The Complexity and Expressive Power of Limit Datalog

MARK KAMINSKI[*], Department of Computer Science, University of Oxford, UK
EGOR V. KOSTYLEV[*], Department of Computer Science, University of Oxford, UK and Department of Informatics, University of Oslo, Norway
BERNARDO CUENCA GRAU, Department of Computer Science, University of Oxford, UK
BORIS MOTIK, Department of Computer Science, University of Oxford, UK
IAN HORROCKS, Department of Computer Science, University of Oxford, UK

Motivated by applications in declarative data analysis, in this paper we study *Datalog*$_\mathbb{Z}$—an extension of Datalog with stratified negation and arithmetic functions over integers. This language is known to be undecidable, so we present the fragment of *limit Datalog*$_\mathbb{Z}$ programs, which is powerful enough to naturally capture many important data analysis tasks. In limit *Datalog*$_\mathbb{Z}$, all intensional predicates with a numeric argument are *limit* predicates that keep maximal or minimal bounds on numeric values. We show that reasoning in limit *Datalog*$_\mathbb{Z}$ is decidable if a *linearity* condition restricting the use of multiplication is satisfied. In particular, limit-linear *Datalog*$_\mathbb{Z}$ is complete for Δ_2^{EXP} and captures Δ_2^{P} over ordered datasets in the sense of descriptive complexity. We also provide a comprehensive study of several fragments of limit-linear *Datalog*$_\mathbb{Z}$. We show that semi-positive limit-linear programs (i.e., programs where negation is allowed only in front of extensional atoms) capture coNP over ordered datasets; furthermore, reasoning becomes coNEXP-complete in combined and coNP-complete in data complexity, where the lower bounds hold already for negation-free programs. In order to satisfy the requirements of data-intensive applications, we also propose an additional stability requirement, which causes the complexity of reasoning to drop to EXP in combined and to P in data complexity, thus obtaining the same bounds as for usual Datalog. Finally, we compare our formalisms with the languages underpinning existing Datalog-based approaches for data analysis and show that core fragments of these languages can be encoded as limit programs; this allows us to transfer decidability and complexity upper bounds from limit programs to other formalisms. Therefore, our paper provides a unified logical framework for declarative data analysis which can be used as a basis for understanding the impact on expressive power and computational complexity of the key constructs available in existing languages.

CCS Concepts: • **Theory of computation → Constraint and logic programming**; *Complexity theory and logic*; *Database query languages (principles)*; *Logic and databases.*

Additional Key Words and Phrases: Datalog, declarative data analytics

[*]Both authors contributed equally to this research.

Authors' addresses: Mark Kaminski, Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford, OX1 3QD, UK, maka5008@gmail.com; Egor V. Kostylev, Department of Computer Science, University of Oxford, UK and Department of Informatics, University of Oslo, Postboks 1080, Blindern 0316, Oslo, Norway, egork@ifi.uio.no; Bernardo Cuenca Grau, Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford, OX1 3QD, UK, bernardo.cuenca.grau@cs.ox.ac.uk; Boris Motik, Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford, OX1 3QD, UK, boris.motik@cs.ox.ac.uk; Ian Horrocks, Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford, OX1 3QD, UK, ian.horrocks@cs.ox.ac.uk.

0004-5411/2018/0-ART0 $15.00
https://doi.org/0000001.0000001

ACM Reference Format:
Mark Kaminski, Egor V. Kostylev, Bernardo Cuenca Grau, Boris Motik, and Ian Horrocks. 2018. The Complexity and Expressive Power of Limit Datalog. *J. ACM* 00, 0, Article 0 (2018), 82 pages. https://doi.org/0000001.0000001

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, and query answering. Although these tasks are currently implemented using code written in usual imperative programming languages, in recent years there has been a significant shift towards declarative solutions, where the definition of the task is clearly separated from its implementation [2, 38, 53, 54, 61]. In declarative data analysis, users 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 first describes what a path length is and then selects only paths of minimum length. Such specification is independent of evaluation details, allowing analysts to focus on the task at hand rather than implementation details. An evaluation strategy can be chosen later, and efficient general algorithms can be reused 'for free'.

An essential ingredient of declarative data analysis is a suitable logic-based language, such that the relevant analysis tasks can be reduced to the fact entailment problem in the corresponding logic. Datalog [1, 14] is a prime candidate since it supports recursion, which is needed, for instance, to capture shortest path computations. Even basic data analysis tasks, however, also require integer arithmetic or aggregation to capture quantitative aspects of data (e.g., the length of a shortest path). Research on extending Datalog with means for capturing numeric computations dates back to the '90s [5, 11, 20, 35, 42, 48, 59] and is currently experiencing a revival [19, 39, 63]. This extensive body of work, however, focuses primarily on integrating recursion with arithmetic and 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 such languages, other than the fact that extending Datalog just with arithmetic function symbols over the integer numbers trivially leads to undecidability of the fact entailment problem as well as other relevant reasoning tasks [14]. Undecidability also carries over to the formalisms underpinning existing Datalog-based data analysis engines such as BOOM [2], DeALS [62], Myria [61], SociaLite [53], Overlog [37], Dyna [16], and Yedalog [8]. This is a significant issue in practice, since it implies that no general termination guarantees can be provided for these systems.

Our aim in this article is to lay a sound foundation for Datalog-based declarative data analysis by developing new extensions of Datalog that are powerful and flexible enough to naturally capture many important analysis tasks, while at the same time ensure decidability of fact entailment. Such languages can then establish the formal basis for the development of reasoning engines supporting complex analytical tasks, and providing correctness, robustness, scalability and extensibility guarantees. Furthermore, they can serve as a unified logical framework providing a basis for understanding the impact of the key constructs available in existing languages on expressive power and computational complexity.

We take as a starting point *Datalog*$_\mathbb{Z}$—a natural extension of Datalog over a two-sorted signature with stratified negation as failure, integer arithmetic, and comparisons. This is a very expressive language, which is powerful enough to capture all the analysis tasks addressed in existing declarative data analysis proposals. Unfortunately, fact entailment (and hence other relevant reasoning problems) in *Datalog*$_\mathbb{Z}$ are trivially undecidable. In Section 2, we introduce the necessary preliminaries on *Datalog*$_\mathbb{Z}$, complexity theory, and integer programming. Then, with the goal of regaining decidability while retaining the key expressivity of *Datalog*$_\mathbb{Z}$ for capturing data analysis tasks, we present in Section 3 the language of *limit Datalog*$_\mathbb{Z}$, which can be equivalently seen either as a semantic or as a syntactic restriction of $Datalog_{\mathbb{Z}}$. In limit $Datalog_{\mathbb{Z}}$, all intensional predicates with a numeric argument are *limit predicates* that keep maximal or minimal bounds on the numeric values for each tuple of other arguments. For example, if we encode a directed graph with weighted edges using a ternary predicate *edge*, then rules (1) and (2), where *dist* is a min limit predicate (i.e., a predicate keeping only a minimal bound), compute the cost of a shortest path from a source node a_s to each other node in the graph.

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

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

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 a' is reachable from a_s with cost at most $m + n$. If these rules and a dataset entail a fact $dist(a, \ell)$, then the length of a shortest path from a_s to a is at most ℓ; hence, $dist(a, k)$ holds for each $k \geq \ell$ since the cost of a shortest path is also at most k. This is different from ordinary $Datalog_{\mathbb{Z}}$, where there is no implicit semantic connection between $dist(a, \ell)$ and $dist(a, k)$.

