

# Complexity and Expressive Power of Disjunction and Negation in Limit Datalog

Mark Kaminski, Bernardo Cuenca Grau, Egor V. Kostylev, Ian Horrocks

Department of Computer Science, University of Oxford, UK

{mark.kaminski,bernardo.cuenca.grau,egor.kostylev,ian.horrocks}@cs.ox.ac.uk

## Abstract

Limit Datalog is a fragment of Datalog<sub>ℤ</sub>—the extension of Datalog with arithmetic functions over the integers—which has been proposed as a declarative language suitable for capturing data analysis tasks. In limit Datalog programs, all intensional predicates with a numeric argument are limit predicates that keep maximal (or minimal) bounds on numeric values. Furthermore, to ensure decidability of reasoning, limit Datalog imposes a linearity condition restricting the use of multiplication in rules. In this paper, we study the complexity and expressive power of limit Datalog programs extended with disjunction in the heads of rules and non-monotonic negation under the stable model semantics. We show that allowing for unrestricted use of negation leads to undecidability of reasoning. Decidability can be restored by stratifying the use of negation over predicates carrying numeric values. We show that the resulting language is  $\Pi_2^{\text{EXP}}$ -complete in combined complexity and that it captures  $\Pi_2^{\text{P}}$  over ordered structures in the sense of descriptive complexity. We also provide a study of several fragments of this language: we show that the complexity and expressive power of the full language are already reached for disjunction-free programs; furthermore, we show that semi-positive disjunctive programs are coNEXP-complete and that they capture coNP.

## 1 Introduction

Declarative languages underpinning existing rule-based reasoning engines for data analysis, such as BOOM (Alvaro et al. 2010), DeALS (Yang, Shkapsky, and Zaniolo 2017), Myria (Wang, Balazinska, and Halperin 2015), So-ciaLite (Seo, Guo, and Lam 2015), Overlog (Loo et al. 2009), Dyna (Eisner and Filardo 2011), and Yedalog (Chin et al. 2015), can be seen as extensions of Datalog where rules are equipped with arithmetic functions and aggregation to capture quantitative aspects of the data. These languages are, however, trivially undecidable due to the interaction between recursion and numeric computations in rules.

Kaminski et al. [2017; 2018] have recently proposed the language of *limit Datalog programs*—a decidable variant of Datalog equipped with stratified negation and arithmetic

functions over the integers that is expressive enough to capture the core features of existing declarative languages for data analysis. In limit Datalog programs, all intensional predicates with a numeric argument are limit predicates that keep maximal (or minimal) bounds on numeric values. For example, if we encode a directed graph with weighted edges using a ternary predicate *edge*, then rules (1) and (2), with *dist* a min limit predicate, compute the cost of a shortest path from a source node  $a_s$  to each other node in the graph.

$$\rightarrow \text{dist}(a_s, 0) \quad (1)$$

$$\text{dist}(x, m) \wedge \text{edge}(x, y, n) \rightarrow \text{dist}(y, m + n) \quad (2)$$

Intuitively, rule (2) says that, if  $x$  is reachable from  $a_s$  with cost at most  $m$  and  $(x, y)$  is an edge of cost  $n$ , then  $y$  is reachable from  $a_s$  with cost at most  $m + n$ . If the rules and a dataset entail a fact  $\text{dist}(a, \ell)$ , then the length of a shortest path from  $a_s$  to  $a$  is at most  $\ell$ ; so,  $\text{dist}(a, k)$  holds for each  $k \geq \ell$  since the cost of a shortest path is also at most  $k$ .

To ensure decidability, limit Datalog imposes a linearity condition restricting the use of multiplication. Under this restriction, fact entailment for semi-positive limit programs is coNEXP-complete in combined and coNP-complete in data complexity (Kaminski et al. 2017), whereas for programs with stratified negation fact entailment is  $\Delta_2^{\text{EXP}}$ -complete and  $\Delta_2^{\text{P}}$ -complete, respectively (Kaminski et al. 2018).

In this paper, we study limit Datalog programs equipped with disjunction in the head of rules and non-monotonic negation in the body. We first observe that allowing for unrestricted use of negation as failure leads to undecidability of reasoning, which holds even for disjunction-free programs satisfying the linearity condition. Thus, to obtain decidability we restrict the use of negation: while still allowing unrestricted application of negation to atoms without a numeric argument, we require that negation over atoms with numeric arguments is stratified. We call the resulting programs *numerically stratified*. The resulting language extends both stratified limit-linear Datalog (Kaminski et al. 2018) and disjunctive Datalog with negation under the stable model certain semantics (Eiter, Gottlob, and Mannila 1997).

We show that fact entailment for numerically stratified programs is  $\Pi_2^{\text{EXP}}$ -complete in combined and  $\Pi_2^{\text{P}}$ -complete in data complexity, where the lower bounds hold already

for disjunctive Datalog without arithmetic (Eiter, Gottlob, and Mannila 1997); furthermore, numerically stratified programs capture  $\Pi_2^P$  in the sense of descriptive complexity. Thus, although addition and linear multiplication provide a great deal of flexibility for modelling, they do not lead to an increase in expressive power or in the complexity of reasoning. Moreover, the complexity and expressive power of numerically stratified programs do not depend on the presence of disjunction in rule heads, which is in contrast to Datalog without arithmetic (Eiter, Gottlob, and Mannila 1997).

We also study the natural fragments of numerically stratified disjunctive limit-linear Datalog. We show that reasoning in the semi-positive fragment is coNEXP-complete in combined and coNP-complete in data complexity, and that it captures coNP; so, reasoning in this fragment is no harder than reasoning over semi-positive limit-linear programs without disjunction (Kaminski et al. 2017) or semi-positive disjunctive programs without arithmetic (Eiter, Gottlob, and Mannila 1997). Extending semi-positive negation to stratified negation, however, makes reasoning  $\Pi_2^{\text{EXP}}$ - and  $\Pi_2^P$ -complete, respectively. This is in contrast to disjunction-free stratified programs, for which reasoning is  $\Delta_2^{\text{EXP}}$ - and  $\Delta_2^P$ -complete (Kaminski et al. 2018).

## 2 Preliminaries

We start with disjunctive Datalog $_{\mathbb{Z}}$ , which extends disjunctive Datalog (Eiter, Gottlob, and Mannila 1997) with arithmetic over integers  $\mathbb{Z}$ . We slightly extend the syntax of Kaminski et al. [2017; 2018] by allowing the special numeric term  $\infty$ , where, intuitively, an atom  $C(a, \infty)$  holds if  $C(a, k)$  holds for all  $k \in \mathbb{Z}$ . This syntactic extension does not affect any of the results in (Kaminski et al. 2017; 2018) but allows the language to detect and react to divergence of numeric values, a feature that is available in closely related languages (Ross and Sagiv 1997).

**Disjunctive Datalog $_{\mathbb{Z}}$  Syntax.** We assume countably infinite, mutually disjoint sets of *objects*, *object variables*, *numeric variables*, and *predicates*. Each predicate has a non-negative integer *arity*, and each position  $i \in [1, v]$  of a  $v$ -ary predicate is of *object* or *numeric sort*. We distinguish binary *comparison* predicates  $\leq$  and  $<$ , with both positions numeric, and all other *standard* predicates. An *object term* is an object or an object variable. A *numeric term* is an integer in  $\mathbb{Z}$ , a numeric variable, the special symbol  $\infty$ , or an expression 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 not mentioning  $\infty$ , and  $+$ ,  $-$  and  $\times$  are the usual *arithmetic functions*. A *constant* is an object or an integer (but not  $\infty$ ). A *standard atom* is an expression of the form  $A(t_1, \dots, t_v)$ , where  $A$  is a  $v$ -ary standard predicate and each  $t_i$  is a term matching the sort of position  $i$  of  $A$ . A *comparison atom* is an expression of the form  $(s_1 \leq s_2)$  or  $(s_1 < s_2)$ , where  $s_1$  and  $s_2$  are numeric terms different from  $\infty$ . We also use usual abbreviations such as  $(s_1 \geq s_2)$  for  $(s_2 \leq s_1)$  and  $(s_1 \doteq s_2)$  for  $(s_1 \leq s_2) \wedge (s_2 \leq s_1)$ .

