < z & \vee \\
(x + c_1 + 2) \equiv_k d & \wedge & x + c_1 + c_2 + 2 < z & \vee \\
 & \vdots & & \\
(x + c_1 + k) \equiv_k d & \wedge & x + c_1 + c_2 + k < z &
\end{array}$$

It is easy to see, that the variable y was successfully eliminated and the resulting constraint is a disjunction of conjunctions of periodicity and gap-order constraints.

This idea is used for the projection operation needed in the $\mathrm{TP}_{\equiv,<}$ operator (a similar idea was used in [10, 11]; our procedure is simpler due to a different representation of constraints, especially we can omit normalization (in the sense of [10, 11]) of constraint tuples).

Algorithm 4.6 (Projection) Let G be a periodicity graph and H be a gap-graph. Let y be a node in both G and H. Let $x \in N_H$ such that $(x, y) \in E_H$ is labeled with c (i.e., $x + c < y$) and let $z_1, \ldots, z_l \in N_H$ such that $(y, z_j) \in E_H$ is labeled with c_j for $0 < j \leq l$. Let $y \equiv_k d \in G$. We form graphs (G_i, H_i) for $0 < i \leq k$ by modifying G and H as follows:

1. delete the edge (x, y)
2. insert edges (x, z_j) with the label $c + c_j + i$
3. add the constraint $x \equiv_k (d - c - i)$
4. check the resulting pair (G_i, H_i) for consistency and discard inconsistent pairs.

We apply this transformation on each consistent (G_i, H_i) until all edges ending in y are eliminated from the gap graphs. Then we remove the node y from all resulting pairs of graphs together with all incident edges.

Lemma 4.7 *Let $\{(G_i, H_i) | i \in I\}$ be the set of graphs produced by Algorithm 4.6 from the pair (G, H) w.r.t. a variable y. Then for all valuations θ:*

$$\theta \models \bigvee_{i \in I} (G_i, H_i) \iff \exists a \in Z.\theta[a/y] \models (G, H)$$

Thus Algorithm 4.6 defines a projection operation for periodicity graphs combined with gap graphs. To eliminate all existential quantifiers we just apply this algorithm recursively to its own output until all existential quantifiers are eliminated.

The projection operation produces (possibly fairly large) set of graphs from a single graph. This is often considered to be a (space) complexity problem. But in the framework of Datalog it is not true: even in standard Datalog each relation is represented as a collection of its tuples. In our case we still manage to represent (often infinite) sets of tuples by a single element. The consequence operator is now defined in the standard way:

Definition 4.8 *Let P be a* Datalog$^{\equiv Z, <Z}$ *program.*

$$\begin{aligned} \mathrm{TP}_{\equiv,<}(I) = \{\ & (A, G, H) : A \leftarrow B_1, \ldots, B_m, C_1, C_2 \in P \\ & (B_i, G_i, H_i) \in I \textit{ for all } 0 < i \leq m \\ & G' = G_1 \wedge \ldots \wedge G_m \wedge G_{C_1} \\ & H' = H_1 \wedge \ldots \wedge H_m \wedge H_{C_2} \\ & (G, H) \in \pi_{FV(A)}(G', H') \textit{ is consistent and not subsumed by } I\} \end{aligned}$$

where G_{C_1} is a periodicity graph corresponding to C_1 and H_{C_2} is a gap graph corresponding to C_2.

We can show that bottom-up evaluation of an Datalog$^{\equiv Z, <Z}$ program using Algorithm 3.7 (using $\mathrm{TP}_{\equiv,<}$ instead $\mathrm{TP}_{\equiv}$) is correct w.r.t. the standard bottom-up evaluation.

Theorem 4.9 *For any* Datalog$^{\equiv Z, <Z}$ *program P and any predicate symbol R in P*

$$R(c_1, \ldots, c_k) \in \mathrm{TP}^{\omega}(\emptyset) \iff (R, G) \in \mathrm{TP}^{\omega}_{\equiv,<}(\emptyset) \wedge [c_1/x_1, \ldots, c_k/x_k] \models G$$

P r o o f: By simultaneous induction on the number of iterations of TP.

Let I be a $(\equiv,<)$-interpretation. Then $\pi_{\equiv}(I) = \{(A,G) : (A,G,H) \in I\}$ and $\pi_{<}(I) = \{(A,H) : (A,G,H) \in I\}$.

Lemma 4.10 *Let P be a* Datalog$^{\equiv Z,<Z}$ *program. Then*

$$\pi_{\equiv}(\mathrm{TP}^i_{\equiv,<}(\emptyset)) \subseteq \mathrm{TP}^i_{\equiv}(\emptyset) \text{ and } \pi_{<}(\mathrm{TP}^i_{\equiv,<}(\emptyset)) \subseteq \mathrm{TP}^i_{<}(\emptyset)$$

for all $0 \leq i$.

