Foundations of Declarative Data Analysis Using Limit Datalog Programs

Mark Kaminski, Bernardo Cuenca Grau, Egor V. Kostylev, Boris Motik and **Ian Horrocks**
Department of Computer Science, University of Oxford, UK
{mark.kaminski, bernardo.cuenca.grau, egor.kostylev, boris.motik, ian.horrocks}@cs.ox.ac.uk

Abstract

Motivated by applications in declarative data analysis, we study $Datalog_{\mathbb{Z}}$—an extension of positive Datalog with arithmetic functions over integers. This language is known to be undecidable, so we propose two fragments. In *limit* $Datalog_{\mathbb{Z}}$ predicates are axiomatised to keep minimal/maximal numeric values, allowing us to show that fact entailment is coNEXPTIME-complete in combined, and coNP-complete in data complexity. Moreover, an additional *stability* requirement causes the complexity to drop to EXPTIME and PTIME, respectively. Finally, we show that stable $Datalog_{\mathbb{Z}}$ can express many useful data analysis tasks, and so our results provide a sound foundation for the development of advanced information systems.

1 Introduction

Analysing complex datasets is currently a hot topic in information systems. The term 'data analysis' covers a broad range of techniques that often involve tasks such as data aggregation, property verification, or query answering. Such tasks are currently often solved imperatively (e.g., using Java or Scala) by specifying *how* to manipulate the data, and this is undesirable because the objective of the analysis is often obscured by evaluation concerns. It has recently been argued that data analysis should be *declarative* [Alvaro *et al.*, 2010; Markl, 2014; Seo *et al.*, 2015; Shkapsky *et al.*, 2016]: users should describe *what* the desired output is, rather than how to compute it. For example, instead of computing shortest paths in a graph by a concrete algorithm, one should (i) describe what a path length is, and (ii) select only paths of minimum length. Such a specification is independent of evaluation details, allowing analysts to focus on the task at hand. An evaluation strategy can be chosen later, and general parallel and/or incremental evaluation algorithms can be reused 'for free'.

An essential ingredient of declarative data analysis is an efficient language that can capture the relevant tasks, and Datalog is a prime candidate since it supports recursion. Apart from recursion, however, data analysis usually also requires integer arithmetic to capture quantitative aspects of data (e.g., the length of a shortest path). Research on combining the two dates back to the '90s [Mumick *et al.*, 1990; Kemp and Stuckey, 1991; Beeri *et al.*, 1991; Van Gelder, 1992; Consens and Mendelzon, 1993; Ganguly *et al.*, 1995; Ross and Sagiv, 1997], and is currently experiencing a revival [Faber *et al.*, 2011; Mazuran *et al.*, 2013]. This extensive body of work, however, focuses primarily on integrating recursion and arithmetic with *aggregate functions* in a coherent semantic framework, where technical difficulties arise due to nonmonotonicity of aggregates. Surprisingly little is known about the computational properties of integrating recursion with arithmetic, apart from that a straightforward combination is undecidable [Dantsin *et al.*, 2001]. Undecidability also carries over to the above formalisms and practical Datalog-based systems such as BOOM [Alvaro *et al.*, 2010], DeALS [Shkapsky *et al.*, 2016], Myria [Wang *et al.*, 2015], SociaLite [Seo *et al.*, 2015], Overlog [Loo *et al.*, 2009], Dyna [Eisner and Filardo, 2011], and Yedalog [Chin *et al.*, 2015].

To develop a sound foundation for Datalog-based declarative data analysis, we study $Datalog_{\mathbb{Z}}$—negation-free Datalog with integer arithmetic and comparisons. Our main contribution is a new *limit* $Datalog_{\mathbb{Z}}$ fragment that, like the existing data analysis languages, is powerful and flexible enough to naturally capture many important analysis tasks. However, unlike $Datalog_{\mathbb{Z}}$ and the existing languages, reasoning with limit programs is decidable, and it becomes tractable in data complexity under an additional *stability* restriction.

In limit $Datalog_{\mathbb{Z}}$, all intensional predicates with a numeric argument are *limit predicates*. Instead of keeping all numeric values for a given tuple of objects, such predicates keep only the minimal (min) or only the maximal (max) bounds of numeric values entailed for the tuple. For example, if we encode a weighted directed graph using a ternary predicate *edge*, then rules (1) and (2), where *sp* is a min limit predicate, compute the cost of a shortest path from a given source node v_0 to every other node.

$$\to sp(v_0, 0) \tag{1}$$

$$sp(x, m) \wedge edge(x, y, n) \to sp(y, m + n) \tag{2}$$

If these rules and a dataset entail a fact $sp(v, k)$, then the cost of a shortest path from v_0 to v is at most k; hence, $sp(v, k')$ holds for each $k' \geq k$ since the cost of a shortest path is also at most k'. Rule (2) intuitively says that, if x is reachable from v_0 with cost at most m and $\langle x, y\rangle$ is an edge of cost n, then v' is reachable from v_0 with cost at most $m + n$. This is different from $Datalog_{\mathbb{Z}}$, where there is no implicit semantic connection between $sp(v, k)$ and $sp(v, k')$, and such semantic connections allow us to prove decidability of limit $Datalog_{\mathbb{Z}}$. We provide a direct semantics for limit predicates based on Herbrand interpretations, but we also show that this semantics can be axiomatised in standard $Datalog_{\mathbb{Z}}$. Our formalism can thus be seen as a fragment of $Datalog_{\mathbb{Z}}$, from which it inherits well-understood properties such as monotonicity and existence of a least fixpoint model [Dantsin *et al.*, 2001].

Our contributions are as follows. First, we introduce limit $Datalog_{\mathbb{Z}}$ programs and argue that they can naturally capture many relevant data analysis tasks. We prove that fact entailment in limit $Datalog_{\mathbb{Z}}$ is undecidable, but, after restricting the use of multiplication, it becomes CONEXPTIME- and CONP-complete in combined and data complexity, respectively. To achieve tractability in data complexity (which is very important for robust behaviour on large datasets), we additionally introduce a *stability* restriction and show that this does not prevent expressing the relevant analysis tasks.

The proofs of all results are given in an extended version of this paper [Kaminski *et al.*, 2017].

2 Preliminaries

In this section, we recapitulate the well-known definitions of Datalog with integers, which we call $Datalog_{\mathbb{Z}}$.