We first provide a direct semantics for limit programs based on *limit-closed interpretations*, which are Herbrand interpretations that are closed under entailment of limit facts; for instance, in our shortest path example, a limit-closed interpretation containing a limit fact $dist(a, \ell)$ will also contain all limit facts $dist(a, k)$ for each $k \geq \ell$, thus capturing that existence of a path from a_s to a of cost at most ℓ implies the existence of such a path of cost at most k for all such k. We then show that this model-theoretic semantics can be equivalently axiomatised in ordinary $Datalog_{\mathbb{Z}}$; as a result, our formalism inherits well-understood properties such as monotonicity of positive programs and existence of a least fixpoint model [14]. In Section 3.2, we show that fact entailment remains undecidable for limit $Datalog_{\mathbb{Z}}$ programs, if no restrictions are imposed on the use of integer multiplication in the program. Towards regaining decidability, we introduce *limit-linear* $Datalog_{\mathbb{Z}}$ where the use of multiplication is suitably restricted. In Section 3.3, we argue that limit-linear $Datalog_{\mathbb{Z}}$ can capture many relevant data analysis tasks described in the literature.

In Section 4, we establish decidability of fact entailment for limit-linear programs, as well as design algorithms for positive programs (i.e., negation-free), semi-positive programs (i.e., where negation appears only in front of extensional atoms), and programs with stratified negation. Our algorithms are based on a reduction to the evaluation problem for Presburger sentences of a certain shape. In particular, after introducing an important characterisation of fact entailment by *pseudointerpretations* in Section 4.1, we show in Section 4.2 that our reduction allows us to design a nondeterministic fact entailment algorithm for positive limit-linear programs providing coNEXP and coNP upper bounds in combined and data complexity, respectively. Furthermore, in Section 4.3, we show that fact entailment for semi-positive programs can be reduced in polynomial time to fact entailment over positive programs. As a result, the aforementioned coNEXP and coNP complexity upper bounds extend also to semi-positive programs. Finally, we focus our attention on limit-linear programs with stratified negation and design a fact entailment algorithm that materialises (i.e., computes all facts derived from) the input program stratum-by-stratum, by relying at each stage on the availability of an oracle for computing the materialisation of a semi-positive program corresponding to each stratum. The analysis of our algorithm provides a Δ_2^{EXP} complexity upper bound (thus placing the problem within the second level of the weak exponential hierarchy), and a Δ_2^{P} upper bound in data complexity (thus within the second level of the polynomial hierarchy).

In Section 5, we characterise the *expressive power* of limit-linear programs and establish complexity lower bounds matching the upper bounds in Section 4. Our main result in Section 5.1 is that the language of semi-positive limit-linear programs *captures* (in the sense of descriptive complexity [29]) the complexity class coNP over ordered datasets; in particular, for every problem in coNP over such datasets, there exists a semi-positive limit-linear program that expresses this problem. The expressive power of query languages is intimately related to their data complexity: since semi-positive limit-linear programs can express all coNP problems, we immediately obtain coNP-hardness of fact entailment in data complexity for such programs. Moreover, the proof of our result on expressive power can be easily adapted for showing coNP-hardness in data complexity even for positive limit-linear programs, as well as coNEXP-hardness in combined complexity for such programs. We then focus our attention in Section 5.2 on stratified limit-linear programs and show that they can capture, again over ordered datasets, the complexity class Δ_2^{P}, even when the number of strata is limited to at most three. The proof of this result can be easily adapted for showing Δ_2^{EXP}- and Δ_2^{P}-hardness in combined and data complexity, respectively, for limit-linear programs with at most two strata, thus matching the upper bounds in Section 4.

The results of Section 5 establish intractability of reasoning over limit-linear programs, even in data complexity. In Section 6, we identify fragments of our language for which reasoning becomes tractable in data complexity and which are therefore well-suited for data-intensive applications. In particular, using the idea of *cyclic dependency* detection formalised in Section 6.1, we identify in Section 6.2 a *stability* condition and show that reasoning over stable limit-linear programs becomes EXP-complete in combined complexity and P-complete in data complexity (i.e., no harder than reasoning in usual Datalog). Stability, however, is a semantic condition that is hard to check; thus, in Section 6.3, we identify a syntactic *type-consistency* condition, which implies stability and can be verified in logarithmic space. We then argue that most of the relevant analysis tasks discussed in our examples can be captured by type-consistent programs.

The majority of rule-based languages for data analysis proposed in the literature provide aggregation, such as sum and max, in rules as a key feature. The expressive power provided by aggregation in such languages can be simulated by limit-linear programs using a combination of recursion and arithmetic. Simulating aggregation in this way, however, may lead to programs that are not very intuitive; furthermore, the simulation has an additional disadvantage since it depends on the assumption that datasets are *ordered*. To address these limitations, in Section 7, we extend the language of limit-linear programs with explicit aggregation constructs, which allow us to express certain analytic tasks more naturally and without relying on the order assumption. We then show that aggregation constructs do not increase the computational complexity of reasoning since each limit-linear program with aggregation can be polynomially rewritten into an aggregate-free limit-linear program.

In Section 8, we compare our language with the formalisms that underpin existing rule-based systems for data analysis. In particular, in Section 8.1, we concentrate on the monotonic programs of Ross and Sagiv [48], and show that a large class of these programs can be captured by limit-linear *Datalog*$_{\mathbb{Z}}$, which implies that reasoning for this class is feasible in Δ_2^{EXP} in combined and in Δ_2^{P} in data complexity. To the best of our knowledge, these are the first decidability results for the formalism of Ross and Sagiv. In Section 8.2, we discuss other related knowledge representation and database approaches in a less formal manner and argue that many of them can be captured by our formalism as well. Therefore, we believe that our paper provides a unified logical framework for declarative data analysis that can be used as a basis for understanding the impact on expressive power and computational complexity of the key constructs available in existing languages.

Finally, in Section 9, we outline promising directions for future work.

For readability, the paper also has an appendix containing formal proofs of some statements (e.g., lemmas and theorems) that are relatively straightforward, but require lengthy reasoning; each such proof is explicitly referenced in the main body.

This paper extends the preliminary results reported in two conference publications [31, 32].[1] The most significant additions with respect to our conference papers are the results on the expressive power of limit-linear programs reported in Section 5, the extension with aggregates described in Section 7, and the decidability and complexity results for the related formalism by Ross and Sagiv [48] reported in Section 8.1. In addition, we have tightened all our complexity lower bounds (e.g., by making them applicable to multiplication-free programs), generalised key definitions such as that of limit-linearity, and significantly improved the proofs of all our technical results.

Summary of Main Technical Results. For the convenience of the reader, we conclude this section by providing a brief guide to the main theorems in our paper.