Corollary 4.11 *The bottom-up evaluation of any* Datalog$^{\equiv Z,<Z}$ *program terminates.*
Proof: Immediate from Lemma 4.10 and termination of the bottom-up evaluation procedure for Datalog$^{<Z}$ [13].

To show polynomial tuple recognition procedure for Datalog$^{\equiv Z,<Z}$ programs we use the same technique: the tuple recognition will then run in PTIME (follows from Theorem 3.1 and complexity of the TEST algorithm in [13])[2].

4.2 Equality constraints

So far we haven't mentioned the most natural class of constraints—the equality constraints, which are needed to include the "standard" Datalog over integers. In [13] the equality constraints are handled by adding extra information to the gap-graph. In the case of Datalog$^{\equiv Z}$ it can be done the same way (in all cases we need some mechanism to represent a single constraint of the form $x = y$).

For constraint classes closed under conjunction we can use following lemma:

Lemma 4.12 *Let φ be a constraint where x and y are free. Then*

$$\varphi \wedge (x = y) \iff \varphi \wedge \varphi[x/y, y/x]$$

After applying this lemma we can eliminate either of x and y by the quantifier elimination procedure described in the previous section (if $x = y$ is the only constraint we can just ignore it).

5 Classes of Constraints with Disjoint Domains

The previous section suggests that we can integrate various classes of constraints into Datalog. The main difficulty of doing so is the design of the quantifier elimination procedure (projection). This becomes much simpler when we combine classes of constraints with disjoint domains (like integers and uninterpreted constants). Then because of the 'constraint' approach, the evaluation of each of the classes is independent of the other classes. The only meeting point is consistency checking or tuple membership checking (the tuple membership checking may be considerably less complex than consistency checking).

Definition 5.1 (Datalog$^{\mathcal{S}_1,\ldots,\mathcal{S}_n}$) *Let $\mathcal{S}_i$ be classes of constraints for $0 < i \leq n$. We say that finite set of clauses of the form*

$$A \leftarrow B_1, \ldots, B_m, C_1, \ldots, C_n$$

where A and $B_1, \ldots, B_m$ are predicate symbols, and C_i is a satisfiable constraint in $\mathcal{S}_i$. Then P is a Datalog$^{\mathcal{S}_1,\ldots,\mathcal{S}_n}$ program.

[2]The degree of the polynomial is higher than in standard Datalog—$O(a^2)$ for atoms of arity a (exactly $(a+1)(a+2)/2$ for Datalog$^{<Z}$ and $2a+2$ for Datalog$^{\equiv Z}$). This is because the values are assigned to *pairs* of attributes rather than to single attributes. In both cases the *exponentiated* expression is a constant w.r.t. the size of the program.

Note that each of the classes $\mathcal{S}_i$ contains the trivial constraint *true*.

Definition 5.2 *Let $\mathcal{S}$ be a class of constraints. Let* $\mathrm{TP}_{\mathcal{S}}$ *be the consequence operator for Datalog$^{\mathcal{S}}$. Let P be a Datalog$^{\mathcal{S}}$ program and C a clause in P. The* relativized consequence operator *is defined as* $\mathrm{TP}_{\mathcal{S}}$ *with respect to the program* $\{C\}$ *and denoted* $\mathrm{TP}_{\mathcal{S}}^{C}$.

A $(\mathcal{S}_1, \ldots, \mathcal{S}_n)$-interpretation is a set of $(n+1)$-tuples $(A, G_1, \ldots, G_n)$ where A is a predicate symbol and G_i are finite representation of a constraint in $\mathcal{C}_i$. All G_i's have the same arity as A.

Definition 5.3 *Let P be a Datalog$^{\mathcal{S}_1,\ldots,\mathcal{S}_n}$ program, such that domains of $\mathcal{S}_i$ and $\mathcal{S}_j$ are disjoint for all $i \neq j$. Let I be a $(\mathcal{S}_1, \ldots, \mathcal{S}_n)$-interpretation. Let* $\pi_{\mathcal{S}_j}(I) = \{(A, G_j) : (A, G_1, \ldots, G_n) \in I\}$. *Then let*

$$\mathrm{TP}_{\mathcal{S}_1,\ldots,\mathcal{S}_n}(I) = \{(A, G_1, \ldots, G_n) : \begin{array}{l} A \leftarrow B_1, \ldots, B_m, C_1, \ldots, C_n \in P \\ G_i = \mathrm{TP}_{\mathcal{S}_i}^{A \leftarrow B_1,\ldots,B_m,C_i}(\pi_{\mathcal{S}_i}(I)) \text{ exists } \} \end{array}$$

Note that G_i must exist for each $0 < i \leq n$. This may not always be the case (e.g., in the case of Datalog$^{\equiv Z, <Z}$ no atom is produced if the resulting gap-graph is inconsistent).