Syntax A vocabulary consists of *predicates*, *objects*, *object variables*, and *numeric variables*. Each predicate has an integer *arity* n, and each position $1 \leq i \leq n$ is of either *object* or *numeric sort*. An *object term* is an object or an object variable. A *numeric term* is an integer, a numeric variable, or of the form $s_1 + s_2$, $s_1 - s_2$, or $s_1 \times s_2$ where s_1 and s_2 are numeric terms, and $+$, $-$, and $\times$ are the standard *arithmetic functions*. A *constant* is an object or an integer. The *magnitude* of an integer is its absolute value. A *standard atom* is of the form $B(t_1, \ldots, t_n)$, where B is a predicate of arity n and each t_i is a term whose type matches the sort of position i of B. A *comparison atom* is of the form $(s_1 < s_2)$ or $(s_1 \leq s_2)$, where $<$ and $\leq$ are the standard *comparison predicates*, and s_1 and s_2 are numeric terms. A *rule* r is of the form $\bigwedge_i \alpha_i \wedge \bigwedge_j \beta_j \rightarrow \alpha$, where $\alpha_{(i)}$ are standard atoms, β_j are comparison atoms, and each variable in r occurs in some α_i. Atom $\mathsf{h}(r) = \alpha$ is the *head* of r; $\mathsf{sb}(r) = \bigwedge_i \alpha_i$ is the *standard body* of r; $\mathsf{cb}(r) = \bigwedge_j \beta_j$ is the *comparison body* of r; and $\mathsf{b}(r) = \mathsf{sb}(r) \wedge \mathsf{cb}(r)$ is the *body* of r. A *ground instance of* r is obtained from r by substituting all variables by constants. A ($Datalog_{\mathbb{Z}}$) *program* $\mathcal{P}$ is a finite set of rules. Predicate B is *intensional* (*IDB*) in $\mathcal{P}$ if B occurs in $\mathcal{P}$ in the head of a rule whose body is not empty; otherwise, B is *extensional* (*EDB*) in $\mathcal{P}$. A term, atom, rule, or program is *ground* if it contains no variables. A *fact* is a ground, function-free, standard atom. Program $\mathcal{P}$ is a *dataset* if $\mathsf{h}(r)$ is a fact and $\mathsf{b}(r) = \emptyset$ for each $r \in \mathcal{P}$. We often say that $\mathcal{P}$ contains a fact α and write $\alpha \in \mathcal{P}$, which actually means $\rightarrow \alpha \in \mathcal{P}$. We write a tuple of terms as $\mathbf{t}$, and we often treat conjunctions and tuples as sets and write, say, $\alpha_i \in \mathsf{sb}(r)$, $|\mathbf{t}|$, and $t_i \in \mathbf{t}$.

Semantics A *(Herbrand) interpretation* I is a (not necessarily finite) set of facts. Such I *satisfies* a ground atom α, written $I \models \alpha$, if (i) α is a standard atom and evaluating the arithmetic functions in α produces a fact in I, or (ii) α is a comparison atom and evaluating the arithmetic functions and comparisons produces $true$. The notion of satisfaction is extended to conjunctions of ground atoms, rules, and programs as in first-order logic, where each rule is universally quantified. If $I \models \mathcal{P}$, then I is a *model* of program $\mathcal{P}$; and $\mathcal{P}$ *entails* a fact α, written $\mathcal{P} \models \alpha$, if $I \models \alpha$ holds whenever $I \models \mathcal{P}$.

Complexity In this paper we study the computational properties of checking $\mathcal{P} \models \alpha$. *Combined complexity* assumes that both $\mathcal{P}$ and α are part of the input. In contrast, *data complexity* assumes that $\mathcal{P}$ is given as $\mathcal{P}' \cup \mathcal{D}$ for $\mathcal{P}'$ a program and $\mathcal{D}$ a dataset, and that only $\mathcal{D}$ and α are part of the input while $\mathcal{P}'$ is fixed. Unless otherwise stated, all numbers in the input are coded in binary, and the *size* $\|\mathcal{P}\|$ of $\mathcal{P}$ is the size of its representation. Checking $\mathcal{P} \models \alpha$ is undecidable even if the only arithmetic function in $\mathcal{P}$ is $+$ [Dantsin *et al.*, 2001].

Presburger arithmetic is first-order logic with constants 0 and 1, functions $+$ and $-$, equality, and the comparison predicates $<$ and $\leq$, interpreted over all integers $\mathbb{Z}$. The complexity of checking sentence validity (i.e., whether the sentence is true in all models of Presburger arithmetic) is known when the number of quantifier alternations and/or the number of variables in each quantifier block are fixed [Berman, 1980; Grädel, 1988; Schöning, 1997; Haase, 2014].

3 Limit Programs

Towards introducing a decidable fragment of $Datalog_{\mathbb{Z}}$ for data analysis, we first note that the undecidability proof of (plain) $Datalog_{\mathbb{Z}}$ outlined by Dantsin *et al.* [2001] uses atoms with at least two numeric terms. Thus, to motivate introducing our fragment, we first prove that undecidability holds even if atoms contain at most one numeric term. The proof uses a reduction from the halting problem for deterministic Turing machines. To ensure that each standard atom in $\mathcal{P}$ has at most one numeric term, combinations of a time point and a tape position are encoded using a single integer.

Theorem 1. *For $\mathcal{P}$ a $Datalog_{\mathbb{Z}}$ program and α a fact, checking $\mathcal{P} \models \alpha$ is undecidable even if $\mathcal{P}$ contains no $\times$ or $-$ and each standard atom in $\mathcal{P}$ has at most one numeric term.*

We next introduce *limit $Datalog_{\mathbb{Z}}$*, where *limit predicates* keep bounds on numeric values. This language can be seen as either a semantic or a syntactic restriction of $Datalog_{\mathbb{Z}}$.

Definition 2. *In* limit $Datalog_{\mathbb{Z}}$*, a predicate is either an* object predicate *with no numeric positions, or a* numeric predicate *where only the last position is numeric. A numeric predicate is either an* ordinary numeric predicate *or a* limit predicate*, and the latter is either a* min *or a* max *predicate. Atoms with object predicates are* object atoms*, and analogously for other types of atoms/predicates. A $Datalog_{\mathbb{Z}}$ rule r is a* limit ($Datalog_{\mathbb{Z}}$) rule *if (i) $\mathsf{b}(r) = \emptyset$, or (ii) each atom in $\mathsf{sb}(r)$ is an object, ordinary numeric, or limit atom, and $\mathsf{h}(r)$ is an object or a limit atom. A* limit ($Datalog_{\mathbb{Z}}$) program *$\mathcal{P}$ is a program containing only limit rules; and $\mathcal{P}$ is* homogeneous *if it does not contain both* min *and* max *predicates.*

In the rest of this paper we make three simplifying assumptions. First, numeric atoms occurring in a rule body are function-free (but comparison atoms and the head can contain arithmetic functions). Second, each numeric variable in a rule occurs in at most one standard body atom. Third, distinct rules in a program use different variables. The third assumption is clearly w.l.o.g. because all variables are universally quantified so their names are immaterial. Moreover, the first two assumptions are w.l.o.g. as well since, for each rule, there exists a logically equivalent rule that satisfies these assumptions. In particular, we can replace an atom such as $A(\mathbf{t}, m_1 + m_2)$ with conjunction

$$A(\mathbf{t}, m) \land I(m_1) \land I(m_2) \land \\ (m \leq m_1 + m_2) \land (m_1 + m_2 \leq m)$$

where m is a fresh variable and I is a fresh predicate axiomatised to hold on all integers as follows:

$$\to I(0) \qquad I(m) \to I(m+1) \qquad I(m) \to I(m-1)$$

Also, we can replace atoms $A_1(\mathbf{t}_1, m) \land A_2(\mathbf{t}_2, m)$ with conjunction $A_1(\mathbf{t}_1, m) \land A_2(\mathbf{t}_2, m') \land (m \leq m') \land (m' \leq m)$, where m' is a fresh variable.

Intuitively, a limit fact $B(\mathbf{a}, k)$ says that the value of B for a tuple of objects $\mathbf{a}$ is at least k (if B is max) or at most k (if B is min). For example, a fact $sp(v, k)$ in our shortest path example from Section 1 says that node v is reachable from v_0 via a path with cost at most k. To capture this intended meaning, we require interpretations I to be *closed* for limit predicates—that is, whenever I contains a limit fact α, it also contains all facts implied by α according to the predicate type. In our example, this captures the observation that the existence of a path from v_0 to v of cost at most k implies the existence of such a path of cost at most k' for each $k' \geq k$.