- Theorem 3.6 establishes, using a simple reduction of Hilbert's tenth problem, that the fact entailment problem for positive (i.e., negation-free) limit programs is undecidable due to multiplication of variables. This result motivates our definition of limit-linear programs.
- Theorem 4.17 shows that fact entailment for positive limit-linear programs is decidable in coNEXP in combined complexity and in coNP in data complexity. In turn, Theorem 4.24 establishes that these upper bounds extend seamlessly to semi-positive limit-linear programs. Matching lower bounds are provided by Theorem 5.4 already for positive limit-linear programs using only 1 as numeric constant (which is tantamount to assuming unary coding of numbers).
- Theorem 4.26 shows that fact entailment for the full class of limit-linear programs (where negation is stratified) is in Δ_2^{EXP} in combined complexity and in Δ_2^{P} in data complexity. Matching lower bounds are provided by Theorem 5.10, already for limit-linear programs admitting at most two strata and using only 1 as a numeric constant.
- Theorems 5.3 and 5.9 establish our main results on expressive power. In particular, they respectively show that the language of semi-positive limit-linear programs captures coNP and the language of all limit-linear-programs captures Δ_2^{P}, both over ordered datasets. Furthermore, the full expressive power of the languages is available already for limit-linear programs where 1 is the only numeric constant, and, for capturing Δ_2^{P}, when programs admit at most three strata.
- Theorem 6.12 establishes that fact entailment for the restricted class of stable limit-linear programs is EXP-complete in combined complexity and P-complete in data complexity, which are both the same as for usual Datalog. Theorem 6.18 then shows that a simple syntactic type-consistency property is sufficient for stability.
- Theorem 7.8 shows that extending limit programs with aggregation neither adds expressive power, nor increases the complexity of reasoning.
- Theorem 8.4 establishes a connection between the monotonic programs by Ross and Sagiv and our formalism, and helps us identify a subclass of monotonic programs for which fact entailment is decidable in DEXP in combined complexity and in DP in data complexity.

2 PRELIMINARIES

We assume familiarity with the basic concepts of complexity theory, logic, and rule-based query languages for databases and knowledge representation.

2.1 Datalog$_{\mathbb{Z}}$

We recapitulate the well-known syntax and semantics of Datalog with stratified negation and arithmetic over the integers, which we call *Datalog*$_{\mathbb{Z}}$. Our formalism is standard and closely related to constraint logic programming (CLP) over the structure $(\mathbb{Z}, \leq, <, +, -, \times, 0, \pm 1, \pm 2, \dots)$, for $\mathbb{Z}$ the set of integers, as defined in [13, 14]. We only deviate in a minor way from the standard CLP definitions by allowing the use of the special symbol ∞, which corresponds to a universal quantification over

[1]Paper [31] received the IJCAI-2017 distinguished paper award.

all integers as explained later on; the introduction of this symbol will allow us to capture the formalism of rules with monotonic aggregates by Ross and Sagiv [48], as described in Section 8.1. None of our main results on expressive power (Theorems 5.3 and 5.9) or complexity lower bounds (Theorems 5.4, 5.5, and 5.10) rely on the presence of ∞ in the language.

Syntax. We assume countably infinite and mutually disjoint sets of *objects*, *object variables*, *numeric variables*, and *predicates*. Each predicate has an *arity* from the set $\mathbb{N}$ of natural numbers with zero, and each position of each predicate is of either *object* or *numeric sort*. The set of predicates includes the usual binary *comparison* predicates $\leq$ and $<$ with both positions numeric. We refer to all other predicates as *standard* to distinguish them from the comparison predicates.

An *object term* is an object or an object variable. A *numeric term* is an integer, a numeric variable, the special symbol ∞, 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 ∞, while $+$, $-$ and $\times$ are the usual *arithmetic functions*. A *constant* is an object or an integer (note that the special symbol ∞ is not considered to be a constant).

A *standard atom* is an expression of the form $A(t_1, \ldots, t_v)$, where A is a standard predicate of arity v and each t_i is a term matching the sort of position i of A. A *comparison atom* is of the form $(s_1 \lhd s_2)$, where s_1 and s_2 are numeric terms different from ∞ and $\lhd$ is a comparison predicate (i.e., $\leq$ or $<$). We use standard abbreviations concerning comparison atoms, such as $(s_1 \geq s_2)$ for $(s_2 \leq s_1)$, $(s_1 \doteq s_2)$ for $(s_1 \leq s_2) \wedge (s_2 \leq s_1)$, and $(s_1 \leq s_2 < s_3)$ for $(s_1 \leq s_2) \wedge (s_2 < s_3)$.

A *positive literal* is a standard atom or a comparison atom, and a *negative (standard) literal* is an expression of the form $\mathtt{not}\,\alpha$, for α a standard atom. A *standard literal* is a literal over a standard atom.

A *rule* ρ is a first-order sentence of the form

$$\forall \mathbf{x}.\, \varphi \rightarrow \alpha, \tag{3}$$

where the *body* φ of ρ is a conjunction of (positive and negative) literals, the *head* α of ρ is a standard atom, and $\mathbf{x}$ is the tuple of all variables in φ and α; the quantifier $\forall \mathbf{x}$ is often omitted.

A rule ρ is *safe* if each variable in ρ occurs in the body of ρ in a positive literal. Safety ensures (object) *domain independence*—that is, that the semantics of a set of rules does not depend on objects in the vocabulary not explicitly mentioned in the rules. Note that, under these definitions, rules such as $(n \leq n) \rightarrow C(n)$ for a numeric variable n and a predicate C with a single numeric position is considered safe. On the one hand, such rules are convenient and often used in the paper; on the other hand, forbidding such rules would not restrict any of the languages we study: as we will see, to simulate such rules it suffices to extend a program with the following rules, for a unary predicate *Integer* with its only position numeric, and then use $\mathit{Integer}(n)$ whenever we need to 'cover' a numeric variable n (in limit-linear programs *Integer* should be max or min, see Definition 3.1).

$$\rightarrow \mathit{Integer}(0) \qquad \mathit{Integer}(n) \rightarrow \mathit{Integer}(n+1) \qquad \mathit{Integer}(n) \rightarrow \mathit{Integer}(n-1)$$

A *fact* is a (safe) rule with empty body and where all the terms in the head are constants (i.e., objects and integers, but not variables, terms with arithmetic functions, or ∞). A *dataset* is a finite set of facts. In addition to facts, it is often convenient to consider *pseudofacts*, which are defined the same as facts except that the special symbol ∞ is allowed to occur in the head atom. We do not usually distinguish between pseudofacts and their head atoms, and often omit $\rightarrow$ when writing pseudofacts (including facts).

A *stratification* of a set $\mathcal{P}$ of rules is a function Λ mapping each standard predicate mentioned in $\mathcal{P}$ to a positive natural number such that, for each rule in $\mathcal{P}$ with head predicate A and each standard body literal λ over predicate B in the rule, we have $\Lambda(B) \leq \Lambda(A)$ if λ is positive and $\Lambda(B) < \Lambda(A)$ if λ is negative. Unless explicitly stated (which will be the case only in Section 6), none of the results in this paper depend on a particular stratification of a set of rules; therefore, we commit to the unique stratification $\Lambda_{\mathcal{P}}$ of a stratified set $\mathcal{P}$ of rules that assigns to each standard predicate in $\mathcal{P}$ the least positive natural number possible. The i-th *stratum* of $\mathcal{P}$, written $\mathcal{P}[i]$, is the set of rules in $\mathcal{P}$ with head predicates A such that $\Lambda_{\mathcal{P}}(A) = i$. A set $\mathcal{P}$ of rules *admits h strata* if h is the largest number with $\mathcal{P}[h]$ nonempty. A standard predicate A is *intensional* (*IDB*) in a set $\mathcal{P}$ of rules if it occurs in $\mathcal{P}$ in the head of a rule that is not a pseudofact; otherwise, A is *extensional* (*EDB*) in $\mathcal{P}$; note that $\Lambda_{\mathcal{P}}(A) = 1$ for every EDB predicate A. Set $\mathcal{P}$ of rules is *positive* if it does not use negative literals (in which case $\mathcal{P}$ admits a single stratum), and it is *semi-positive* if the predicate of each negative literal is EDB in $\mathcal{P}$ (in which case it admits two strata).