A *rule*  $\rho$  is an expression of the form  $\varphi \rightarrow \psi$ , where the *body*  $\varphi$  of  $\rho$  is a conjunction of *literals*—that is, standard atoms, negations  $\neg \alpha$  of standard atoms  $\alpha$ , and comparison atoms,—while the *head*  $\psi$  of  $\rho$  is a non-empty disjunction of

standard atoms; all the variables in  $\rho$  are implicitly universally quantified. Without loss of generality, we will consider only *safe* rules—that is, rules where all variables occur in positive body literals. A *pseudofact* is a disjunction-free rule with empty body where all terms are constants or  $\infty$ ; a *fact* is pseudofact without  $\infty$ . A standard predicate is *intensional* (IDB) in a program  $\mathcal{P}$  if it occurs in  $\mathcal{P}$  in the head of a rule that is not a pseudofact; otherwise, it is *extensional* (EDB). A (disjunctive Datalog $_{\mathbb{Z}}$ ) *program* is a finite set rules, and a *dataset* is a program consisting of facts.

A *stratification* of a program  $\mathcal{P}$  is a function  $\Lambda$  from the standard predicates of  $\mathcal{P}$  to positive natural numbers such that, for every rule in  $\mathcal{P}$ , we have  $\Lambda(A_1) = \Lambda(A_2)$  for each two predicates  $A_1, A_2$  in the head, and, for each head predicate  $A$  and each standard body literal  $\lambda$  over predicate  $B$ ,  $\Lambda(B) < \Lambda(A)$  if  $\lambda$  a negative and  $\Lambda(B) \leq \Lambda(A)$  otherwise.

A program is *positive* if it does not use negation, *semi-positive* if it allows negation only in front of EDB predicates, and *stratified* if it admits a stratification. A program is *disjunction-free* if it does not use disjunction (i.e., every rule has only one atom in the head).

**Disjunctive Datalog $_{\mathbb{Z}}$  Semantics.** An expression (e.g., term, atom, or rule)  $E'$  is an *instance* of  $E$  if  $E' = E\sigma$  for a sort-compatible substitution  $\sigma$ . Substitution  $\sigma$  is a *grounding* of  $E$  if  $E'$  is *ground*—that is, mentions no variables.

A (*Herbrand*) *interpretation*  $\mathcal{I}$  is a set of facts. It *satisfies* a ground literal  $\lambda$ , written  $\mathcal{I} \models \lambda$ , if one of the following holds (under the usual semantics of integer arithmetic):

- $\lambda$  is a standard atom and  $\mathcal{I}$  contains each fact obtained from  $\lambda$  by replacing every occurrence of  $\infty$  by an integer and evaluating all the numeric terms;
- $\lambda$  is a negative literal  $\neg \alpha$  and  $\mathcal{I} \not\models \alpha$ ; and
- $\lambda$  is a comparison atom that evaluates to *true*.

The notion of satisfaction extends to conjunctions of ground literals, rules, and sets of rules as in first-order logic. An interpretation  $\mathcal{I}$  is a *model* of a set  $\mathcal{Q}$  of rules if  $\mathcal{I} \models \mathcal{Q}$ . A model  $\mathcal{I}$  of  $\mathcal{Q}$  is *minimal* if  $\mathcal{Q}$  has no model  $\mathcal{I}'$  with  $\mathcal{I}' \subset \mathcal{I}$ .

A *Gelfond-Lifschitz reduct* of a program  $\mathcal{P}$  over an interpretation  $\mathcal{I}$  is the set  $\mathcal{P}^{\mathcal{I}}$  of positive rules obtained from the set of all ground instances of rules in  $\mathcal{P}$  by

- removing all rules with  $\neg \alpha$  in the body for  $\alpha \in \mathcal{I}$ ;
- removing all negative literals from the rest of the rules.

An interpretation  $\mathcal{I}$  is a *stable model* of a program  $\mathcal{P}$  if  $\mathcal{I}$  is a minimal model of  $\mathcal{P}^{\mathcal{I}}$ . Every stable model of a program  $\mathcal{P}$  is also a minimal model of  $\mathcal{P}$ ; if  $\mathcal{P}$  is positive, then the opposite also holds (Przymusiński 1991). Every stratified and disjunction-free program has a unique stable model (Przymusiński 1991), sometimes called *materialisation*, which can be constructed by computing (possibly transfinite) fix-points stratum by stratum (Apt, Blair, and Walker 1988; Dantsin et al. 2001; Kaminski et al. 2018).

A program  $\mathcal{P}$  (*certainly*) *entails* a fact  $\gamma$ , written  $\mathcal{P} \models \gamma$ , if  $\gamma \in \mathcal{I}$  for every stable model  $\mathcal{I}$  of  $\mathcal{P}$ .

**Computational and Descriptive Complexity.** We assume standard definitions of basic complexity classes such as P, NP, coNP, and EXP, as well as polynomial hierarchy classes  $\Delta_i^P$ ,  $\Sigma_i^P$ , and  $\Pi_i^P$ , for  $i \geq 0$ , and their counterparts in the weak

exponential hierarchy  $\Delta_i^{\text{EXP}}$ ,  $\Sigma_i^{\text{EXP}}$ , and  $\Pi_i^{\text{EXP}}$  (Papadimitriou 1994; Ladner and Lynch 1976; Hemachandra 1989).

We study decidability and complexity of *fact entailment*—that is, the problem of checking whether  $\mathcal{P} \models \gamma$  for a program  $\mathcal{P}$  and fact  $\gamma$ —for disjunctive Datalog<sub>ℤ</sub> and its fragments. *Combined complexity* assumes that the input is both  $\mathcal{P}$  and  $\gamma$ , while *data complexity* assumes that  $\mathcal{P} = \mathcal{P}' \cup \mathcal{D}$  for  $\mathcal{P}'$  a program and  $\mathcal{D}$  a dataset, and that only  $\mathcal{D}$  and  $\gamma$  form the input while  $\mathcal{P}'$  is fixed. Our results do not depend on whether integers are coded in unary or in binary.

It is known that, without numbers and arithmetic, fact entailment for disjunctive Datalog<sub>ℤ</sub> is  $\Pi_2^{\text{EXP}}$ -complete in combined and  $\Pi_2^{\text{P}}$ -complete in data complexity, where the lower bounds already hold for stratified programs (Eiter, Gottlob, and Mannila 1997); also, for disjunction-free programs the problem is coNEXP- and coNP-complete, respectively (Kolaitis and Papadimitriou 1991; Schlipf 1995), and the same holds if disjunction is allowed, but only positive or semi-positive programs are considered (Chandra and Harel 1985; Eiter, Gottlob, and Mannila 1997); finally, if no disjunction and only stratified negation or no negation is allowed, then it is EXP- and P-complete, respectively (Dantsin et al. 2001). However, it is undecidable for positive programs with arithmetic allowing only for function  $+$  and standard predicates with at most one numeric position (Kaminski et al. 2017).