Definition 3. *An interpretation I is* limit-closed *if, for each limit fact $B(\mathbf{a}, k) \in I$ where B is a* min *(resp.* max*) predicate, $B(\mathbf{a}, k') \in I$ holds for each integer k' with $k \leq k'$ (resp. $k' \leq k$). An interpretation I is a* model *of a limit program $\mathcal{P}$ if $I \models \mathcal{P}$ and I is limit-closed. The notion of entailment is modified to take into account only limit-closed models.*

The semantics of limit predicates in a limit $Datalog_{\mathbb{Z}}$ program $\mathcal{P}$ can be axiomatised explicitly by extending $\mathcal{P}$ with the following rules, where Z is a fresh predicate. Thus, limit $Datalog_{\mathbb{Z}}$ can be seen as a syntactic fragment of $Datalog_{\mathbb{Z}}$.

$$\to Z(0) \qquad Z(m) \to Z(m+1) \qquad Z(m) \to Z(m-1)$$
$$B(\mathbf{x}, m) \land Z(n) \land (m \leq n) \to B(\mathbf{x}, n) \quad \text{for each min predicate } B \text{ in } \mathcal{P}$$
$$B(\mathbf{x}, m) \land Z(n) \land (n \leq m) \to B(\mathbf{x}, n) \quad \text{for each max predicate } B \text{ in } \mathcal{P}$$

Each limit program can be reduced to a homogeneous program; however, for the sake of generality, in our technical results we do not require programs to be homogeneous.

Proposition 4. *For each limit program $\mathcal{P}$ and fact α, a homogeneous program $\mathcal{P}'$ and fact α' can be computed in linear time such that $\mathcal{P} \models \alpha$ if and only if $\mathcal{P}' \models \alpha'$.*

Intuitively, program $\mathcal{P}'$ in Proposition 4 is obtained by replacing all min (or all max) predicates in $\mathcal{P}$ by fresh max (resp. min) predicates and negating their numeric arguments.

In Section 1 we have shown that limit $Datalog_{\mathbb{Z}}$ can compute the cost of shortest paths in a graph. We next present further examples of data analysis tasks that our formalism can handle. In all examples, we assume that all objects in the input are arranged in an arbitrary linear order using facts $\mathit{first}(a_1)$, $\mathit{next}(a_1, a_2)$, ..., $\mathit{next}(a_{n-1}, a_n)$; we use this order to simulate aggregation by means of recursion.

Example 5. *Consider a social network where agents are connected by the 'follows' relation. Agent a_s introduces (tweets) a message, and each agent a_i retweets the message if at least k_{a_i} agents that a_i follows tweet the message, where k_{a_i} is a positive threshold uniquely associated with a_i. Our goal is to determine which agents tweet the message eventually. To achieve this using limit $Datalog_{\mathbb{Z}}$, we encode the network structure in a dataset $\mathcal{D}_{tw}$ containing facts $\mathit{follows}(a_i, a_j)$ if a_i follows a_j, and ordinary numeric facts $th(a_i, k_{a_i})$ if a_i's threshold is k_{a_i}. Program $\mathcal{P}_{tw}$, containing rules (3)–(8), encodes message propagation, where nt is a* max *predicate.*

$$\to tw(a_s) \tag{3}$$
$$\mathit{follows}(x, y') \land \mathit{first}(y) \to nt(x, y, 0) \tag{4}$$
$$\mathit{follows}(x, y) \land \mathit{first}(y) \land tw(y) \to nt(x, y, 1) \tag{5}$$
$$nt(x, y', m) \land \mathit{next}(y', y) \to nt(x, y, m) \tag{6}$$
$$nt(x, y', m) \land \mathit{next}(y', y) \land \\ \mathit{follows}(x, y) \land tw(y) \to nt(x, y, m+1) \tag{7}$$
$$th(x, m) \land nt(x, y, n) \land (m \leq n) \to tw(x) \tag{8}$$

Specifically, $\mathcal{P}_{tw} \cup \mathcal{D}_{tw} \models tw(a_i)$ iff a_i tweets the message. Intuitively, $nt(a_i, a_j, m)$ is true if, out of agents $\{a_1, \ldots, a_j\}$ (according to the order), at least m agents that a_i follows tweet the message. Rules (4) and (5) initialise nt for the first agent in the order; nt is a max *predicate, so if the first agent tweets the message, rule (5) 'overrides' rule (4). Rules (6) and (7) recurse over the order to compute nt as stated above.*

Example 6. *Limit $Datalog_{\mathbb{Z}}$ can also solve the problem of counting paths between pairs of nodes in a directed acyclic graph. We encode the graph in the obvious way as a dataset $\mathcal{D}_{cp}$ that uses object predicates node and edge. Program $\mathcal{P}_{cp}$, consisting of rules (9)–(14) where np and np' are* max *predicates, then counts the paths.*

$$\mathit{node}(x) \to np(x, x, 1) \tag{9}$$
$$\mathit{node}(x) \land \mathit{node}(y) \land \mathit{first}(z) \to np'(x, y, z, 0) \tag{10}$$
$$\mathit{edge}(x, z) \land \\ np(z, y, m) \land \mathit{first}(z) \to np'(x, y, z, m) \tag{11}$$
$$np'(x, y, z', m) \land \mathit{next}(z', z) \to np'(x, y, z, m) \tag{12}$$
$$np'(x, y, z', m) \land \mathit{next}(z', z) \land \\ \mathit{edge}(x, z) \land np(z, y, n) \to np'(x, y, z, m+n) \tag{13}$$
$$np'(x, y, z, m) \to np(x, y, m) \tag{14}$$

Specifically, $\mathcal{P}_{cp} \cup \mathcal{D}_{cp} \models np(a_i, a_j, k)$ iff at least k paths exist from node a_i to node a_j. Intuitively, $np'(a_i, a_j, a_k, m)$ is true if m is at least the sum of the number of paths from each $a' \in \{a_1, \ldots, a_k\}$ (according to the order) to a_j for which there exists an edge from a_i to a'. Rule (9) says that each node has one path to itself. Rule (10) initialises aggregation by saying that, for the first node z, there are zero paths from x to y, and rule (11) overrides this if there exists an edge from x to z. Finally, rule (12) propagates the sum for x to the next z in the order, and rule (13) overrides this if there is an edge from x to z by adding the number of paths from z' and z to y.

Example 7. *Assume that, in the graph from Example 6, each node a_i is associated with a bandwidth b_{a_i} limiting the number of paths going through a_i to at most b_{a_i}. To count the paths compliant with the bandwidth requirements, we extend $\mathcal{D}_{cp}$ to dataset $\mathcal{D}_{bcp}$ that additionally contains an ordinary numeric fact $bw(a_i, b_{a_i})$ for each node a_i, and we define $\mathcal{P}_{bcp}$ by replacing rule* (14) *in $\mathcal{P}_{cp}$ with the following rule.*

$$np'(x, y, z, m) \wedge bw(z, n) \wedge (m \leq n) \rightarrow np(x, y, m)$$

Then, $\mathcal{P}_{bcp} \cup \mathcal{D}_{bcp} \models np(a_i, a_j, k)$ iff there exist at least k paths from node a_i to node a_j, where the bandwidth requirement is satisfied for all nodes on each such path.

4 Fixpoint Characterisation of Entailment

Programs are often grounded to eliminate variables and thus simplify the presentation. In limit $Datalog_{\mathbb{Z}}$, however, numeric variables range over integers, so a grounding can be infinite. Thus, we first specialise the notion of a grounding.