A (*Datalog*$_{\mathbb{Z}}$) *program* (*with stratified negation as failure*) is a finite stratified set of safe rules.

Semantics. We adopt the standard notion of *substitutions*—that is, sort-compatible partial mappings of variables to constants. For E an expression (such as a term, atom, or rule) and σ a substitution, $E\sigma$ is the expression obtained by replacing with $\sigma(x)$ each occurrence of every variable x in E on which σ is defined. An expression E' is an *instance* of E if $E' = E\sigma$ for some substitution σ. An expression is *ground* if it mentions no variables (note that each pseudofact is ground by definition, but not every ground atom is a pseudofact, because it may mention arithmetic functions). A substitution σ is a *grounding* of an expression E if $E\sigma$ is ground.

A *Herbrand interpretation*, or just *interpretation*, is a (possibly infinite) set of facts (e.g., a dataset is an interpretation). An interpretation $\mathcal{I}$ *satisfies* a ground literal λ, written $\mathcal{I} \models \lambda$, if one of the following holds:

- λ is a standard atom and $\mathcal{I}$ contains each fact obtained from λ by replacing every occurrence of ∞ by an arbitrary integer (which can be different for different occurrences) and evaluating all the numeric terms (under the usual semantics of arithmetic functions over integers);
- λ is a comparison atom that evaluates to true (under the usual semantics of arithmetic functions and comparisons); and
- λ is a negative literal not α and $\mathcal{I} \not\models \alpha$.

Note that satisfaction of a comparison atom does not actually depend on $\mathcal{I}$, and this case is included only for uniformity. The notion of satisfaction extends to conjunctions of ground literals, rules, and sets of safe rules as in first-order logic. An interpretation $\mathcal{I}$ is a *model* of a program $\mathcal{P}$ if $\mathcal{I} \models \mathcal{P}$.

We next define the notion of entailment for *Datalog*$_{\mathbb{Z}}$ programs. Analogously to the case of usual Datalog with stratified negation, it is formulated in terms of the least fixpoint of an *immediate consequence operator* $\mathcal{S}$, which is defined as follows: for $\mathcal{I}$ an interpretation and $\mathcal{P}$ a program, $\mathcal{S}_{\mathcal{P}}(\mathcal{I})$ is the interpretation defined as the set of all facts γ for which there exists a ground instance $\varphi \to \alpha$ of a rule in $\mathcal{P}$ such that $\mathcal{I} \models \varphi$ and γ can be obtained from α by replacing each occurrence of ∞ by an integer and evaluating all numeric terms. By definition, an interpretation $\mathcal{I}$ is a model of a program $\mathcal{P}$ if and only if $\mathcal{S}_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$. Also, for each positive program $\mathcal{P}$ operator $\mathcal{S}_{\mathcal{P}}$ is monotonic with respect to set inclusion—that is, $\mathcal{S}_{\mathcal{P}}(\mathcal{I}_1) \subseteq \mathcal{S}_{\mathcal{P}}(\mathcal{I}_2)$ for every two interpretations $\mathcal{I}_1$ and $\mathcal{I}_2$ such that $\mathcal{I}_1 \subseteq \mathcal{I}_2$; moreover, the same holds when $\mathcal{P}$ is semi-positive, while $\mathcal{I}_1$ and $\mathcal{I}_2$ coincide on the EDB predicates. Hence we can define the semantics of each stratum of a program as the fixpoint of the immediate consequence operator for this stratum provided the result for the previous stratum is added at each step (the existence of this fixpoint is guaranteed by the Knaster-Tarski theorem due to monotonicity). The presence of atoms involving ∞, however, makes the usual definition of the fixpoint for a (semi-)positive program based on induction over natural numbers insufficient, as illustrated by the following example.

Example 2.1. Consider a positive program $\mathcal{P}$ consisting of the following rules over binary predicates A, with one position of object sort and another of numeric sort, and *next*, with both positions of object sort, which use objects a and a', an object variable x, and a numeric variable m.

$$\begin{array}{lll} \rightarrow A(a,0) & A(x,m) \rightarrow A(x,m+1) & A(x,\infty) \wedge \mathit{next}(x,y) \rightarrow A(y,0) \\ \rightarrow \mathit{next}(a,a') & A(x,m) \rightarrow A(x,m-1) & \end{array}$$

Assuming as usual $\mathcal{S}^0_{\mathcal{P}}(\mathcal{I}) = \mathcal{I}$, $\mathcal{S}^{i+1}_{\mathcal{P}}(\mathcal{I}) = \mathcal{S}_{\mathcal{P}}(\mathcal{S}^i_{\mathcal{P}}(\mathcal{I}))$ for each $i \in \mathbb{N}$, and $\mathcal{S}^{\infty}_{\mathcal{P}}(\mathcal{I}) = \bigcup_{i\in\mathbb{N}} \mathcal{S}^i_{\mathcal{P}}(\mathcal{I})$, we then have that $\mathcal{S}^{\infty}_{\mathcal{P}}(\emptyset) = \{A(a,k) \mid k \in \mathbb{Z}\} \cup \{\mathit{next}(a,a')\}$. However,

$$\mathcal{S}^{\infty}_{\mathcal{P}}(\mathcal{S}^{\infty}_{\mathcal{P}}(\emptyset)) = \{A(a,k), A(a',k) \mid k \in \mathbb{Z}\} \cup \{\mathit{next}(a,a')\};$$

so, the fixpoint of $\mathcal{S}_{\mathcal{P}}$ is not reached after countably infinite number of iterations. Note that both predicates in this example have no more than one numeric position, which, as we will see, is the most relevant situation for this paper.

Therefore, we need an extension of the usual fixpoint semantics of entailment to transfinite induction. Formally, the *partial materialisation* $\mathcal{M}^{\kappa}_i$ of a program $\mathcal{P}$, for an ordinal κ and natural number $i \geq 1$, is an interpretation that is defined by induction on κ and i as follows, where ω_1 is the smallest uncountable ordinal and $\mathcal{M}^{\omega_1}_0 = \emptyset$ is assumed for uniformity (we adopt the convention that ordinal 0 is neither successor nor limit):

$$\mathcal{M}^0_i = \mathcal{M}^{\omega_1}_{i-1}, \quad \mathcal{M}^{\kappa}_i = \mathcal{S}_{\mathcal{P}[i]}(\mathcal{M}^{\kappa'}_i) \cup \mathcal{M}^{\omega_1}_{i-1} \text{ if } \kappa = \kappa' + 1, \quad \mathcal{M}^{\kappa}_i = \bigcup_{\kappa' < \kappa} \mathcal{M}^{\kappa'}_i \text{ if } \kappa \text{ is a limit ordinal.}$$

The *materialisation* $\mathcal{M}(\mathcal{P})$ of $\mathcal{P}$ is then the partial materialisation $\mathcal{M}^{\omega_1}_h$, where h is the number of strata $\mathcal{P}$ admits. The choice of ω_1 in these definitions is justified by the fact that, for each i, $\mathcal{M}^{\omega_1}_i$ is a countable set; therefore, there is an ordinal $\kappa < \omega_1$ such that $\mathcal{M}^{\kappa}_i = \mathcal{M}^{\kappa'}_i$ for each $\kappa' \geq \kappa$. It is immediate to check that $\mathcal{M}(\mathcal{P}) \models \mathcal{P}$—that is, the materialisation of a program $\mathcal{P}$ is a model of $\mathcal{P}$.

A program $\mathcal{P}$ *entails* a fact γ, written $\mathcal{P} \models \gamma$, if $\gamma \in \mathcal{M}(\mathcal{P})$.