Descriptive complexity studies the relationship between the expressive power of logical languages and computational complexity (Immerman 1999; Grädel 2007). In this setting, a *logical language* is a set of *sentences* each of which evaluates on a dataset to either *true* or *false*. As is customary, we concentrate on the family  $\mathcal{D}$  of datasets over predicates without numeric positions that are *ordered*—that is, mention at least two objects and contain facts  $\text{first}(a_1)$ ,  $\text{next}(a_1, a_2)$ ,  $\dots$ ,  $\text{next}(a_{c-1}, a_c)$ ,  $\text{last}(a_c)$  over special standard predicates  $\text{first}$ ,  $\text{next}$ , and  $\text{last}$  for an enumeration  $a_1, \dots, a_c$  of all objects in the dataset; also, these facts are the only facts over these predicates in the dataset. A logical language  $L$  *captures* a complexity class  $C$  (over *ordered datasets*) if the following holds for each signature (i.e., set of predicates)  $\Sigma$  containing the special predicates:

- the problem of evaluating a sentence  $\varphi \in L$  on a dataset  $\mathcal{D} \in \mathcal{D}$  over  $\Sigma$  is in  $C$  if  $\varphi$  is considered fixed;
- for each problem  $P$  in  $C$ , there is a sentence  $\varphi_P \in L$  such that, for every  $\mathcal{D} \in \mathcal{D}$  over  $\Sigma$ ,  $\varphi_P$  is *true* on  $\mathcal{D}$  if and only if (the appropriate binary encoding of)  $\mathcal{D}$  is in  $P$ .

When applying these definitions to disjunctive Datalog<sub>ℤ</sub>, we define a program  $\mathcal{P}$  to be *true* on a dataset  $\mathcal{D} \in \mathcal{D}$  over EDB predicates in  $\mathcal{P}$  if  $\mathcal{P} \cup \mathcal{D} \models \text{goal}$ , where *goal* is a distinguished nullary predicate (note that  $\mathcal{P}$  can use IDB predicates with numeric positions, which are not in the signature in the above sense). If a fragment  $L$  of disjunctive Datalog<sub>ℤ</sub> captures a class  $C$ , then fact entailment for  $L$  is  $C$ -hard in data complexity (Eiter, Gottlob, and Mannila 1997).

It is known that, without arithmetic, full and stratified disjunctive Datalog<sub>ℤ</sub> capture  $\Pi_2^{\text{P}}$  (Eiter, Gottlob, and Mannila 1997), the disjunction-free and semi-positive fragments capture coNP (Kolaitis and Papadimitriou 1991; Schlipf 1995), while programs that are disjunction-free and either semi-positive or stratified capture P (Dantsin et al. 2001).

### 3 Limit-Linear Disjunctive Datalog<sub>ℤ</sub>

As outlined in the previous section, allowing even a very restrictive form of arithmetic in disjunction-free positive Datalog<sub>ℤ</sub> makes fact entailment undecidable. To recover decidability, Kaminski et al. [2017] suggested the limit-linear fragment of disjunction-free positive Datalog<sub>ℤ</sub>, for which the problem is coNEXP-complete in combined and coNP-complete in data complexity. Moreover, the complexity stays the same for the semi-positive fragment, and grows to  $\Delta_2^{\text{EXP}}$  and  $\Delta_2^{\text{P}}$ , respectively, if stratified negation is allowed (Kaminski et al. 2018).<sup>1</sup> It also follows from the results of Kaminski et al. that semi-positive and stratified disjunction-free Datalog<sub>ℤ</sub> capture coNP and  $\Delta_2^{\text{P}}$ , respectively. In this section, we introduce an extension of the limit-linear language with disjunction and stable models negation.

The key feature of limit programs is that IDB predicates are partitioned into *object* and *limit* predicates: the former have only object positions, while each of the latter has a distinguished numeric position that keeps a limit (i.e., a maximal or minimal bound) on integers. As the language of Kaminski et al. [2017], limit and limit-linear disjunctive Datalog<sub>ℤ</sub> can be seen as either semantic or syntactic restrictions of disjunctive Datalog<sub>ℤ</sub>. We start with the more intuitive semantics-based definitions.

**Definition 1.** A limit program is a pair  $(\mathcal{P}, \tau)$  such that

- $\mathcal{P}$  is a disjunctive Datalog<sub>ℤ</sub> program where each standard predicate is either object with all positions object, or numeric with the last position numeric and all others object;
- $\tau$  is a partial function from numeric predicates in  $\mathcal{P}$  to  $\{\max, \min\}$  that is total on the numeric IDB predicates and the numeric predicates of atoms mentioning  $\infty$  in  $\mathcal{P}$ .

The numeric predicates of  $(\mathcal{P}, \tau)$  in the domain of  $\tau$  are *limit* and all other numeric predicates are *exact*. A limit predicate  $C$  is *max* or *min* if  $\tau(C)$  is *max* or *min*, respectively.

All notions on predicates in Definition 1 propagate to atoms, literals, etc. All syntactic notions defined on regular programs (e.g., EDB and semi-positive) extend to a limit program  $(\mathcal{P}, \tau)$  by applying them to  $\mathcal{P}$ . For  $C$  a limit predicate, we write  $\preceq_C^\tau$ ,  $\prec_C^\tau$  and  $\max_C^\tau$  for  $\leq$ ,  $<$  and  $\max$ , respectively, if  $\tau(C) = \max$ , and for  $\geq$ ,  $>$  and  $\min$ , otherwise.

We move on to the semantics of limit programs.

**Definition 2.** An interpretation  $\mathcal{I}$  is *limit-closed* for a limit program  $(\mathcal{P}, \tau)$  if for every limit fact  $C(\mathbf{a}, k) \in \mathcal{I}$  we also have  $C(\mathbf{a}, k') \in \mathcal{I}$  for each integer  $k' \preceq_C^\tau k$ . An interpretation is a model of  $(\mathcal{P}, \tau)$  if it is limit-closed and is a model of  $\mathcal{P}$ . (Certain) entailment of a fact by a limit program  $(\mathcal{P}, \tau)$  is then defined in the same way as for ordinary disjunctive Datalog<sub>ℤ</sub> programs.<sup>2</sup>

Having the direct semantics at hand, we next argue that limit programs can be easily rewritten to regular Datalog<sub>ℤ</sub> programs. Indeed, the semantics of limit predicates in a limit

<sup>1</sup>The  $\Delta_2^{\text{EXP}}$  bounds are not explicitly claimed in (Kaminski et al. 2018), but follow by adapting their data complexity proofs.

<sup>2</sup>This definition applied to stratified limit Datalog<sub>ℤ</sub> is different from the definition of Kaminski et al. [2018]; however, the two definitions can be shown equivalent.

program  $(\mathcal{P}, \tau)$  can be equivalently axiomatised using the regular program  $\mathcal{P} \cup \mathcal{A}(\tau)$ , for  $\mathcal{A}(\tau)$  the set of rules

$$C(\mathbf{x}, m) \wedge (n \preceq_C^\tau m) \rightarrow C(\mathbf{x}, n),$$

for each predicate  $C$  in the domain of  $\tau$  and a tuple  $\mathbf{x}$  of distinct object variables of appropriate size. In particular, an interpretation is a stable model of  $\mathcal{P} \cup \mathcal{A}(\tau)$  if and only if it is a stable model of  $(\mathcal{P}, \tau)$ .