Definition 8. *A rule r is* semi-ground *if each variable in r is a numeric variable that occurs in r in a limit body atom. A limit program $\mathcal{P}$ is* semi-ground *if all of its rules are semi-ground. The* semi-grounding *of $\mathcal{P}$ contains, for each $r \in \mathcal{P}$, each rule obtained from r by replacing each variable not occurring in r in a numeric argument of a limit atom with a constant of $\mathcal{P}$.*

Obviously, $\mathcal{P} \models \alpha$ if and only if $\mathcal{P}' \models \alpha$ for $\mathcal{P}'$ the semi-grounding of $\mathcal{P}$. We next characterise entailment of limit programs by *pseudo-interpretations*, which compactly represent limit-closed interpretations. If a limit-closed interpretation I contains $B(\mathbf{b}, k)$ where B is a min predicate, then either the *limit* value $\ell \leq k$ exists such that $B(\mathbf{b}, \ell) \in I$ and $B(\mathbf{b}, k') \notin I$ for $k' < \ell$, or $B(\mathbf{b}, k') \in I$ holds for all $k' \leq k$, and dually for B a max predicate. Thus, to characterise the value of B on a tuple of objects $\mathbf{b}$ in I, we just need the limit value, or information that no such value exists.

Definition 9. *A* pseudo-interpretation *J is a set of facts over integers extended with a special symbol ∞ such that $k = k'$ holds for all limit facts $B(\mathbf{b}, k)$ and $B(\mathbf{b}, k')$ in I.*

Limit-closed interpretations correspond naturally and one-to-one to pseudo-interpretations, so we can recast the notions of satisfaction and model using pseudo-interpretations. Unlike for interpretations, the number of facts in a pseudo-model of a semi-ground limit program $\mathcal{P}$ can be bounded by $|\mathcal{P}|$.

Definition 10. *A limit-closed interpretation I* corresponds *to a pseudo-interpretation J if I contains exactly all object and ordinary numeric facts of J, and, for each limit predicate B, each tuple of objects $\mathbf{b}$, and each integer ℓ, (i) $B(\mathbf{b}, k) \in I$ for all k if and only if $B(\mathbf{b}, \infty) \in J$, and (ii) $B(\mathbf{b}, \ell) \in I$ and $B(\mathbf{b}, k) \notin I$ for all $k < \ell$ (resp. $\ell < k$) and B is a* min *(resp.* max*) predicate if and only if $B(\mathbf{b}, \ell) \in J$.*

Let J and J' be pseudo-interpretations corresponding to interpretations I and I'. Then, J satisfies *a ground atom α, written $J \models \alpha$, if $I \models \alpha$; J is a* pseudo-model *of a program $\mathcal{P}$, written $J \models \mathcal{P}$, if $I \models \mathcal{P}$; finally, $J \sqsubseteq J'$ holds if $I \subseteq I'$.*

Example 11. *Let I be the interpretation consisting of $A(1)$, $A(2)$, $B(a, k)$ for $k \leq 5$, and $B(b, k)$ for $k \in \mathbb{Z}$, where A is an ordinary numeric predicate, B is a* max *predicate, and a and b are objects. Then, $\{A(1), A(2), B(a, 5), B(b, \infty)\}$ is the pseudo-interpretation corresponding to I.*

We next introduce the *immediate consequence* operator $\mathbf{T}_{\mathcal{P}}$ of a limit program $\mathcal{P}$ on pseudo-interpretations. We assume for simplicity that $\mathcal{P}$ is semi-ground. To apply a rule $r \in \mathcal{P}$ to a pseudo-interpretation J while correctly handling limit atoms, operator $\mathbf{T}_{\mathcal{P}}$ converts r into a linear integer constraint $\mathcal{C}(r, J)$ that captures all ground instances of r applicable to the limit-closed interpretation I corresponding to J. If $\mathcal{C}(r, J)$ has no solution, r is not applicable to J. Otherwise, $\mathsf{h}(r)$ is added to J if it is not a limit atom; and if $\mathsf{h}(r)$ is a min (max) atom $B(\mathbf{b}, m)$, then the minimal (maximal) solution ℓ for m in $\mathcal{C}(r, J)$ is computed, and J is updated such that the limit value of B on $\mathbf{b}$ is at least (at most) ℓ—that is, the application of r to J keeps only the 'best' limit value.

Definition 12. *For $\mathcal{P}$ a semi-ground limit program, $r \in \mathcal{P}$, and J a pseudo-interpretation, $\mathcal{C}(r, J)$ is the conjunction of comparison atoms containing (i) $\mathsf{cb}(r)$; (ii) $(0 < 0)$ if an object or ordinary numeric atom $\alpha \in \mathsf{sb}(r)$ exists with $\alpha \notin J$, or a limit atom $B(\mathbf{b}, s) \in \mathsf{sb}(r)$ exists with $B(\mathbf{b}, \ell) \notin J$ for each ℓ; and (iii) $(\ell \leq s)$ (resp. $(s \leq \ell)$) for each* min *(resp.* max*) atom $B(\mathbf{b}, s) \in \mathsf{sb}(r)$ with $B(\mathbf{b}, \ell) \in J$ and $\ell \neq \infty$. Rule r is* applicable *to J if $\mathcal{C}(r, J)$ has an integer solution.*

Assume r is applicable to J. If $\mathsf{h}(r)$ is an object or ordinary numeric atom, let $\mathsf{hd}(r, J) = \mathsf{h}(r)$. If $\mathsf{h}(r) = B(\mathbf{b}, s)$ is a min *(resp.* max*) atom, the* optimum value *$\mathsf{opt}(r, J)$ is the smallest (resp. largest) value of s in all solutions to $\mathcal{C}(r, J)$, or ∞ if no such bound on the value of s in the solutions to $\mathcal{C}(r, J)$ exists; moreover, $\mathsf{hd}(r, J) = B(\mathbf{b}, \mathsf{opt}(r, J))$.*

Operator $\mathbf{T}_{\mathcal{P}}(J)$ maps J to the smallest (w.r.t. $\sqsubseteq$) pseudo-interpretation satisfying $\mathsf{hd}(r, J)$ for each $r \in \mathcal{P}$ applicable to J. Finally, $\mathbf{T}^0_{\mathcal{P}} = \emptyset$, and $\mathbf{T}^n_{\mathcal{P}} = \mathbf{T}_{\mathcal{P}}(\mathbf{T}^{n-1}_{\mathcal{P}})$ for $n > 0$.

Example 13. *Let r be $A(x) \wedge (2 \leq x) \rightarrow B(x + 1)$ with A and B* max *predicates. Then, $\mathcal{C}(r, \emptyset) = (2 \leq x) \wedge (0 < 0)$ does not have a solution, and therefore rule r is not applicable to the empty pseudo-interpretation. Moreover, for $J = \{A(3)\}$, conjunction $\mathcal{C}(r, J) = (2 \leq x) \wedge (x \leq 3)$ has two solutions—$\{x \mapsto 2\}$ and $\{x \mapsto 3\}$—and therefore rule r is applicable to J. Finally, B is a* max *predicate, and so $\mathsf{opt}(r, J) = \max\{2 + 1, 3 + 1\} = 4$, and $\mathsf{hd}(r, J) = B(4)$. Consequently, $\mathbf{T}_{\{r\}}(J) = \{B(4)\}$.*

Lemma 14. *For each semi-ground limit program $\mathcal{P}$, operator $\mathbf{T}_{\mathcal{P}}$ is monotonic w.r.t. $\sqsubseteq$. Moreover, $J \models \mathcal{P}$ if and only if $\mathbf{T}_{\mathcal{P}}(J) \sqsubseteq J$ for each pseudo-interpretation J.*

Monotonicity ensures existence of the *closure* $\mathbf{T}^\infty_{\mathcal{P}}$ of $\mathcal{P}$—the least pseudo-interpretation such that $\mathbf{T}^n_{\mathcal{P}} \sqsubseteq \mathbf{T}^\infty_{\mathcal{P}}$ for each $n \geq 0$. The following theorem characterises entailment and provides a bound on the number of facts in the closure.