For positive programs, our definition of entailment coincides with the usual first-order notion—that is, for a positive program $\mathcal{P}$ and a fact γ, we have that $\mathcal{P} \models \gamma$ if and only if $\gamma \in \mathcal{I}$ for every $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{P}$, or, in other words, that $\mathcal{M}(\mathcal{P}) = \bigcap_{\mathcal{I} \models \mathcal{P}} \mathcal{I}$. Furthermore, for programs not involving ∞, our semantics coincides with the conventional semantics in the literature [4, 14]; in this case we could also use usual induction based on natural numbers without affecting the results.

We study *fact entailment*—that is, the problem of checking whether $\mathcal{P} \models \gamma$ for a program $\mathcal{P}$ and a fact γ. It is well known, however, that this problem is undecidable even for positive *Datalog*$_{\mathbb{Z}}$ programs not mentioning ∞ and using $+$ as the only arithmetic function [14]. However, all undecidability proofs that we know of require programs using standard predicates having at least two numeric positions. In the context of this paper it will be useful to refine this result to apply also to programs where no standard predicate has more than one numeric position.

Theorem 2.2. *The fact entailment problem is undecidable for positive programs that do not mention* ∞, $\times$ *and* $-$, *and that use standard predicates with no object positions and at most one numeric position.*

The proof of this theorem is given in the appendix.

In light of this undecidability result, our goal is to identify restrictions on programs such that, on the one hand, the resulting languages are rich enough to capture interesting practical examples and, on the other hand, they have decidable (and even tractable in some cases) fact entailment.

2.2 Complexity and Expressive Power

We will usually assume that all integers in the input to fact entailment are coded in binary, and write $\|E\|$ for the size of the representation of an expression (e.g., term, atom, program) E assuming this coding of numbers. Our theorems on expressive power and complexity lower bounds in

Section 5, however, hold also under unary encoding of integers, so the choice of encoding is immaterial for all main results of this paper. Additionally, it is sometimes convenient to assume that each integer is represented in unit (i.e., constant) space, and we write $[\![E]\!]$ for the size of the representation of E under this assumption; thus, for each expression E, we have $[\![E]\!] \leq ||E||$ and $||E|| \leq [\![E]\!] \cdot \max_{k \text{ integer in } E} ||k||$ (the latter assuming that E mentions at least one integer).

Computational Complexity. We assume standard definitions of the basic time computational complexity classes such as P, NP, coNP, EXP, NEXP, and coNEXP, as well as of the basic space complexity class L. We also recall the complexity class DP, which is the class of all decision problems P for which there exist problems P_1 and P_2 in NP such that an instance of a problem is in P if and only if it is in P_1 and not in P_2 [44, 45], as well as the complexity class DEXP, which is defined in the same way but on the basis of NEXP instead of NP.

We assume familiarity with the Ladner-Lynch oracle Turing machine model [36]. As usual, given a complexity class C, classes P^C and NP^C consist of decision problems solvable in polynomial time by a deterministic and nondeterministic, respectively, oracle Turing machine with an oracle set whose membership problem is in C. In particular, the classes Δ_i^{P}, Σ_i^{P}, and Π_i^{P}, forming the *polynomial hierarchy*, are defined inductively as follows:

$$\Delta_0^{\mathsf{P}} = \Sigma_0^{\mathsf{P}} = \Pi_0^{\mathsf{P}} = \mathsf{P}, \qquad \Delta_{i+1}^{\mathsf{P}} = \mathsf{P}^{\Sigma_i^{\mathsf{P}}}, \qquad \Sigma_{i+1}^{\mathsf{P}} = \mathsf{NP}^{\Sigma_i^{\mathsf{P}}}, \qquad \Pi_{i+1}^{\mathsf{P}} = \mathsf{coNP}^{\Sigma_i^{\mathsf{P}}}.$$

The classes Δ_i^{EXP}, Σ_i^{EXP}, and Π_i^{EXP} of the *weak exponential hierarchy* [27] are defined analogously as follows:

$$\Delta_0^{\mathsf{EXP}} = \Sigma_0^{\mathsf{EXP}} = \Pi_0^{\mathsf{EXP}} = \mathsf{EXP}, \qquad \Delta_{i+1}^{\mathsf{EXP}} = \mathsf{EXP}^{\Sigma_i^{\mathsf{P}}}, \qquad \Sigma_{i+1}^{\mathsf{EXP}} = \mathsf{NEXP}^{\Sigma_i^{\mathsf{P}}}, \qquad \Pi_{i+1}^{\mathsf{EXP}} = \mathsf{coNEXP}^{\Sigma_i^{\mathsf{P}}}.$$

As already mentioned, our main goal is to study decidability and complexity of the fact entailment problem for several fragments of *Datalog*$_{\mathbb{Z}}$. *Combined complexity* assumes that both a program $\mathcal{P}$ and a fact γ form the input of the problem; in contrast, *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 γ form the input while $\mathcal{P}'$ is fixed. In the latter case, for technical reasons we silently assume that all predicates used in $\mathcal{D}$ and γ are mentioned also in $\mathcal{P}'$, so $\max_{\rho \in \mathcal{P}'} [\![\rho]\!]$ is always larger than $\max_{\gamma' \in \mathcal{D}} [\![\gamma']\!]$ and $[\![\gamma]\!]$. Note that we do not require $\mathcal{D}$ to contain only facts over EDB predicates, as it is often done in the literature; however, our data complexity lower bounds in Section 5 hold under this requirement as well.

Descriptive Complexity. This branch of computer science studies the relationship between the expressive power of logical languages and computational complexity [23, 29]. In this setting, a *logical language* is a set of *sentences* each of which evaluates on a dataset to either true or false, where the considered datasets involve no numeric terms—that is, the datasets whose signatures contain no predicates with numeric positions. The languages FO and SO of first-order and second-order Boolean queries, respectively, are canonical examples of logical languages.

As is often done in the studies of descriptive complexity, especially for Datalog-based languages [17, 29], we shall restrict our attention to the family D of ordered datasets, where a dataset $\mathcal{D}$ is *ordered* if

- its signature includes a special sub-signature Σ_{ord} that consists of unary predicates *first* and *last*, and a binary predicate *next*,
- it mentions at least two objects and contains facts

$$\mathit{first}(a_1), \mathit{next}(a_1, a_2), \ldots, \mathit{next}(a_{c-1}, a_c), \mathit{last}(a_c)$$

for some repetition-free enumeration $a_1, \ldots, a_c$ of all objects in $\mathcal{D}$, and
- the aforementioned facts are the only facts over Σ_{ord} in $\mathcal{D}$.

To study the connections between expressivity of logical languages and computational complexity, we need a way to encode each dataset $\mathcal{D} \in D$ into a string $\mathrm{code}(\mathcal{D})$ forming an input to a Turing machine (or any other formalism underlying a complexity class); however, the main results on descriptive complexity are independent of the precise coding details as long as $\mathrm{code}(\mathcal{D})$ satisfies certain reasonable assumptions [23]. A key objective of descriptive complexity is to characterise a logical language L in terms of the subfamilies of D that L can define. In particular, L *captures* a complexity class C (*over ordered datasets*) if the following conditions hold for each signature Σ extending Σ_{ord}:

1. the problem of evaluating a sentence $\varphi \in L$ on a dataset $\mathcal{D} \in D$ over Σ is in C if φ is considered fixed (i.e., if it is not taken as part of the input);
2. for each problem P in C, there is a sentence $\varphi_P \in L$ such that, for every dataset $\mathcal{D} \in D$ over Σ, sentence φ_P evaluates to true on $\mathcal{D}$ if and only if $\mathrm{code}(\mathcal{D}) \in P$.