Limit programs have an intuitive semantics and allow to declaratively encode many tasks involving arithmetic, such as shortest path and company control (Kaminski et al. 2017). However, due to the presence of multiplication, even disjunction-free positive Datalog<sub>ℤ</sub> already has undecidable fact entailment. Kaminski et al. [2017] recover decidability by imposing an additional linearity restriction, which we next generalise to programs with disjunction and negation.<sup>3</sup>

If  $C$  is a limit predicate,  $\mathbf{t}$  is a tuple of object terms, and  $s$  is a numeric term different from  $\infty$ , then a *least upper bound (LUB)* expression  $\lceil C(\mathbf{t}, s) \rceil$  is an abbreviation for the conjunction  $C(\mathbf{t}, s) \wedge \text{not } C(\mathbf{t}, s + k)$ , where  $k = 1$  if  $C$  is max and  $k = -1$  otherwise.

**Definition 3.** A numeric variable  $m$  is guarded in a rule of a limit program if it occurs in the rule body either in a function-free positive exact literal or in an LUB expression of the form  $\lceil C(\mathbf{t}, m) \rceil$ . Such a rule is *limit-linear (LL-rule)* if at most one argument of each multiplication in the rule uses unguarded variables. A limit program is *limit-linear (LL-program)* if so is each of its rules.

LL-programs are our main object of study in this paper. In what follows, we investigate their computational properties and expressive power. To avoid notational clutter, we write a limit program  $(\mathcal{P}, \tau)$  as just  $\mathcal{P}$ , assuming that  $\tau$  is given implicitly. We also omit  $\tau$  in notations such as  $\preceq_C^\tau$  and, when considering a union of limit programs, assume that their respective  $\tau$  coincide on the shared predicates.

Next, we demonstrate the power of limit-linear disjunctive Datalog<sub>ℤ</sub> by encoding (the complement of) the Knapsack problem. Note that this encoding does not require a constant for the value of every possible solution, unlike the encoding of the problem in disjunctive Datalog without arithmetic by Eiter, Gottlob, and Mannila [1997].

**Example 4.** An instance  $K = \langle c, \{(w_1, v_1), \dots, (w_n, v_n)\} \rangle$  of the 0-1 knapsack problem consists of a capacity  $c$  and a set of items with weights  $w_i$  and values  $v_i$ , where  $c$ ,  $w_i$ , and  $v_i$  are positive integers. A solution  $S$  to  $K$  with value  $\sum_{i \in S} v_i$  and weight  $w = \sum_{i \in S} w_i$  is a subset of  $\{1, \dots, n\}$  with  $w \leq c$ . Such an instance  $K$  is naturally represented as an ordered dataset  $\mathcal{D}_K$  using an exact unary fact  $\text{capacity}(c)$ , and exact binary facts  $\text{weight}(a_i, w_i)$  and  $\text{value}(a_i, v_i)$  where constants  $a_i$  represent the items. For  $B$  a binary exact predicate and  $\text{op} \in \{\text{max}, \text{min}\}$ , let the positive LL-program  $\mathcal{P}_{\text{op}}^B$  be defined as follows, where *in* and *out* are unary object

predicates, and  $s_B$  and  $\text{sum}_B$  are limit predicates of type op.

$$\begin{aligned} \text{first}(x) \wedge \text{out}(x) &\rightarrow s_B(x, 0) \\ \text{first}(x) \wedge \text{in}(x) \wedge B(x, n) &\rightarrow s_B(x, n) \\ \text{next}(y, x) \wedge s_B(y, m) \wedge \text{out}(x) &\rightarrow s_B(x, m) \\ \text{next}(y, x) \wedge s_B(y, m) \wedge \text{in}(x) \wedge B(x, n) &\rightarrow s_B(x, m + n) \\ \text{last}(x) \wedge s_B(x, m) &\rightarrow \text{sum}_B(m) \end{aligned}$$

Assuming that predicates *in* and *out* partition the objects of a dataset  $\mathcal{D}$  and that predicate  $B$  is uniquely defined on all such objects, program  $\mathcal{P}_{\text{op}}^B \cup \mathcal{D}$  defines predicate  $\text{sum}_B$  to have the limit value  $\sum \{k \mid \exists a : \{\text{in}(a), B(a, k)\} \subseteq \mathcal{D}\}$ .

Let the positive LL-program  $\mathcal{P}_{ks}$  be then defined as the extension of  $\mathcal{P}_{\text{max}}^{\text{weight}} \cup \mathcal{P}_{\text{min}}^{\text{value}}$  with the following rules, where *no-sol* is a unary min predicate.

$$\begin{aligned} \text{first}(x) &\rightarrow \text{in}(x) \vee \text{out}(x) \\ \text{next}(y, x) &\rightarrow \text{in}(x) \vee \text{out}(x) \\ \text{sum}_{\text{weight}}(m) \wedge \text{capacity}(n) \wedge (n < m) &\rightarrow \text{no-sol}(0) \\ \text{sum}_{\text{value}}(m) &\rightarrow \text{no-sol}(m + 1) \end{aligned}$$

Program  $\mathcal{P}_{ks}$  encodes the 0-1 knapsack problem in the sense that  $\mathcal{P}_{ks} \cup \mathcal{D}_K \models \text{no-sol}(k)$  if and only if all solutions of an instance  $K$  have values less than  $k$ .  $\triangleleft$

Note that the program in Example 4 is positive. To demonstrate the use of negation and the symbol  $\infty$ , let us extend the example in Section 1 to a program classifying graph nodes according to whether or not they are reachable from a distinguished source node via a path with a negative cost cycle.

**Example 5.** Let a directed graph  $G$  with possibly negatively weighted edges be encoded as in Section 1. Let program  $\mathcal{P}_{nc}$  consist of rules (1) and (2) from Section 1 as well as the following rules, where *dist* is a min predicate.

$$\begin{aligned} \text{dist}(x, \infty) &\rightarrow \text{neg-cycle}(x) & (3) \\ \text{not } \text{dist}(x, \infty) &\rightarrow \text{no-neg-cycle}(x) & (4) \end{aligned}$$

For a dataset  $\mathcal{D}_G$  encoding  $G$ ,  $\mathcal{P}_{nc} \cup \mathcal{D}_G \models \text{neg-cycle}(a)$  holds if and only if  $a$  is reachable from the distinguished source node  $a_s$  in  $G$  via a cycle whose edges sum to a negative value, while  $\mathcal{P}_{nc} \cup \mathcal{D}_G \models \text{no-neg-cycle}(a)$  holds if and only if  $a$  is not reachable from  $a_s$  via such a path.  $\triangleleft$

As stratified negation can be used to compute shortest paths in the absence of negative cycles (Kaminski et al. 2018), it is possible to further extend Example 5 to a program computing shortest paths in arbitrary graphs.

## 4 Numeric Stratification

We next observe that fact entailment for LL-programs is undecidable, and, furthermore, undecidability holds even for disjunction-free programs. Since fact entailment for stratified disjunction-free LL-programs is decidable (Kaminski et al. 2018), this implies that undecidability of our full language is caused by unrestricted use of negation as failure. By contrast, the addition of unrestricted negation under the stable model semantics to (disjunctive) Datalog without arithmetic does not affect decidability but only complexity of reasoning (Eiter, Gottlob, and Mannila 1997).

<sup>3</sup>Our generalisation is equivalent to the one in (Kaminski et al. 2018) on stratified disjunction-free programs.

**Theorem 6.** *The fact entailment problem for disjunction-free LL-programs is undecidable.*

The theorem is shown by a reduction of (the complement of) Hilbert’s tenth problem, which exploits the fact that programs with unrestricted negation may have stable models with arbitrarily big numeric values for limit predicates.

Thus, to achieve decidability, we need to restrict the use of negation. We do so by adopting a new form of stratification, called *numeric stratification*, that relaxes the usual notion of stratification as familiar from arithmetic-free programs and studied by Kaminski et al. [2018].