Theorem 15. *For $\mathcal{P}$ a semi-ground limit program and α a fact, $\mathcal{P} \models \alpha$ if and only if $\mathbf{T}^\infty_{\mathcal{P}} \models \alpha$; also, $|\mathbf{T}^\infty_{\mathcal{P}}| \leq |\mathcal{P}|$; and $J \models \mathcal{P}$ implies $\mathbf{T}^\infty_{\mathcal{P}} \sqsubseteq J$ for each pseudo-interpretation J.*

The proofs for the first and the third claim of Theorem 15 use the monotonicity of $\mathbf{T}_{\mathcal{P}}$ analogously to plain Datalog. The second claim holds since, for each $n \geq 0$, each pair of distinct facts in $\mathbf{T}^n_{\mathcal{P}}$ must be derived by distinct rules in $\mathcal{P}$.

5 Decidability of Entailment: Limit-Linearity

We now start our investigation of the computational properties of limit $Datalog_{\mathbb{Z}}$. Theorem 15 bounds the cardinality of the closure of a semi-ground program, but it does not bound the magnitude of the integers occurring in limit facts; in fact, integers can be arbitrarily large. Moreover, due to multiplication, checking rule applicability requires solving nonlinear inequalities over integers, which is undecidable.

Theorem 16. *For $\mathcal{P}$ a semi-ground limit program and α a fact, checking $\mathcal{P} \models \alpha$ and checking applicability of a rule of $\mathcal{P}$ to a pseudo-interpretation are both undecidable.*

The proof of Theorem 16 uses a straightforward reduction from Hilbert's tenth problem.

Checking rule applicability is undecidable due to products of variables in inequalities; however, for *linear inequalities* that prohibit multiplying variables, the problem can be solved in NP, and in polynomial time if we bound the number of variables. Thus, to ensure decidability, we next restrict limit programs so that their semi-groundings contain only linear numeric terms. All our examples satisfy this restriction.

Definition 17. *A limit rule r is* limit-linear *if each numeric term in r is of the form $s_0 + \sum_{i=1}^{n} s_i \times m_i$, where (i) each m_i is a distinct numeric variable occurring in a limit body atom of r, (ii) term s_0 contains no variable occurring in a limit body atom of r, and (iii) each s_i with $i \geq 1$ is a term constructed using multiplication $\times$, integers, and variables not occurring in limit body atoms of r. A limit-linear program contains only limit-linear rules.*[1]

In the rest of this section, we show that entailment for limit-linear programs is decidable and provide tight complexity bounds. Our upper bounds are obtained via a reduction to the validity of Presburger formulas of a certain shape.

Lemma 18. *For $\mathcal{P}$ a semi-ground limit-linear program and α a fact, there exists a Presburger sentence $\varphi = \forall \mathbf{x} \exists \mathbf{y}. \bigvee_{i=1}^{n} \psi_i$ that is valid if and only if $\mathcal{P} \models \alpha$. Each ψ_i is a conjunction of possibly negated atoms. Moreover, $|\mathbf{x}| + |\mathbf{y}|$ and each $\|\psi_i\|$ are bounded polynomially by $\|\mathcal{P}\| + \|\alpha\|$. Number n is bounded polynomially by $|\mathcal{P}|$ and exponentially by $\max_{r \in \mathcal{P}} \|r\|$. Finally, the magnitude of each integer in φ is bounded by the maximal magnitude of an integer in $\mathcal{P}$ and α.*

The reduction in Lemma 18 is based on three main ideas. First, for each limit atom $B(\mathbf{b}, s)$ in a semi-ground program $\mathcal{P}$, we use a Boolean variable $def_{B\mathbf{b}}$ to indicate that an atom of the form $B(\mathbf{b}, \ell)$ exists in a pseudo-model of $\mathcal{P}$, a Boolean variable $fin_{B\mathbf{b}}$ to indicate whether the value of ℓ is finite, and an integer variable $val_{B\mathbf{b}}$ to capture ℓ if it is finite. Second, each rule of $\mathcal{P}$ is encoded as a universally quantified Presburger formula by replacing each standard atom with its encoding. Finally, entailment of α from $\mathcal{P}$ is encoded as a sentence stating that, in every pseudo-interpretation, either some rule in $\mathcal{P}$ is not satisfied, or α holds; this requires universal quantifiers to quantify over all models, and existential quantifiers to negate the (universally quantified) program.

Lemma 19 bounds the magnitude of integers in models of Presburger formulas from Lemma 18. These bounds follow from recent deep results on semi-linear sets and their connection to Presburger arithmetic [Chistikov and Haase, 2016].

[1]Note that each multiplication-free limit program can be normalised in polynomial time to a limit-linear program.

Lemma 19. *Let $\varphi = \forall \mathbf{x} \exists \mathbf{y}. \bigvee_{i=1}^{n} \psi_i$ be a Presburger sentence where each ψ_i is a conjunction of possibly negated atoms of size at most k mentioning at most ℓ variables, a is the maximal magnitude of an integer in φ, and $m = |\mathbf{x}|$. Then, φ is valid if and only if φ is valid over models where each integer variable assumes a value whose magnitude is bounded by $(2^{O(\ell \log \ell)} \cdot a^{k\ell})^{n 2^{\ell} \cdot O(m^4)}$.*

Lemmas 18 and 19 provide us with bounds on the size of counter-pseudo-models for entailment.

Theorem 20. *For $\mathcal{P}$ a semi-ground limit-linear program, $\mathcal{D}$ a dataset, and α a fact, $\mathcal{P} \cup \mathcal{D} \not\models \alpha$ if and only if a pseudo-model J of $\mathcal{P} \cup \mathcal{D}$ exists where $J \not\models \alpha$, $|J| \leq |\mathcal{P} \cup \mathcal{D}|$, and the magnitude of each integer in J is bounded polynomially by the largest magnitude of an integer in $\mathcal{P} \cup \mathcal{D}$, exponentially by $|\mathcal{P}|$, and double-exponentially by $\max_{r \in \mathcal{P}} \|r\|$.*

By Theorem 20, the following nondeterministic algorithm decides $\mathcal{P} \not\models \alpha$:

1. compute the semi-grounding $\mathcal{P}'$ of $\mathcal{P}$;
2. guess a pseudo-interpretation J that satisfies the bounds given in Theorem 20;
3. if $\mathbf{T}_{\mathcal{P}'}(J) \sqsubseteq J$ (so $J \models \mathcal{P}'$) and $J \not\models \alpha$, return true.

Step 1 requires exponential (polynomial in data) time, and it does not increase the maximal size of a rule. Hence, Step 2 is nondeterministic exponential (polynomial in data), and Step 3 requires exponential (polynomial in data) time to solve a system of linear inequalities. Theorem 21 proves that these bounds are both correct and tight.

Theorem 21. *For $\mathcal{P}$ a limit-linear program and α a fact, deciding $\mathcal{P} \models \alpha$ is* CONEXPTIME*-complete in combined and* CONP*-complete in data complexity.*

The upper bounds in Theorem 21 follow from Theorem 20, CONP-hardness in data complexity is shown by a reduction from the square tiling problem, and CONEXPTIME-hardness in combined complexity is shown by a similar reduction from the succinct version of square tiling.