In other words, L captures C if, for each signature Σ, the sentences of L define precisely all subfamilies of D over Σ that can be recognised in C. For instance, it is well-known that the language SO of second-order Boolean queries captures the polynomial hierarchy (i.e., the union of all of its members), while its restrictions SO$\exists$ and SO$\forall$ that allow for only existential and universal, respectively, quantification over second-order variables capture NP and coNP [29].

Moreover, a logical language L closed under first-order reductions (i.e., first-order-expressible transformations between datasets) satisfies condition 2 above for each signature Σ if and only if L satisfies this condition for a single signature $\Sigma_{\mathrm{graph}} = \{edge\} \cup \Sigma_{\mathrm{ord}}$, for *edge* a binary predicate [14, 29]. Datasets over Σ_{graph} in D represent directed graphs with ordered sets of nodes. Given such a representation $\mathcal{D}$ of a graph with c nodes, we can fix $\mathrm{code}(\mathcal{D})$ as a binary string of length c^2 such that $\mathcal{D}$ contains a fact $edge(a, b)$ for objects a and b if and only if the bit number $i_a \cdot c + i_b$ in $\mathrm{code}(\mathcal{D})$ is 1, where $i_a, i_b \in [0, c-1]$ are the positions of a and b in the enumeration of the nodes induced by *next* in $\mathcal{D}$. Thus, to prove that a logical language captures a complexity class C over ordered datasets it suffices to show that this language is closed under first-order reductions and satisfies condition 2 for such encodings of datasets over Σ_{graph} in D.

We next apply these definitions to $Datalog_{\mathbb{Z}}$. To this end, we consider a distinct nullary predicate *goal* and define a $Datalog_{\mathbb{Z}}$ program $\mathcal{P}$ to be true on a dataset $\mathcal{D} \in D$ over EDB predicates in $\mathcal{P}$ if $\mathcal{P} \cup \mathcal{D} \models goal$. Note that the signature of dataset $\mathcal{D}$ contains only EDB predicates of program $\mathcal{P}$; thus, while $\mathcal{D}$ cannot contain predicates with numeric positions, program $\mathcal{P}$ can—that is, such predicates are treated as 'internal' to $\mathcal{P}$.

The ability of a $Datalog_{\mathbb{Z}}$ fragment to capture a complexity class immediately implies hardness of the corresponding fact entailment problem for this class in data complexity.

Proposition 2.3. *If a fragment L of $Datalog_{\mathbb{Z}}$ captures a complexity class C over ordered datasets, then the fact entailment problem for L is C-hard in data complexity.*

2.3 Presburger Arithmetic, Integer Programs, Semi-Linear Sets

In this section, we summarise several results on linear arithmetic, which we later use to design fact entailment algorithms for our fragments of $Datalog_{\mathbb{Z}}$.

Presburger Arithmetic. This is the language of first-order logic over a special signature, which consists of constants 0 and 1, the binary addition and subtraction functions $+$ and $-$, and the binary comparison predicates $<$ and $\leq$. Formulas in this language are always evaluated over integers $\mathbb{Z}$ with the usual interpretation of the signature [7, 22, 26, 51]. For convenience, we slightly extend the signature of Presburger formulas by allowing the use of all integers (and not just 0 and 1) as constants (coded in binary) as well as multiplication $\times$ with at least one argument variable-free. These extensions of the syntax have no effect on the expressive power of the language or on any result in this paper, because each integer can be axiomatised using 1, + and −, and multiplication by a variable-free numeric term can be rewritten as addition. Finally, we use truth values false and true as aliases for constants 0 and 1, respectively, in Presburger formulas, as well as allow for Boolean (i.e., ranging over $\{\text{false}, \text{true}\}$) variables, which can be axiomatised using numeric variables in a straightforward way.

A *solution* to a Presburger formula ξ is an assignment σ of integers (and truth values) to the free variables of ξ such that $\xi\sigma$ evaluates to true; ξ is *satisfiable* if it has a solution. Solutions to a Presburger formula with v free variables (ordered in some way) can also be seen as vectors in $\mathbb{Z}^v$.

Integer Programming. A numeric term (different from ∞) is *linear* if each multiplication in it has at least one argument that is variable-free (note that usually only multiplication by a constant is allowed; our slight relaxation will be convenient later on and does not change any essential properties of linear terms). An *integer (linear) program* (*IP*) is a conjunction of comparison atoms where all numeric terms are linear. Note that every IP is a quantifier-free Presburger formula. By evaluating variable-free subterms and factoring out variables, each IP ψ with r conjuncts and v variables $\mathbf{n}$ (when seen as a system of inequalities rather than a formula) can be polynomially rewritten into an equivalent *normal form* $M\mathbf{n} \leq \mathbf{k}$, where M is an integer matrix in $\mathbb{Z}^{r\times v}$ and $\mathbf{k}$ is an integer vector in $\mathbb{Z}^r$. Moreover, if the maximal magnitude (i.e., the absolute value) of an integer in ψ is $b \in \mathbb{N}$, then the magnitudes of all integers in the normal form of ψ are bounded by b^u, where u is the maximal number of integers in a comparison atom in ψ (and hence the maximal size of the binary representations of integers increases at most polynomially). Further, every normal IP with r inequalities over v variables can be transformed, by introducing r 'slack' variables, to an equivalent *augmented* form—that is, a system $M'\mathbf{n} = \mathbf{k}'$ of equations, for $M' \in \mathbb{Z}^{r'\times v'}$ and $\mathbf{k}' \in \mathbb{Z}^{r'}$, with $r' = 2r$, $v' = r + v$, and the same maximal magnitude of an integer as in the normal IP. An augmented IP is still an IP, because equalities can be seen as abbreviations of inequalities as usual. A *linear program* (*LP*) is defined as an IP but with constants and solutions ranging over real numbers rather than integers.

As shown by Papadimitriou [43], a system $M'\mathbf{n} = \mathbf{k}'$ of equations (i.e., an augmented IP), for $M' \in \mathbb{Z}^{r'\times v'}$ and $\mathbf{k}' \in \mathbb{Z}^{r'}$, has a solution in natural numbers if and only if it has a solution in natural numbers bounded by $v'(r'b)^{2r'+1}$, where b is the maximal magnitude of an integer in M' and $\mathbf{k}'$. It immediately follows that such a system has a solution in integers if and only if it has a solution in integers of magnitudes bounded by $v'(r'b)^{2r'+1}$. Combining this observation with the properties of IPs, we obtain the following corollary of Papadimitriou's result.

Proposition 2.4. *If a normal IP $M\mathbf{n} \leq \mathbf{k}$ with $M \in \mathbb{Z}^{r\times v}$ and $\mathbf{k} \in \mathbb{Z}^r$ has a solution, then it has a solution over integers whose magnitudes are bounded by $(r + v)(2rb)^{4r+1}$, where b is the maximal magnitude of an integer in M and $\mathbf{k}$.*

Papadimitriou's result implies that checking satisfiability of an augmented IP is NP-complete. By the above observations, we conclude the same for arbitrary IPs.

Corollary 2.5. *The IP satisfiability problem is* NP*-complete.*

Kannan [34] showed that checking satisfiability of a normal IP is fixed-parameter tractable in the number of variables. His algorithm can decide satisfiability of a normal IP ψ with v variables in $O(v^{9v/2} \cdot \|\psi\|)$ arithmetic operations, while the size of each integer produced by the algorithm is in $O(v^{2v} \cdot \|\psi\|)$ [34, Theorem 5.4]. Since IPs can be converted to normal form without increasing the number of variables, this implies the following proposition.