**Definition 7.** A numeric stratification  $\Lambda$  of a program  $\mathcal{P}$  is defined the same as usual stratification except that  $\Lambda(B) < \Lambda(A)$  is required for a head predicate  $A$  and the predicate  $B$  of a negative body literal  $\lambda$  only if  $B$  is numeric (and  $\Lambda(B) \leq \Lambda(A)$  if  $B$  is object). A program  $\mathcal{P}$  is numerically stratified if it admits a numeric stratification  $\Lambda$ . The  $i$ -th stratum  $\mathcal{P}_\Lambda^i$  of  $\mathcal{P}$  with respect to  $\Lambda$  is a subset of  $\mathcal{P}$  consisting of all rules with  $\Lambda(A) = i$  for all head predicates  $A$ .

Note that numeric stratification only disallows cyclic dependencies between numeric predicates; in particular, limit programs admitting such a stratification strictly extend disjunctive Datalog programs without arithmetic but with unrestricted use of negation as studied by Eiter, Gottlob, and Mannila [1997]. We next demonstrate the power of numerically stratified programs by encoding  $\forall\exists$ SAT, a prototypical decision problem complete for  $\Pi_2^P$ .

**Example 8.** Let the program  $\mathcal{P}_{\forall\exists}$  be defined as follows, where  $ass$ ,  $F'$  and  $T'$  are max predicates and all other numeric predicates are exact.

$$uvar(x) \wedge \text{not } T(x) \rightarrow F(x) \quad (5)$$

$$uvar(x) \wedge \text{not } F(x) \rightarrow T(x) \quad (6)$$

$$\rightarrow ass(0) \quad (7)$$

$$root(x) \wedge F'(x, n) \rightarrow ass(n + 1) \quad (8)$$

$$\begin{aligned} & ass(n) \wedge shift(x, k) \wedge \\ & (0 \leq m_1) \wedge (0 \leq m_2 < k) \wedge \\ & (n \doteq 2 \times k \times m_1 + k + m_2) \rightarrow T'(x, n) \end{aligned} \quad (9)$$

$$\begin{aligned} & ass(n) \wedge shift(x, k) \wedge \\ & (0 \leq m_1) \wedge (0 \leq m_2 < k) \wedge \\ & (n \doteq 2 \times k \times m_1 + m_2) \rightarrow F'(x, n) \end{aligned} \quad (10)$$

$$ass(n) \wedge T(x) \rightarrow T'(x, n) \quad (11)$$

$$ass(n) \wedge F(x) \rightarrow F'(x, n) \quad (12)$$

$$\text{not}(x, y) \wedge F'(y, n) \rightarrow T'(x, n) \quad (13)$$

$$\text{not}(x, y) \wedge T'(y, n) \rightarrow F'(x, n) \quad (14)$$

$$\text{or}(x, y, z) \wedge F'(y, n) \wedge F'(z, n) \rightarrow F'(x, n) \quad (15)$$

$$\text{or}(x, y, z) \wedge T'(y, n) \rightarrow T'(x, n) \quad (16)$$

$$\text{or}(x, y, z) \wedge T'(z, n) \rightarrow T'(x, n) \quad (17)$$

$$maxAss(n) \wedge \text{not } ass(n + 1) \rightarrow sat \quad (18)$$

For  $\psi = \forall x_1 \dots x_v \exists y_1 \dots y_u \varphi$  with  $\varphi$  a propositional formula over variables  $x_1, \dots, x_v, y_1, \dots, y_u$  using disjunction and negation, let a dataset  $\mathcal{D}_\psi$  consist of the following

facts, where  $a_\chi$  is a fresh object for each subformula  $\chi$  of  $\varphi$ :

$$uvar(a_{x_j}), \quad \text{for each } j \in [1, v],$$

$$maxAss(2^u - 1),$$

$$shift(a_{y_i}, 2^{i-1}), \quad \text{for each } i \in [1, u],$$

$$root(a_\varphi),$$

$$\text{or}(a_\chi, a_{\chi_1}, a_{\chi_2}), \quad \text{for each subformula } \chi = \chi_1 \vee \chi_2 \text{ of } \varphi,$$

$$\text{not}(a_\chi, a_{\chi_1}), \quad \text{for each subformula } \chi = \neg \chi_1 \text{ of } \varphi.$$

One can show that  $\mathcal{P}_{\forall\exists} \cup \mathcal{D}_\psi \models sat$  if and only if  $\psi$  holds. Intuitively, rules (5) and (6) perform a ‘universal guess’ of truth values for  $x_1, \dots, x_v$  using non-stratified negation over object predicates, after which the remaining rules search for an assignment of  $y_1, \dots, y_u$  that, together with the guessed values for  $x_1, \dots, x_v$ , will make  $\varphi$  true. Each truth assignment of  $y_1, \dots, y_u$  can be seen as an integer of length  $u$  in binary representation and stored in this form in predicate  $ass$ . So, rules (7) and (8) iterate through all integers  $\ell$  representing assignments of  $y_1, \dots, y_u$  until  $\varphi$  evaluates to true; the latter is checked by rules (9)–(17) computing the value of  $\varphi$  for a given  $\ell$  as a fact  $T'(a_\varphi, \ell)$  or  $F'(a_\varphi, \ell)$ . Finally, rule (18) uses (stratified) negation to detect whether a satisfying assignment has been found.  $\triangleleft$

Note that formula  $\psi$  in Example 8 is encoded as a dataset and that the program does not use disjunction. Thus, the example implies that fact entailment for disjunction-free numerically stratified LL-programs is  $\Pi_2^P$ -hard in data complexity. This is in contrast to Datalog without arithmetic, where the problem is coNP-complete (Eiter, Gottlob, and Mannila 1997).

## 5 Complexity Upper Bounds

In this section, we establish the upper bounds on the complexity of fact entailment for semi-positive and numerically stratified limit-linear disjunctive Datalog<sub>Z</sub>. We begin by showing that for semi-positive LL-programs, the problem is in coNEXP in combined and in coNP in data complexity. Building on this result, we then show that for numerically stratified LL-programs, the problem is in  $\Pi_2^{\text{EXP}}$  in combined and in  $\Pi_2^P$  in data complexity. In the next section, we will see that both bounds are tight and consider implications on other fragments of limit-linear disjunctive Datalog<sub>Z</sub>.

To establish the complexity bounds, we build on the machinery developed for disjunction-free Datalog<sub>Z</sub> by Kaminski et al. [2017; 2018]. We start with a definition of OG-grounding, extending a similar notion by Kaminski et al.<sup>4</sup>

**Definition 9.** An LL-program is object-and-guarded-ground (OG-ground) if none of its rules have object or guarded numeric variables. The canonical OG-grounding  $\mathcal{G}(\mathcal{P})$  of an LL-program  $\mathcal{P}$  is the OG-ground program that consists of all OG-ground instances  $\rho\sigma$  of rules  $\rho$  in  $\mathcal{P}$  with  $\sigma$  a substitution mapping all object and guarded numeric variables of  $\rho$  to constants mentioned in  $\mathcal{P}$ .

The size of the OG-grounding is exponentially bounded in the size of the original program; however, the bound is

<sup>4</sup>See ‘semi-ground(ing)’ in (Kaminski et al. 2018).

polynomial in the size of the data component of the program. The OG-grounding is equivalent to the original program in the following sense.

**Lemma 10.** *A limit-closed interpretation is a stable model of an LL-program  $\mathcal{P}$  if and only if it is a stable model of the OG-grounding  $\mathcal{G}(\mathcal{P})$ .*

Next, we define OG-reducts, which lift Gelfond-Lifschitz reducts to OG-ground programs with arithmetic. For this, we first introduce a natural finite representation of limit-closed interpretations over a finite number of objects.