6 Tractability of Entailment: Stability

Tractability in data complexity is important on large datasets, so we next present an additional *stability* condition that brings the complexity of entailment down to EXPTIME in combined and PTIME in data complexity, as in plain Datalog.

6.1 Cyclic Dependencies in Limit Programs

The fixpoint of a plain Datalog program can be computed in PTIME in data complexity. However, for $\mathcal{P}$ a limit-linear program, a naïve computation of $\mathbf{T}_{\mathcal{P}}^{\infty}$ may not terminate since repeated application of $\mathbf{T}_{\mathcal{P}}$ can produce larger and larger numbers. Thus, we need a way to identify when the numeric argument ℓ of a limit fact $A(\mathbf{a}, \ell)$ *diverges*—that is, grows or decreases without a bound; moreover, to obtain a procedure tractable in data complexity, divergence should be detected after polynomially many steps. Example 22 illustrates that this can be achieved by analysing cyclic dependencies.

Example 22. *Let $\mathcal{P}_c$ contain facts $A(0)$ and $B(0)$, and rules $A(m) \rightarrow B(m)$ and $B(m) \rightarrow A(m+1)$, where A and B are* max *predicates. Applying the first rule copies the value* *of A into B, and applying the second rule increases the value of A; thus, both A and B diverge in $\mathbf{T}^{\infty}_{\mathcal{P}_c}$. The existence of a cyclic dependency between A and B, however, does not necessarily lead to divergence. Let program $\mathcal{P}'_c$ be obtained from $\mathcal{P}_c$ by adding a* max *fact $C(5)$ and replacing the first rule with $A(m) \wedge C(n) \wedge (m \leq n) \rightarrow B(m)$. While a cyclic dependency between A and B still exists, the increase in the values of A and B is bounded by the value of C, which is independent of A or B; thus, neither A nor B diverge in $\mathbf{T}^{\infty}_{\mathcal{P}'_c}$.*

In the rest of this section, we extend $<$, $+$, and $-$ to ∞ by defining $k < \infty$, $\infty + k = \infty$, and $\infty - k = \infty$ for each integer k. We formalise cyclic dependencies as follows.

Definition 23. *For each n-ary limit predicate B and each tuple $\mathbf{b}$ of $n-1$ objects, let $v_{B\mathbf{b}}$ be a* node *unique for B and $\mathbf{b}$. The* value propagation graph *of a semi-ground limit-linear program $\mathcal{P}$ and a pseudo-interpretation J is the directed weighted graph $G^J_{\mathcal{P}} = (\mathcal{V}, \mathcal{E}, \mu)$ defined as follows.*

1. *For each limit fact $B(\mathbf{b}, \ell) \in J$, we have $v_{B\mathbf{b}} \in \mathcal{V}$.*
2. *For each rule $r \in \mathcal{P}$ applicable to J with the head of the form $A(\mathbf{a}, s)$ where $v_{A\mathbf{a}} \in \mathcal{V}$ and each body atom $B(\mathbf{b}, m)$ of r where $v_{B\mathbf{b}} \in \mathcal{V}$ and variable m occurs in term s, we have $\langle v_{B\mathbf{b}}, v_{A\mathbf{a}} \rangle \in \mathcal{E}$; such r is said to* produce *the edge $\langle v_{B\mathbf{b}}, v_{A\mathbf{a}} \rangle$ in $\mathcal{E}$.*
3. *For each $r \in \mathcal{P}$ and each edge $e = \langle v_{B\mathbf{b}}, v_{A\mathbf{a}} \rangle \in \mathcal{E}$ produced by r, $\delta^e_r(J) = \infty$ if $\mathsf{opt}(r, J) = \infty$; otherwise,*

$$\delta^e_r(J) = \begin{cases} \mathsf{opt}(r,J) - \ell & \text{if } B \text{ and } A \text{ are } \mathsf{max}, \ell \neq \infty \\ -\mathsf{opt}(r,J) - \ell & \text{if } B \text{ is } \mathsf{max}, A \text{ is } \mathsf{min}, \ell \neq \infty \\ -\mathsf{opt}(r,J) + \ell & \text{if } B \text{ and } A \text{ are } \mathsf{min}, \ell \neq \infty \\ \mathsf{opt}(r,J) + \ell & \text{if } B \text{ is } \mathsf{min}, A \text{ is } \mathsf{max}, \ell \neq \infty \\ \mathsf{opt}(r,J) & \text{if } \ell = \infty \end{cases}$$

where ℓ is such that $B(\mathbf{b}, \ell) \in J$. The weight of each edge $e \in \mathcal{E}$ is then given by

$$\mu(e) = \max\{\delta^e_r(J) \mid r \in \mathcal{P} \text{ produces } e\}.$$

A positive-weight cycle *in $G^J_{\mathcal{P}}$ is a cycle for which the sum of the weights of the contributing edges is greater than 0.*

Intuitively, $G^J_{\mathcal{P}}$ describes how, for each limit predicate B and objects $\mathbf{b}$ such that $B(\mathbf{b}, \ell) \in J$, operator $\mathbf{T}_{\mathcal{P}}$ propagates ℓ to other facts. The presence of a node $v_{B\mathbf{b}}$ in $\mathcal{V}$ indicates that $B(\mathbf{b}, \ell) \in J$ holds for some $\ell \in \mathbb{Z} \cup \{\infty\}$; this ℓ can be uniquely identified given $v_{B\mathbf{b}}$ and J. An edge $e = \langle v_{B\mathbf{b}}, v_{A\mathbf{a}} \rangle \in \mathcal{E}$ indicates that at least one rule $r \in \mathcal{P}$ is applicable to J where $\mathsf{h}(r) = A(\mathbf{a}, s)$, $B(\mathbf{b}, m) \in \mathsf{sb}(r)$, and m occurs in s; moreover, applying r to J produces a fact $A(\mathbf{a}, \ell')$ where ℓ' satisfies $\ell + \mu(e) \leq \ell'$ if both A and B are max predicates, and analogously for the other types of A and B. In other words, edge e indicates that the application of $\mathbf{T}_{\mathcal{P}}$ to J will propagate the value of $v_{B\mathbf{b}}$ to $v_{A\mathbf{a}}$ while increasing it by at least $\mu(e)$. Thus, presence of a positive-weight cycle in $G^J_{\mathcal{P}}$ indicates that repeated rule applications might increment the values of all nodes on the cycle.

6.2 Stable Programs

As Example 22 shows, the presence of a positive-weight cycle in $G^J_{\mathcal{P}}$ does not imply the divergence of all atoms corresponding to the nodes in the cycle. This is because the weight of such a cycle may decrease after certain rule applications and so it is no longer positive.

This motivates the *stability* condition, where edge weights in $G^J_{\mathcal{P}}$ may only grow but never decrease with rule application. Hence, once the weight of a cycle becomes positive, it will remain positive and thus guarantee the divergence of all atoms corresponding to its nodes. Intuitively, $\mathcal{P}$ is stable if, whenever a rule $r \in \mathcal{P}$ is applicable to some J, rule r is also applicable to each J' with larger limit values, and applying r to such J' further increases the value of the head. Definition 24 defines stability as a condition on $G^J_{\mathcal{P}}$. Please note that, for all pseudo-interpretations J and J' with $J \sqsubseteq J'$ and $G^J_{\mathcal{P}} = (\mathcal{V}, \mathcal{E}, \mu)$ and $G^{J'}_{\mathcal{P}} = (\mathcal{V}', \mathcal{E}', \mu')$ the corresponding value propagation graphs, we have $\mathcal{E} \subseteq \mathcal{E}'$.