Proposition 2.6. *There exists a polynomial p such that satisfiability of an IP ψ with v variables can be checked in time polynomial in $2^{p(v)} + \|\psi\|$.*

In this proposition, we also silently claim that both the polynomials not only exist, but can be efficiently (in polynomial time) computed; this silent assumption applies to all such statements in this paper.

Integer Optimisation. An *integer optimisation program* (*IOP*) has the form $(\psi, \mathrm{op}\, s)$ for ψ an IP, $\mathrm{op} \in \{\max, \min\}$, and s a linear numeric term (different from ∞); a solution to ψ is *optimal* for $\max s$ or $\min s$ if it maximises or minimises, respectively, the value of s over all solutions to ψ; this value of s is the *optimal value* of the IOP. An IOP $(\psi, \mathrm{op}\, s)$ is *bounded* if it has an optimal solution; the IOP is *satisfiable*, *normal*, or *augmented* if so is ψ. A *linear optimisation program* (*LOP*) is defined analogously to an IOP but with constants and solutions ranging over the real numbers.

As shown by Meyer [40, Corollary 5.2], an IOP $(\psi, \mathrm{op}\, s)$ with ψ a satisfiable augmented IP is bounded if and only if so is its *linear relaxation*, which is the LOP obtained from $(\psi, \mathrm{op}\, s)$ by omitting the requirement that solutions range over integers. Checking boundedness of a LOP can be reduced to checking satisfiability of a LP (via the duality theorem; see, e.g., [52]), which, in turn, is known to be tractable. Thus, we obtain the following proposition.

Proposition 2.7. *The boundedness problem for satisfiable IOPs is in* P.

As a corollary of his main result, Papadimitriou [43] obtained a pseudopolynomial algorithm for finding the optimal value of a satisfiable and bounded augmented IOP $(\psi, \mathrm{op} \sum_{i=1}^{v} k_i n_i)$ with r equalities over variables $n_1, \ldots, n_v$ in ψ, provided that r is fixed. The algorithm determines the optimal value k for which the IP $\psi \cup \{\sum_{i=1}^{v} k_i n_i = k\}$ is satisfiable by trying out all possible values for k in the interval $[-\ell, \ell]$, for $\ell = v^2(rb^2)^{2r+3} \cdot \sum_{i=1}^{v} |k_i| + 1$, where b is the maximal magnitude of an integer in the IOP. Moreover, by exploiting binary search, the number of such satisfiability checks can be reduced to logarithmically many in the length of the interval—that is, to polynomially many in the size of the IOP—even if r is part of the input. Since every linear term can be normalised to the form $\sum_{i=1}^{v} k_i n_i$ in polynomial time by evaluating variable-free subterms and factoring out variables, the result transfers to our setting as follows (using Proposition 2.6 for the deterministic bound).

Proposition 2.8. *There exists a polynomial p such that the optimal value of a satisfiable and bounded IOP $(\psi, \mathrm{op}\, s)$ with v variables can be computed in polynomial time using satisfiability checks, each of which can be done in* NP *and in time polynomial in $2^{p(v)} + \|(\psi, \mathrm{op}\, s)\|$.*

Semi-Linear Sets. Given a vector $\boldsymbol{\ell} \in \mathbb{Z}^v$ and a finite set $L = \{\boldsymbol{m}_1, \ldots, \boldsymbol{m}_t\} \subseteq \mathbb{Z}^v$ of such vectors, let

$$S(\boldsymbol{\ell}, L) = \left\{ \boldsymbol{\ell} + \sum_{i=1}^{t} d_i \boldsymbol{m}_i \;\middle|\; d_1, \ldots, d_t \in \mathbb{N} \right\}.$$

A set of vectors from $\mathbb{Z}^v$ is *linear* if it is equal to $S(\boldsymbol{\ell}, L)$, for some $\boldsymbol{\ell}$ and L as above. A set of vectors is *semi-linear* if it is a finite union of linear sets. It is known that the set of all solutions to a Presburger formula is semi-linear. Moreover, Chistikov and Haase [9] recently observed, based on results of von zur Gathen and Sieveking [60], that the set of solutions to an IP can be bounded as follows.

Proposition 2.9 ([9, Proposition 3]). *The set of all solutions to a satisfiable normal IP $M\mathbf{n} \leq \mathbf{k}$ with $v = |\mathbf{n}|$ variables is a union of 2^v linear sets $S(\boldsymbol{\ell}_i, L_i)$, $i \in [1, 2^v]$, of vectors in $\mathbb{Z}^v$ with the magnitudes of integers in all $\boldsymbol{\ell}_i$ and L_i bounded by $2^{O(v \log v)} \cdot b^{v-1} \cdot \max(b, b')$, where b and b' are the maximal magnitudes of integers in M and $\mathbf{k}$, respectively.*

This proposition implies that the maximal value of each variable m in $\mathbf{n}$ over all solutions to the IP $M\mathbf{n} \leq \mathbf{k}$ is either equal to the value of m in some $\boldsymbol{\ell}_i$ or does not exist (i.e., is unbounded), and the same holds for the minimal value. Hence optimal values of IOPs can be bounded as follows.

Corollary 2.10. *There exists a polynomial p such that, for each satisfiable and bounded normal IOP $(M\mathbf{n} \le \mathbf{k}, \operatorname{op} m)$ with m a variable in $\mathbf{n}$ such that the maximal magnitudes of integers in M and $\mathbf{k}$ are bounded by $b > 1$ and $b' > 0$, respectively, the magnitude of the optimal value of the IOP is bounded by $b^{p(v)} \cdot b'$, where $v = |\mathbf{n}|$.*

Semi-linear sets are closed under projection, union, intersection, and complement. Chistikov and Haase [9] established the following bound on the complement of a semi-linear set.

Theorem 2.11 ([9, Theorem 21]). *If R is the union of h linear sets $S(\boldsymbol{\ell}_i, L_i) \subseteq \mathbb{Z}^v$, for $i \in [1, h]$, then the complement of R is the union of linear sets $S(\boldsymbol{\ell}'_j, L'_j) \subseteq \mathbb{Z}^v$ with magnitudes of integers in all $\boldsymbol{\ell}'_j$ and L'_j bounded by $(\max(b, 2))^{h \cdot O(v^4)}$, for b the maximal magnitude of an integer in all $\boldsymbol{\ell}_i$ and L_i.*

3 LIMIT AND LIMIT-LINEAR PROGRAMS

In this section, we propose the language of *limit Datalog$_{\mathbb{Z}}$* programs, which can be seen either as a semantic or as a syntactic restriction of *Datalog$_{\mathbb{Z}}$*. Then, we further restrict the use of multiplication to avoid trivial undecidability of fact entailment, arriving to the main object of study in this paper—*limit-linear* programs. Finally, we provide application examples of limit-linear programs.