**Definition 11.** *A pseudointerpretation is a set  $\mathcal{J}$  of pseudo-facts such that  $\ell_1 = \ell_2$  for all limit pseudofacts  $C(\mathbf{a}, \ell_1)$  and  $C(\mathbf{a}, \ell_2)$  in  $\mathcal{J}$ . A limit-closed interpretation  $\mathcal{I}$  and a pseudointerpretation  $\mathcal{J}$  correspond to each other if the following holds for each predicate  $A$  and each tuple of objects  $\mathbf{a}$  of appropriate size:*

- for  $A$  object, we have  $A(\mathbf{a}) \in \mathcal{I}$  if and only if  $A(\mathbf{a}) \in \mathcal{J}$ ;
- for  $A$  exact and  $k \in \mathbb{Z}$ , we have  $A(\mathbf{a}, k) \in \mathcal{I}$  if and only if  $A(\mathbf{a}, k) \in \mathcal{J}$ ;
- for  $A$  limit and  $\ell \in \mathbb{Z}$ , we have  $A(\mathbf{a}, k) \in \mathcal{I}$  for all  $k \preceq_A \ell$  and  $A(\mathbf{a}, k) \notin \mathcal{I}$  for all  $k \succ_A \ell$  if and only if  $A(\mathbf{a}, \ell) \in \mathcal{J}$ ;
- for  $A$  limit, we have  $A(\mathbf{a}, k) \in \mathcal{I}$  for all  $k \in \mathbb{Z}$  if and only if  $A(\mathbf{a}, \infty) \in \mathcal{J}$ .

As shown in (Kaminski et al. 2017), the correspondence relation in Definition 11 is a bijection, so we may use pseudointerpretations and limit-closed interpretations interchangeably. A pseudointerpretation is a *pseudomodel* of an LL-program  $\mathcal{P}$  if its corresponding interpretation is a model of  $\mathcal{P}$ ; a pseudomodel is *minimal* or *stable* if so is the corresponding model.

**Definition 12.** *The OG-reduct  $\mathcal{P}_{\text{OG}}^{\mathcal{J}}$  of an LL-program  $\mathcal{P}$  over a pseudointerpretation  $\mathcal{J}$  is the OG-ground positive program obtained from  $\mathcal{G}(\mathcal{P})$  by modifying it as follows, for each negative literal  $\text{not } \alpha$  in each rule  $\rho$  in  $\mathcal{G}(\mathcal{P})$ :*

- for  $\alpha$  an object atom,
  - if  $\alpha \in \mathcal{J}$ , then delete  $\rho$ ;
  - otherwise, delete  $\text{not } \alpha$  from  $\rho$ ;
- for  $\alpha = B(\mathbf{a}, s)$  exact, consider all  $k_1 < \dots < k_h$  with  $B(\mathbf{a}, k_i) \in \mathcal{J}$  for each  $i \in [1, h]$  and
  - if  $h > 0$ , replace  $\rho$  with  $h + 1$  rules obtained from  $\rho$  by replacing  $\text{not } \alpha$  with comparison atoms ( $s < k_1$ ), ( $k_{i-1} < s < k_i$ ) for  $i \in [2, h]$ , and ( $k_h < s$ );
  - otherwise, delete  $\text{not } \alpha$  from  $\rho$ ; and
- for  $\alpha = C(\mathbf{a}, s)$  limit,
  - if  $C(\mathbf{a}, \infty) \in \mathcal{J}$ , then delete  $\rho$ ;
  - otherwise, if  $s \neq \infty$  and there is some  $k \in \mathbb{Z}$  with  $C(\mathbf{a}, k) \in \mathcal{J}$ , then replace  $\text{not } \alpha$  in  $\rho$  with the comparison atom ( $k \prec_C s$ );
  - otherwise, delete  $\text{not } \alpha$  from  $\rho$ .

Similarly to OG-groundings, the size of the OG-reduct is exponentially bounded in the size of the original program, and polynomially in the size of the pseudointerpretation and the dataset component of the program. Moreover, the OG-reduct is also essentially equivalent to the original program.

---

#### ALGORITHM 1: Fact Non-Entailment

---

**Input:** numerically stratified LL-program  $\mathcal{P}$  and fact  $\gamma$

**Output:** *true* if and only if  $\mathcal{P} \not\models \gamma$

- 1 compute a numeric stratification  $\Lambda$  of  $\mathcal{P}$  with minimal number of strata
  - 2 guess a pseudomodel  $\mathcal{J}$  of  $\mathcal{P}$  satisfying the bounds of Lemma 17
  - 3 **return** *true* if  $\mathcal{J} \not\models \gamma$ ,  $\mathcal{J} \models \mathcal{P}_{\text{OG}}^{\mathcal{J}}$  and  $\mathcal{J}' \not\models \mathcal{P}_{\text{OG}}^{\mathcal{J}}$  for each  $\mathcal{J}' \sqsubset \mathcal{J}$ , and *false* otherwise
- 

**Lemma 13.** *A pseudointerpretation  $\mathcal{J}$  is a stable pseudomodel of an LL-program  $\mathcal{P}$  if and only if  $\mathcal{J}$  is a minimal pseudomodel of the OG-reduct  $\mathcal{P}_{\text{OG}}^{\mathcal{J}}$ .*

We are now ready to show the upper bounds for semi-positive LL-programs. To this end, note that for each stable pseudomodel  $\mathcal{J}$  of an LL-program  $\mathcal{P}$  and each EDB fact  $\gamma$ , we have  $\gamma \in \mathcal{J}$  if and only if  $\gamma \in \mathcal{P}$ . Thus, if  $\mathcal{P}$  is a semi-positive program,  $\mathcal{J}$  is a stable pseudomodel of  $\mathcal{P}$ , and  $\mathcal{D}$  is the set of all facts in  $\mathcal{P}$ , then  $\mathcal{P}_{\text{OG}}^{\mathcal{J}} = \mathcal{P}_{\text{OG}}^{\mathcal{D}}$  as all negative literals in  $\mathcal{P}$  must be EDB. Consequently, to show that  $\gamma$  is not entailed by a semi-positive  $\mathcal{P}$ , by Lemma 13, it suffices to find a minimal pseudomodel of  $\mathcal{P}_{\text{OG}}^{\mathcal{D}}$  that does not satisfy  $\gamma$ . For this, in turn, it suffices to find a pseudomodel  $\mathcal{J}$  of  $\mathcal{P}_{\text{OG}}^{\mathcal{D}}$  not satisfying  $\gamma$  (since  $\mathcal{P}_{\text{OG}}^{\mathcal{D}}$  is positive, it then follows that at least one minimal pseudomodel does not satisfy  $\gamma$ ). As shown in (Kaminski et al. 2017), for semi-positive disjunction-free  $\mathcal{P}$ , such a pseudomodel is essentially a solution to an encoding of  $\mathcal{P}_{\text{OG}}^{\mathcal{D}}$  and the negation of  $\gamma$  in Presburger arithmetic; moreover, using techniques from (von zur Gathen and Sieveking 1978; Chistikov and Haase 2016), the reduction provides a bound on the size of the pseudomodel that is exponential in the size of  $\mathcal{P}$  and polynomial in the size of  $\mathcal{D}$ , and hence the pseudomodel can be guessed and checked in coNEXP in general and in coNP in data complexity. It turns out that this approach can be extended to LL-programs with disjunction, yielding the following result.