Definition 24. *A semi-ground limit-linear program $\mathcal{P}$ is* stable *if, for all pseudo-interpretations J and J' with $J \sqsubseteq J'$, $G^J_{\mathcal{P}} = (\mathcal{V}, \mathcal{E}, \mu)$, $G^{J'}_{\mathcal{P}} = (\mathcal{V}', \mathcal{E}', \mu')$, and each $e \in \mathcal{E}$,*

1. *$\mu(e) \leq \mu'(e)$, and*
2. *$e = \langle v_{B\mathbf{b}}, v_{A\mathbf{a}} \rangle$ and $B(\mathbf{b}, \infty) \in J$ imply $\mu(e) = \infty$.*

A limit-linear program is stable *if its semi-grounding is stable.*

Example 25. *Program $\mathcal{P}_c$ from Example 22 is stable, while $\mathcal{P}'_c$ is not: for $J = \{A(0), C(0)\}$ and $J' = \{A(1), C(0)\}$, we have $J \sqsubseteq J'$, but $\mu(\langle v_A, v_B \rangle) = 0$ and $\mu'(\langle v_A, v_B \rangle) = -1$.*

For each semi-ground program $\mathcal{P}$ and each integer n, we have $\mathbf{T}^n_{\mathcal{P}} \sqsubseteq \mathbf{T}^{n+1}_{\mathcal{P}}$, and stability ensures that edge weights only grow after rule application. Thus, recursive application of the rules producing edges involved in a positive-weight cycle leads to divergence, as shown by the following lemma.

Lemma 26. *For each semi-ground stable program $\mathcal{P}$, each pseudo-interpretation J with $J \sqsubseteq \mathbf{T}^{\infty}_{\mathcal{P}}$, and each node $v_{A\mathbf{a}}$ on a positive-weight cycle in $G^J_{\mathcal{P}}$, we have $A(\mathbf{a}, \infty) \in \mathbf{T}^{\infty}_{\mathcal{P}}$.*

Algorithm 1 uses this observation to deterministically compute the fixpoint of $\mathcal{P}$. The algorithm iteratively applies $\mathbf{T}_{\mathcal{P}}$; however, after each step, it computes the corresponding value propagation graph (line 4) and, for each $A(\mathbf{a}, \ell)$ where node $v_{A\mathbf{a}}$ occurs on a positive-weight cycle (line 5), it replaces ℓ with ∞ (line 6). By Lemma 26, this is sound. Moreover, since the algorithm repeatedly applies $\mathbf{T}_{\mathcal{P}}$, it necessarily derives each fact from $\mathbf{T}^{\infty}_{\mathcal{P}}$ eventually. Finally, Lemma 27 shows that the algorithm terminates in time polynomial in the number of rules in a semi-ground program. Intuitively, the proof of the lemma shows that, without introducing a new edge or a new positive weight cycle in the value propagation graph, repeated application of $\mathbf{T}_{\mathcal{P}}$ necessarily converges in $O(|\mathcal{P}|^2)$ steps; moreover, the number of edges in $G^J_{\mathcal{P}}$ is at most quadratic $|\mathcal{P}|$, and so a new edge or a new positive weight cycle can be introduced at most $O(|\mathcal{P}|^2)$ many times.

Lemma 27. *When applied to a semi-ground stable program $\mathcal{P}$, Algorithm 1 terminates after at most $8|\mathcal{P}|^6$ iterations of the loop in lines 2–8.*

Lemmas 26 and 27 imply the following theorem.

Theorem 28. *For $\mathcal{P}$ a semi-ground stable program, $\mathcal{D}$ a dataset, and α a fact, Algorithm 1 decides $\mathcal{P} \cup \mathcal{D} \models \alpha$ in time polynomial in $\|\mathcal{P} \cup \mathcal{D}\|$ and exponential in $\max_{r \in \mathcal{P}} \|r\|$.*

Since the running time is exponential in the maximal size of a rule, and semi-grounding does not increase rule sizes,

Algorithm 1 Entailment for Semi-Ground Stable Programs

```
Input: semi-ground stable program $\mathcal{P}$, fact $\alpha$
Output: true if $\mathcal{P} \models \alpha$
1: $J' := \emptyset$
2: repeat
3:     $J := J'$
4:     $G^J_{\mathcal{P}} := (\mathcal{V}, \mathcal{E}, \mu)$
5:     for each $v_{A\mathbf{a}} \in \mathcal{V}$ in a positive-weight cycle in $G^J_{\mathcal{P}}$ do
6:         replace $A(\mathbf{a}, \ell)$ in $J$ with $A(\mathbf{a}, \infty)$
7:     $J' := \mathbf{T}_{\mathcal{P}}(J)$
8: until $J = J'$
9: return true if $J \models \alpha$ and false otherwise
```

Algorithm 1 combined with a semi-grounding preprocessing step provides an exponential time decision procedure for stable, limit-linear programs. This upper bound is tight since entailment in plain Datalog is already EXPTIME-hard in combined and PTIME-hard in data complexity.

Theorem 29. *For $\mathcal{P}$ a stable program and α a fact, checking $\mathcal{P} \models \alpha$ is* EXPTIME*-complete in combined and* PTIME*-complete in data complexity.*

6.3 Type-Consistent Programs

Unfortunately, the class of stable programs is not recognisable, which can again be shown by a reduction from Hilbert's tenth problem.

Proposition 30. *Checking stability of a limit-linear program $\mathcal{P}$ is undecidable.*

We next provide a sufficient condition for stability that captures programs such as those in Examples 5–7. Intuitively, Definition 31 syntactically prevents certain harmful interactions. In the second rule of program $\mathcal{P}'_c$ from Example 22, numeric variable m occurs in a max atom and on the left-hand side of a comparison atom $(m \leq n)$; thus, if the rule is applicable for some value of m, it is not necessarily applicable for each $m' \geq m$, which breaks stability.

Definition 31. *A semi-ground limit-linear rule r is* type-consistent *if*

- *each numeric term t in r is of the form $k_0 + \sum_{i=1}^{n} k_i \times m_i$ where k_0 is an integer and each k_i, $1 \leq i \leq n$, is a nonzero integer, called the* coefficient of variable m_i in t*;*
- *if $\mathsf{h}(r) = A(\mathbf{a}, s)$ is a limit atom, then each variable occurring in s with a positive (resp. negative) coefficient also occurs in a (unique) limit body atom or r that is of the same (resp. different) type (i.e.,* min *vs.* max*) as $\mathsf{h}(r)$; and*
- *for each comparison $(s_1 < s_2)$ or $(s_1 \leq s_2)$ in r, each variable occurring in s_1 with a positive (resp. negative) coefficient also occurs in a (unique)* min *(resp.* max*) body atom, and each variable occurring in s_2 with a positive (resp. negative) coefficient also occurs in a (unique)* max *(resp.* min*) body atom of r.*

A semi-ground limit-linear program is type-consistent *if all of its rules are type-consistent. Moreover, a limit-linear program $\mathcal{P}$ is* type-consistent *if the program obtained by first semi-grounding $\mathcal{P}$ and then simplifying all numeric terms as much as possible is type-consistent.*

The first condition of Definition 31 ensures that each variable occurring in a numeric term contributes to the value of the term. For example, it disallows terms such as $0 \cdot x$ and $x - x$, since a rule with such a term in the head may violate the second condition. Moreover, the second condition of Definition 31 ensures that, if the value of a numeric variable x occurring in the head 'increases' w.r.t. the type of the body atom introducing x (i.e., x increases if it occurs in a max body atom and decreases otherwise), then so does the value of the numeric term in the head; this is essential for the first condition of stability (cf. Definition 24). Finally, the third condition of Definition 31 ensures that comparisons cannot be invalidated by 'increasing' the values of the variables involved, which is required for both conditions of stability.