We use this definition to add the standard domain of "uninterpreted constants" to our framework of constraints.

Example 5.4 Let D be a flat domain of "uninterpreted constants" equipped with equality only. Then we can define language Datalog$^{=D}$ as follows:

1. a $=$-interpretation is a set of tuples (A, R_A) where A is a predicate symbol and R_A is a A-tuple over D.
2. a conjunction of two atoms (A, R_A) and (B, R_B) is a tuple where fields of R_A are matched against fields of R_B with the same name,
3. a conjunction of atoms is consistent if the matching is successful,
4. a projection operation is simple removing a field from the tuple, and
5. (A, R_A) subsumes (A, S_A) iff $S_A = R_A$

Both the $\mathrm{TP}_{=D}$ operator and the bottom-up evaluation procedure follow from Definition 5.3.

The bottom-up evaluation of Datalog$^{=D}$ program is exactly the same as in standard Datalog. But this is not the only way of thinking about bottom-up evaluation in Datalog$^{=D}$. We can also choose the $=$-interpretation to be a set of finite relations over D. Then conjunction becomes the natural join, projection becomes the relational algebra projection, and subsumption becomes the subset relation. Also consistency checking is easy: we simply look only for nonempty relations.

Now we can use Definition 5.3 to define the language Datalog$^{=D,<Z,\equiv Z}$. The result is the language developed in the previous section enriched by the standard domain of uninterpreted constants. The evaluation procedure is again the standard bottom-up evaluation algorithm.

Note that we don't need a specialized consistency checking procedure over all the conjunctions of $= D$, $< Z$, and $\equiv Z$ constraints. The reason is that the domains D and Z are disjoint and thus we can check the consistency separately.

The correctness and termination for the bottom-up evaluation of Datalog$^{\mathcal{S}_1,\ldots,\mathcal{S}_n}$ is proven similarly to the Datalog$^{\equiv Z,<Z}$ case:

Theorem 5.5 *For a Datalog$^{\mathcal{S}_1,\ldots,\mathcal{S}_n}$ program P and a predicate symbol R in P*

$$R(c_1,\ldots,c_k) \in \mathrm{TP}^\omega(\emptyset) \iff \begin{array}{l}(R, G_1,\ldots,G_n) \in \mathrm{TP}^\omega_{\mathcal{S}_1,\ldots,\mathcal{S}_n}(\emptyset) \\ \wedge (\forall i.[c_1/x_1,\ldots,c_k/x_k] \models G_i)\end{array}$$

Lemma 5.6 *Let P be a Datalog$^{\mathcal{S}_1,\ldots,\mathcal{S}_n}$ program. Then $\pi_{\mathcal{S}_j}(\mathrm{TP}^i_{\mathcal{S}_1,\ldots,\mathcal{S}_n}(\emptyset)) \subseteq \mathrm{TP}^i_{\mathcal{S}_j}(\emptyset)$ for all $0 < j \leq n$ and $0 \leq i$.*

Note that the $\mathrm{TP}^i_{\mathcal{S}_j}$ operator is applied on clauses of P projected to Datalog$^{\mathcal{S}_j}$ (i.e., all constraint goals except C_j are deleted from the clauses of P).

Corollary 5.7 *Let $\mathrm{TP}^\omega_{\mathcal{S}_j}$ be finite for every $0 < j \leq n$. Then the $\mathrm{TP}^\omega_{\mathcal{S}_1,\ldots,\mathcal{S}_n}(\emptyset)$ is finite as well and thus bottom-up evaluation of Datalog$^{\mathcal{S}_1,\ldots,\mathcal{S}_n}$ programs terminates.*

We can use the standard bottom-up evaluation algorithm whose termination is guaranteed. Moreover the product of the cardinalities of the $\mathrm{TP}^\omega_{\mathcal{S}_j}$ sets bounds the number of TP applications needed to reach the fixed point. Especially, if all the sets are polynomial in the size of the program then we have PTIME evaluation procedure. The tuple recognition procedure for Datalog$^{\mathcal{S}_1,\ldots,\mathcal{S}_n}$ can be based on the combination of tuple recognition procedures for Datalog$^{\mathcal{S}_j}$ in similar way as in Definition 5.3.