**Theorem 14.** *The fact entailment problem for semi-positive LL-programs is in coNEXP in combined and in coNP in data complexity.*

Furthermore, from the bounds on the size of countermodels to fact entailment we can derive a bound on the size of all minimal models of a program, which can be stated for positive OG-ground programs as follows.

**Lemma 15.** *There are polynomials  $p_1$ ,  $p_2$ , and  $p_3$  such that each minimal pseudomodel  $\mathcal{J}$  of each positive OG-ground program  $\mathcal{P}$  satisfies  $|\mathcal{J}| \leq |\mathcal{P}|$  and the magnitudes of integers in  $\mathcal{J}$  are bounded by the following number, where  $b$  is the maximal magnitude of an integer in  $\mathcal{P}$ , and  $u$  is the maximal size of a rule in  $\mathcal{P}$  assuming that integers take unit size:*

$$p_1(b)^{p_2(|\mathcal{P}|) \cdot 2^{p_3(u)}}.$$

Lemma 15 allows us to bound the magnitude of numbers in stable pseudomodels of an LL-program  $\mathcal{P}$  provided we can bound the numbers occurring in  $\mathcal{P}_{\text{OG}}^{\mathcal{J}}$  for each stable

	disjunction-free	disjunctive
positive	coNEXP/ coNP (Kaminski et al. 2017)	coNEXP/ coNP [Theorem 19]
semi-positive	coNEXP/ coNP (Kaminski et al. 2018)	coNEXP/ coNP [Theorem 19]
stratified	$\Delta_2^{\text{EXP}} / \Delta_2^{\text{P}}$ (Kaminski et al. 2018)	$\Pi_2^{\text{EXP}} / \Pi_2^{\text{P}}$ [Theorem 19]
numerically stratified	$\Pi_2^{\text{EXP}} / \Pi_2^{\text{P}}$ [Corollary 21]	$\Pi_2^{\text{EXP}} / \Pi_2^{\text{P}}$ [Theorem 19]
unrestricted	undecidable [Theorem 6]	undecidable [Theorem 6]

Table 1: Complexity of fact entailment for fragments of limit-linear Datalog<sub>ℤ</sub> in the format ‘combined complexity / data complexity’ (all bounds are tight)

pseudomodel  $\mathcal{J}$  of  $\mathcal{P}$ . By an argument similar to the one leading to Theorem 14, we can do so if all negated numeric literals in  $\mathcal{P}$  are EDB, as is the case for each individual stratum of a numerically stratified LL-program; indeed, in this case, for each stable pseudomodel  $\mathcal{J}$  of  $\mathcal{P}$ , the reduct  $\mathcal{P}_{\text{OG}}^{\mathcal{J}}$  contains only numbers from  $\mathcal{P}$ .

The following lemma then allows us to compute stable pseudomodels of a numerically stratified LL-program by combining the stable pseudomodels of its strata.

**Lemma 16.** *For each numerically stratified LL-program  $\mathcal{P}$  with a numeric stratification  $\Lambda$  and each  $h \geq 1$ , a pseudointerpretation  $\mathcal{J}$  is a stable pseudomodel of  $\bigcup_{i=1}^h \mathcal{P}_{\Lambda}^i$  if and only if  $\mathcal{J}$  is a stable pseudomodel of  $\mathcal{P}_{\Lambda}^h \cup \mathcal{J}'$ , for  $\mathcal{J}'$  a stable pseudomodel of  $\bigcup_{i=1}^{h-1} \mathcal{P}_{\Lambda}^i$ .*

Combining the implications of Lemma 15 on the strata of numerically stratified programs with Lemma 16, we obtain the following bounds on the size and the magnitude of numbers occurring in stable pseudomodels.

**Lemma 17.** *There are polynomials  $p_1$ ,  $p_2$ , and  $p_3$  such that each stable pseudomodel  $\mathcal{J}$  of each numerically stratified LL-program  $\mathcal{P}$  satisfies  $|\mathcal{J}| \leq |\mathcal{P}_{\text{OG}}^{\mathcal{J}}|$ , and the magnitudes of integers in  $\mathcal{J}$  are bounded by the following number, where  $b$  is the maximal magnitude of an integer in  $\mathcal{P}$ ,  $h$  is the minimal number of non-empty strata over all numeric stratifications of  $\mathcal{P}$ , and  $u$  is the maximal size of a rule in  $\mathcal{P}$  assuming that all integers take unit size:*

$$p_1(b)^{h \cdot p_2(|\mathcal{P}_{\text{OG}}^{\mathcal{J}}|) \cdot 2^{p_3(u)}}.$$

Recall now that all exact atoms in a numerically stratified LL-program  $\mathcal{P}$  are EDB, and all EDB atoms in every stable pseudomodel  $\mathcal{J}$  of  $\mathcal{P}$  come from  $\mathcal{P}$ . So,  $|\mathcal{P}_{\text{OG}}^{\mathcal{J}}|$  can be bounded independently of  $\mathcal{J}$  by  $(c+1)^u \cdot |\mathcal{G}(\mathcal{P})|$ , for  $c$  the number of distinct constants in  $\mathcal{P}$  and  $u$  as in Lemma 17, since each rule in  $\mathcal{G}(\mathcal{P})$  yields at most  $(c+1)^u$  rules in  $\mathcal{P}_{\text{OG}}^{\mathcal{J}}$ .

Hence, Lemma 17 yields Algorithm 1 for deciding non-entailment of a fact  $\gamma$  by a numerically stratified LL-program  $\mathcal{P}$ , where  $\mathcal{J}' \sqsubset \mathcal{J}$  holds if  $\mathcal{I} \sqsubset \mathcal{I}'$  for the interpretations  $\mathcal{I}$  and  $\mathcal{I}'$  corresponding to  $\mathcal{J}$  and  $\mathcal{J}'$ , respectively. Given  $\mathcal{P}$  and  $\gamma$ , the algorithm guesses a pseudomodel  $\mathcal{J}$  of  $\mathcal{P}$  witnessing  $\mathcal{P} \not\models \gamma$ , and then checks that  $\mathcal{J}$  is indeed a minimal model of  $\mathcal{P}_{\text{OG}}^{\mathcal{J}}$  and does not satisfy  $\gamma$ . Note that the bounds on the maximal magnitude of numbers given by Lemma 17 are doubly exponential in general and exponential in the size of the dataset, which implies that  $\mathcal{J}$  can be guessed in expo-

nential and polynomial time, respectively. Finally, the universal guess of  $\mathcal{J}'$  in line 3 can be done using an NP-oracle. Thus, analysis of Algorithm 1 yields the following result.

**Theorem 18.** *The fact entailment problem for numerically stratified LL-programs is in  $\Pi_2^{\text{EXP}}$  in combined and in  $\Pi_2^{\text{P}}$  in data complexity.*

## 6 Lower Bounds and Expressivity

We next complement the upper bounds established in Section 5 by matching lower bounds for all the fragments of disjunctive Datalog<sub>ℤ</sub> studied in this paper; furthermore, we establish their expressive power. While several lower bounds follow from existing results, the expressivity and complexity of disjunction-free numerically stratified programs is established via a novel and intricate Turing machine reduction.

Recall that the complexity of disjunction-free stratified Datalog<sub>ℤ</sub> and its fragments follows from (Kaminski et al. 2017; 2018), while for the disjunctive fragments, the upper bounds of Theorems 14 and 18 are the same as the bounds of Eiter, Gottlob, and Mannila [1997] for the same fragments without arithmetic. So, we can derive the following theorem.