Type consistency is a purely syntactic condition that can be checked by looking at one rule and one atom at a time. Hence, checking type consistency is feasible in LOGSPACE.

Proposition 32. *Each type-consistent limit-linear program is stable.*

Proposition 33. *Checking whether a limit-linear program is type-consistent can be accomplished in* LOGSPACE.

7 Conclusion and Future Work

We have introduced several decidable/tractable fragments of Datalog with integer arithmetic, thus obtaining a sound theoretical foundation for declarative data analysis. We see many challenges for future work. First, our formalism should be extended with aggregate functions. While certain forms of aggregation can be simulated by iterating over the object domain, as in our examples in Section 3, such a solution may be too cumbersome for practical use, and it relies on the existence of a linear order over the object domain, which is a strong theoretical assumption. Explicit support for aggregation would allow us to formulate tasks such as the ones in Section 3 more intuitively and without relying on the ordering assumption. Second, it is unclear whether integer constraint solving is strictly needed in Step 7 of Algorithm 1: it may be possible to exploit stability of $\mathcal{P}$ to compute $\mathbf{T}_{\mathcal{P}}(J)$ more efficiently. Third, we shall implement our algorithm and apply it to practical data analysis problems. Fourth, it would be interesting to establish connections between our results and existing work on data-aware artefact systems [Damaggio *et al.*, 2012; Koutsos and Vianu, 2017], which faces similar undecidability issues in a different formal setting.

Acknowledgments

We thank Christoph Haase for explaining to us his results on Presburger arithmetic and semi-linear sets, which were key to some of our technical results. Our work has also benefited from discussions with Michael Benedikt. This research was supported by the Royal Society and the EPSRC projects DBOnto, MaSI^3, and ED^3.

References

[Alvaro *et al.*, 2010] Peter Alvaro, Tyson Condie, Neil Conway, Khaled Elmeleegy, Joseph M. Hellerstein, and Russell Sears. BOOM analytics: exploring data-centric, declarative programming for the cloud. In *EuroSys*. ACM, 2010.

[Beeri *et al.*, 1991] Catriel Beeri, Shamim A. Naqvi, Oded Shmueli, and Shalom Tsur. Set constructors in a logic database language. *J. Log. Program.*, 10(3&4), 1991.

[Berman, 1980] Leonard Berman. The complexitiy of logical theories. *Theor. Comput. Sci.*, 11, 1980.

[Chin *et al.*, 2015] Brian Chin, Daniel von Dincklage, Vuk Ercegovac, Peter Hawkins, Mark S. Miller, Franz Josef Och, Christopher Olston, and Fernando Pereira. Yedalog: Exploring knowledge at scale. In *SNAPL*, 2015.

[Chistikov and Haase, 2016] Dmitry Chistikov and Christoph Haase. The taming of the semi-linear set. In *ICALP*, 2016.

[Consens and Mendelzon, 1993] Mariano P. Consens and Alberto O. Mendelzon. Low complexity aggregation in GraphLog and Datalog. *Theor. Comput. Sci.*, 116(1), 1993.

[Damaggio *et al.*, 2012] Elio Damaggio, Alin Deutsch, and Victor Vianu. Artifact systems with data dependencies and arithmetic. *ACM Trans. Database Syst.*, 37(3):22:1–22:36, 2012.

[Dantsin *et al.*, 2001] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3), 2001.

[Eisner and Filardo, 2011] Jason Eisner and Nathaniel Wesley Filardo. Dyna: Extending datalog for modern AI. In *Datalog*, 2011.

[Faber *et al.*, 2011] Wolfgang Faber, Gerald Pfeifer, and Nicola Leone. Semantics and complexity of recursive aggregates in answer set programming. *Artif. Intell.*, 175(1), 2011.

[Ganguly *et al.*, 1995] Sumit Ganguly, Sergio Greco, and Carlo Zaniolo. Extrema predicates in deductive databases. *J. Comput. Syst. Sci.*, 51(2), 1995.

[Grädel, 1988] Erich Grädel. Subclasses of presburger arithmetic and the polynomial-time hierarchy. *Theor. Comput. Sci.*, 56, 1988.

[Haase, 2014] Christoph Haase. Subclasses of Presburger arithmetic and the weak EXP hierarchy. In *CSL-LICS*, 2014.

[Kaminski *et al.*, 2017] Mark Kaminski, Bernardo Cuenca Grau, Egor V. Kostylev, Boris Motik, and Ian Horrocks. Foundations of declarative data analysis using limit datalog programs. *CoRR*, abs/1705.06927, 2017.

[Kemp and Stuckey, 1991] David B. Kemp and Peter J. Stuckey. Semantics of logic programs with aggregates. In *ISLP*, 1991.

[Koutsos and Vianu, 2017] Adrien Koutsos and Victor Vianu. Process-centric views of data-driven business artifacts. *J. Comput. System Sci.*, 86:82–107, 2017.

[Loo *et al.*, 2009] Boon Thau Loo, Tyson Condie, Minos N. Garofalakis, David E. Gay, Joseph M. Hellerstein, Petros Maniatis, Raghu Ramakrishnan, Timothy Roscoe, and Ion Stoica. Declarative networking. *Commun. ACM*, 52(11), 2009.

[Markl, 2014] Volker Markl. Breaking the chains: On declarative data analysis and data independence in the big data era. *PVLDB*, 7(13), 2014.

[Mazuran *et al.*, 2013] Mirjana Mazuran, Edoardo Serra, and Carlo Zaniolo. Extending the power of datalog recursion. *VLDB J.*, 22(4), 2013.

[Mumick *et al.*, 1990] Inderpal Singh Mumick, Hamid Pirahesh, and Raghu Ramakrishnan. The magic of duplicates and aggregates. In *VLDB*, pages 264–277, 1990.

[Ross and Sagiv, 1997] Kenneth A. Ross and Yehoshua Sagiv. Monotonic aggregation in deductive databases. *J. Comput. System Sci.*, 54(1), 1997.

[Schöning, 1997] Uwe Schöning. Complexity of presburger arithmetic with fixed quantifier dimension. *Theory Comput. Syst.*, 30(4), 1997.

[Seo *et al.*, 2015] Jiwon Seo, Stephen Guo, and Monica S. Lam. SociaLite: An efficient graph query language based on datalog. *IEEE Trans. Knowl. Data Eng.*, 27(7), 2015.

[Shkapsky *et al.*, 2016] Alexander Shkapsky, Mohan Yang, Matteo Interlandi, Hsuan Chiu, Tyson Condie, and Carlo Zaniolo. Big data analytics with datalog queries on Spark. In *SIGMOD*. ACM, 2016.

[Van Gelder, 1992] Allen Van Gelder. The well-founded semantics of aggregation. In *PODS*, 1992.

[Wang *et al.*, 2015] Jingjing Wang, Magdalena Balazinska, and Daniel Halperin. Asynchronous and fault-tolerant recursive datalog evaluation in shared-nothing engines. *PVLDB*, 8(12), 2015.