6 Conclusion

We have described bottom-up evaluation procedures for several versions of Datalog enhanced with constraints and provided general way for combining various classes of constraints into a single language. We also provided complexity bounds for the evaluation algorithms.

6.1 Open Problems

Complexity of satisfiability. The satisfiability check for conjunctions of constraints of the form $x + c < y$, $x \equiv_k (y + c)$, and $x \equiv_k c$ is NP-complete. On the other hand satisfiability of conjunctions of $x + c < y$ and $x \equiv_k c$ can be done in PTIME. What is the complexity of consistency checking for constraints of the form $x + c < y$, $x \equiv_k y$, and $x \equiv_k c$? (The NP-completeness proof for the first case utilizes the $x \equiv_k (y + c)$ type of constraint). Note that in all three cases the expressive power of generalized Datalog is the same. Also, the data complexity of the satisfiability check is bounded by constant (maximal arity) for any program.

Extensions: negation, negative gap-order constraints. The gap-order constraints allow to specify the minimal distance between two variables. The maximal distance constraint (like maximal waiting time for changing planes in Example 1.1) can be expressed in a very limited way using an enumeration technique similar to Algorithm 4.6. Note that a general solution that adds gap order constraints with negative size of the gap (i.e., $x - 2 < y$) leads immediately to an undecidable theory—we can define the successor relation over integers as "$\mathrm{succ}(x, y) :- \ y - 2 < x, x < y$".

For the same reason it is not possible to extend the language with standard stratified negation. Is there a limited version of negation that can be included in the language without this penalty? Note that for periodicity constraints alone this is not a problem: the constraint of the form $x \not\equiv_k y + c$ can be introduced without any problems.

Expressiveness. In [1] the expressiveness of a number of deductive and constraint query languages is discussed. However, only monadic programs are considered. It is interesting to see whether the expressiveness of query languages defined in this paper can be also formally characterized.

Efficient Implementation. Periodicity constraints define non-convex sets. In this respect they differ from most common constraint languages. [12] describe how to adapt interval management techniques to constraint databases. But this works only for convex sets. Thus periodicity constraints call for new storage management techniques.

References

[1] Baudinet, M., Chomicki, J., Wolper, P. Temporal Deductive Databases. In *Temporal Databases: Theory, Design, And Implementation*, Tansel, A. et al. (editors). Benjamin Cummings 1993.

[2] Baudinet, M., Níezette, M., Wolper P. On the Representation of Infinite Temporal Data and Queries. *10th ACM Symposium on Principles of Database Systems,* pp 280–290, 1991.

[3] Chandra, A. K. Structure and Complexity of Relational Queries. *Journal of Computer and System Sciences,* 25:156–178, 1982.

[4] Chomicki, J., Imieliński, T. Temporal Deductive Databases and Infinite Objects. Proc. *7th ACM Symposium on Principles of Database Systems,* pp 61–73, 1988.

[5] Chomicki, J., Imieliński, T. Finite Representation of Infinite Query Answers, In *ACM Transactions on Database Systems*, pp. 181–223, (18) 2, 1993.

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

[7] Hardy, G. H. An Introduction to the Theory of Numbers. Oxford University Press, 1979.

[8] Jensen, C. S., Snodgrass, R. Temporal Specialization and Generalization. *IEEE Transactions on Knowledge and Data Engineering*, 1993.

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

[10] Kabanza, F., Stevenne, J-M., Wolper, P. Handling Infinite Temporal Data. In *9th ACM Symposium on Principles of Database Systems,* 1990.

[11] Kabanza, F., Stevenne, J-M., Wolper, P. Handling Infinite Temporal Data. *Journal of Computer and System Sciences*, 1994 (in print).

[12] Kanellakis, P. C., Kuper, G. M., Revesz, P.Z. Constraint Query Languages, Proc. *9th ACM Symposium on Principles of Database Systems,* pp 299–313, 1990.

[13] Revesz, P. Z. A Closed Form for Datalog Queries with Integer Order. Proc. *3rd International Conference on Database Theory*, pp 187–201, Springer-Verlag LNCS 470, 1990.

[14] Revesz, P. Z. A Closed Form Evaluation for Datalog Queries with Integer (Gap)-Order Constraints. *Theoretical Computer Science*, vol. 116, no. 1, 117–149, 1993.

[15] Vardi, M. Y. The Complexity of Relational Query Languages. In *ACM SIGACT Symposium on Theory of Computing,* pp 137–146, 1982.

[16] Williams, H. P. Fourier–Motzkin Elimination Extension to Integer Programming Problems. In *Journal of Combinatorial Theory* (A) 21, pp 118-123, 1976.