**Theorem 19.** *The fact entailment problem for numerically stratified and stratified LL-programs is  $\Pi_2^{\text{EXP}}$ -complete in combined and  $\Pi_2^{\text{P}}$ -complete in data complexity. The problem for semi-positive and positive LL-programs is coNEXP-complete and coNP-complete, respectively. The languages of numerically stratified and stratified LL-programs capture  $\Pi_2^{\text{P}}$ , while semi-positive LL-programs capture coNP.*

Note, however, that the situation is different for numerically stratified LL-programs without disjunction, where fact entailment is coNEXP- and coNP-complete, respectively, in the case without arithmetic (Eiter, Gottlob, and Mannila 1997), and the language captures coNP, but the best upper bounds that we have with arithmetic are  $\Pi_2^{\text{EXP}}$  and  $\Pi_2^{\text{P}}$  from Theorem 18, with the latter bound being tight by Example 8. We next show that for such programs, arithmetic adds not only complexity but also expressivity.

**Theorem 20.** *The language of disjunction-free numerically stratified LL-programs captures  $\Pi_2^{\text{P}}$ .*

The proof idea is similar to the one of Example 8, but the technical details are much more elaborate. In particular, the universal guesses of the main coNP Turing machine are simulated by non-stratified negation over object predicates; then, the guesses of the NP oracle are modelled using arithmetic: each such guess is represented by an integer of

polynomial size, and checking these integers one by one is equivalent to guessing an accepting computation.

It is important to note that, as the proof of this theorem implies, the full expressive power of the language is available already when the programs do not mention  $\infty$ ,  $\times$  or  $-$ , and when the only used integer is 1. As already noted,  $\Pi_2^P$ -hardness of fact entailment in data complexity is shown in Example 8 (and also implied by capturing); moreover, it is possible to adapt the construction in the proof of Theorem 20 to  $\Pi_2^{\text{EXP}}$ -hardness in combined complexity.

**Corollary 21.** *The fact entailment problem for disjunction-free numerically stratified LL-programs is  $\Pi_2^{\text{EXP}}$ -complete in combined and  $\Pi_2^P$ -complete in data complexity.*

## 7 Conclusion and Future Work

We have studied disjunctive limit-linear Datalog $_{\mathbb{Z}}$ , which extends both disjunctive Datalog (Eiter, Gottlob, and Mannila 1997) and stratified limit-linear Datalog $_{\mathbb{Z}}$  (Kaminski et al. 2018). Our complexity results are summarised in Table 1. Data complexity coincides with the expressive power for all fragments with negation. For disjunctive fragments, the addition of arithmetic does not increase complexity or expressive power; without disjunction, however, arithmetic increases both. For future work, we first plan to study possibility semantics of stable models for Datalog $_{\mathbb{Z}}$  with disjunction and negation, as it is done by Eiter, Gottlob, and Mannila [1997] in the setting without arithmetic. Note that Eiter, Gottlob, and Mannila showed that in most cases the complexity of fact entailment under possibility semantics is ‘symmetric’ to the one under certainty semantics: for example, for the full language it is  $\Sigma_2^{\text{EXP}}$ -complete in combined and  $\Sigma_2^P$ -complete in data complexity; however, for some fragments symmetry is not preserved. Second, we plan to consider well-founded semantics of negation (Gelder, Ross, and Schlipf 1991; Przymusiński 1995) in the setting with arithmetic.

## 8 Acknowledgments

Research funded by the EPSRC projects OASIS, ED<sup>3</sup>, and AnaLOG, as well as by the SIRIUS Centre for Scalable Access in the Oil and Gas Domain.

## References

Alvaro, P.; Condie, T.; Conway, N.; Elmeleegy, K.; Hellerstein, J. M.; and Sears, R. 2010. BOOM analytics: exploring data-centric, declarative programming for the cloud. In *EuroSys*, 223–236.

Apt, K. R.; Blair, H. A.; and Walker, A. 1988. Towards a theory of declarative knowledge. In Minker, J., ed., *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann. 89–148.

Chandra, A. K., and Harel, D. 1985. Horn clauses queries and generalizations. *J. Log. Program.* 2(1):1–15.

Chin, B.; von Dincklage, D.; Ercegovic, V.; Hawkins, P.; Miller, M. S.; Och, F. J.; Olston, C.; and Pereira, F. 2015. Yedalog: Exploring knowledge at scale. In *SNAPL*, 63–78.

Chistikov, D., and Haase, C. 2016. The taming of the semi-linear set. In *ICALP*, 128:1–128:13.

Dantsin, E.; Eiter, T.; Gottlob, G.; and Voronkov, A. 2001. Complexity and expressive power of logic programming. *ACM Comput. Surv.* 33(3):374–425.

Eisner, J., and Filardo, N. W. 2011. Dyna: Extending datalog for modern AI. In *Datalog 2.0*, 181–220.

Eiter, T.; Gottlob, G.; and Mannila, H. 1997. Disjunctive datalog. *ACM Trans. on Database Syst.* 22(3):364–418.

Gelder, A. V.; Ross, K. A.; and Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *J. ACM* 38(3):620–650.

Grädel, E. 2007. Finite model theory and descriptive complexity. In *Finite Model Theory and Its Applications*. Springer. 125–230.

Hemachandra, L. A. 1989. The strong exponential hierarchy collapses. *J. Comput. Syst. Sci.* 39(3):299–322.

Immerman, N. 1999. *Descriptive Complexity*. Springer.

Kaminski, M.; Cuenca Grau, B.; Kostylev, E.; Motik, B.; and Horrocks, I. 2017. Foundations of declarative data analysis using limit datalog programs. In *IJCAI*, 1123–1130.

Kaminski, M.; Cuenca Grau, B.; Kostylev, E. V.; Motik, B.; and Horrocks, I. 2018. Stratified negation in limit datalog programs. In *IJCAI*, 1875–1881.

Kolaitis, P. G., and Papadimitriou, C. H. 1991. Why not negation by fixpoint? *J. Comput. Syst. Sci.* 43(1):125–144.

Ladner, R. E., and Lynch, N. A. 1976. Relativization of questions about log space computability. *Math. Systems Theor.* 10:19–32.

Loo, B. T.; Condie, T.; Garofalakis, M. N.; Gay, D. E.; Hellerstein, J. M.; Maniatis, P.; Ramakrishnan, R.; Roscoe, T.; and Stoica, I. 2009. Declarative networking. *Commun. ACM* 52(11):87–95.

Papadimitriou, C. H. 1994. *Computational Complexity*. Addison-Wesley.

Przymusiński, T. C. 1991. Stable semantics for disjunctive programs. *New Generation Comput.* 9(3/4):401–424.

Przymusiński, T. C. 1995. Static semantics for normal and disjunctive logic programs. *Ann. Math. Artif. Intell.* 14(2–4):323–357.

Ross, K. A., and Sagiv, Y. 1997. Monotonic aggregation in deductive databases. *J. Comput. Syst. Sci.* 54(1):79–97.

Schlipf, J. S. 1995. The expressive powers of the logic programming semantics. *J. Comput. Syst. Sci.* 51(1):64–86.

Seo, J.; Guo, S.; and Lam, M. S. 2015. Socialite: An efficient graph query language based on datalog. *IEEE Trans. Knowl. Data Eng.* 27(7):1824–1837.

von zur Gathen, J., and Sieveking, M. 1978. A bound on solutions of linear integer equalities and inequalities. *Proc. AMS* 72(1):155–158.

Wang, J.; Balazinska, M.; and Halperin, D. 2015. Asynchronous and fault-tolerant recursive datalog evaluation in shared-nothing engines. *PVLDB* 8(12):1542–1553.

Yang, M.; Shkapsky, A.; and Zaniolo, C. 2017. Scaling up the performance of more powerful datalog systems on multicore machines. *VLDB J.* 26(2):229–248.