3.1 Syntax and Semantics of Limit Programs

The key feature of limit programs is that their IDB predicates are partitioned into *object* and *limit* predicates: object predicates are characterised by the fact that they only have object positions, while limit predicates have a distinguished numeric position that keeps a limit (i.e., a maximal or minimal bound) of integers. For instance, consider again rules (1) and (2) given in the introduction, where *dist* is a min limit predicate. The semantics of limit *Datalog$_{\mathbb{Z}}$* is defined in such a way that a fact $dist(a, k)$ is entailed from these rules and a dataset if and only if the distance from the source node a_s to a is at most k; as a result, all facts $dist(a, \ell)$ with $\ell \ge k$ are also entailed. This is in contrast to ordinary numeric predicates, where there is no semantic relationship between $dist(a, k)$ and such $dist(a, \ell)$. The intended semantics of limit predicates can either be defined model-theoretically or axiomatised using rules over ordinary predicates; in particular, our example limit program is equivalent to an ordinary *Datalog$_{\mathbb{Z}}$* program consisting of rules (1), (2), and the following rule (4), where *dist* is now treated as an ordinary predicate.

$$dist(x, m) \wedge (m \le n) \rightarrow dist(x, n) \tag{4}$$

In the rest of this section, we make this intuition precise by formally specifying the syntax and semantics of limit *Datalog$_{\mathbb{Z}}$* programs and establishing their key semantic properties.

Definition 3.1. A *limit* (*Datalog$_{\mathbb{Z}}$*) program is a pair $(\mathcal{P}, \tau)$ such that

- $\mathcal{P}$ is a *Datalog$_{\mathbb{Z}}$* program where each standard predicate is either an *object predicate* with only object positions, or a *numeric predicate* with its last position numeric and all other positions object; and
- τ 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 ∞ in $\mathcal{P}$.

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

All syntactic notions defined on ordinary programs (EDB or IDB predicate, stratification, and so on) extend to a limit program $(\mathcal{P}, \tau)$ by applying them to $\mathcal{P}$. For C a limit predicate, we write $\preceq^{\tau}_{C}$ for $\le$ if $\tau(C) = \max$, and for $\ge$ if $\tau(C) = \min$, with $\prec^{\tau}_{C}$, $\succeq^{\tau}_{C}$, and $\succ^{\tau}_{C}$ defined accordingly; similarly, for S a set of integers, we write $\max^{\tau}_{C} S$ for $\max S$ if $\tau(C) = \max$, and for $\min S$ if $\tau(C) = \min$. All notions introduced in Definition 3.1 for predicates directly propagate to atoms, literals, and facts.

Example 3.2. Consider the limit program $(\mathcal{P}, \tau)$, where $\mathcal{P}$ consists of rules (1) and (2) and τ labels the numeric IDB predicate *dist* as min. The EDB predicate *edge* is an exact predicate, and the program contains no object predicates. The program is homogeneous, as it contains only min limit predicates, and positive, as it is negation-free.

We conclude the definition of the syntax by introducing an abbreviation that captures a recurrent use of stratified negation in examples and will play an important role in the rest of the paper. For C a limit predicate, $\mathbf{t}$ a tuple of object terms of appropriate size, and s a numeric term different from ∞, a *least upper bound* (*LUB*) expression $\lceil C(\mathbf{t}, s)\rceil$ is an abbreviation for the conjunction of literals $C(\mathbf{t}, s) \land \mathsf{not}\, C(\mathbf{t}, s + k)$, where $k = 1$ if C is max and $k = -1$ if C is min.

We next define the direct model-theoretic semantics of limit *Datalog*$_{\mathbb{Z}}$. In this semantics, the intended meaning of limit predicates is captured by requiring all the models $\mathcal{I}$ of a program to be closed for limit predicates—that is, whenever $\mathcal{I}$ contains a limit fact γ, it also contains all facts implied by γ according to the predicate type.

Definition 3.3. An interpretation $\mathcal{I}$ is *limit-closed* for a limit program $(\mathcal{P}, \tau)$ if $C(\mathbf{a}, k') \in \mathcal{I}$ for each limit fact $C(\mathbf{a}, k) \in \mathcal{I}$ and each integer k' such that $k' \leq^{\tau}_{C} k$.

Note that a ground LUB expression $\lceil C(\mathbf{a}, s)\rceil$ with max or min predicate C is satisfied by a limit-closed interpretation $\mathcal{I}$ if and only if s evaluates to the greatest or least, respectively, integer k such that $C(\mathbf{a}, k) \in \mathcal{I}$.

The semantics of fact entailment is then defined in terms of an immediate consequence operator acting on limit-closed interpretations, as specified next.

Definition 3.4. Given a limit program $(\mathcal{P}, \tau)$ and an interpretation $\mathcal{I}$ that is limit-closed for $(\mathcal{P}, \tau)$, let $\mathcal{S}_{(\mathcal{P},\tau)}(\mathcal{I})$ be the least limit-closed superset of $\mathcal{S}_{\mathcal{P}}(\mathcal{I})$ (where $\mathcal{S}_{\mathcal{P}}$ is the ordinary immediate consequence operator, see Section 2.1).

Same as the immediate consequence operator $\mathcal{S}_{\mathcal{P}}$ for ordinary *Datalog*$_{\mathbb{Z}}$ programs, operator $\mathcal{S}_{(\mathcal{P},\tau)}$ for limit programs is then used to define (partial) materialisations of limit programs. The following proposition establishes that $\mathcal{S}_{(\mathcal{P},\tau)}$ inherits the main properties of $\mathcal{S}_{\mathcal{P}}$.

The following holds:

1. $\mathcal{I} \models \mathcal{P}$ if and only if $\mathcal{S}_{(\mathcal{P},\tau)}(\mathcal{I}) \subseteq \mathcal{I}$, for each limit program $(\mathcal{P}, \tau)$ and each interpretation $\mathcal{I}$ that is limit-closed for $(\mathcal{P}, \tau)$; and
2. $\mathcal{S}_{(\mathcal{P},\tau)}(\mathcal{I}_1) \subseteq \mathcal{S}_{(\mathcal{P},\tau)}(\mathcal{I}_2)$ for each semi-positive limit program $(\mathcal{P}, \tau)$ and interpretations $\mathcal{I}_1$ and $\mathcal{I}_2$ that are limit-closed for $(\mathcal{P}, \tau)$, satisfy $\mathcal{I}_1 \subseteq \mathcal{I}_2$, and coincide on the EDB predicates.

The proof of this proposition is given in the appendix.

Claim 2 of Proposition 3.1 and the Knaster-Tarski theorem guarantee the existence of the least fixpoint of the immediate consequence operator $\mathcal{S}_{(\mathcal{P},\tau)}$ for each stratum of a limit program [12], and hence we can define materialisations and fact entailment for limit programs in the same way as for ordinary programs.

Definition 3.5. The *partial materialisations* and the *materialisation* $\mathcal{M}(\mathcal{P}, \tau)$ of a limit program $(\mathcal{P}, \tau)$ are defined as the partial materialisations and materialisation of an ordinary program (see Section 2.1) but using $\mathcal{S}_{(\mathcal{P},\tau)}$ instead of $\mathcal{S}_{\mathcal{P}}$. A limit program $(\mathcal{P}, \tau)$ *entails* a fact γ, written $(\mathcal{P}, \tau) \models \gamma$, if and only if $\gamma \in \mathcal{M}(\mathcal{P}, \tau)$.

Having direct semantics of limit programs at hand, we next argue that they can be easily rewritten to ordinary *Datalog*$_{\mathbb{Z}}$ programs. As illustrated by rule (4) in the beginning of this section, the semantics of limit predicates in a limit program $(\mathcal{P}, \tau)$ can be equivalently axiomatised by means of the ordinary program $\mathcal{P} \cup \mathcal{A}_\tau$, where $\mathcal{A}_\tau$ contains the following rule for each predicate C in the domain of τ and for $\mathbf{x}$ a tuple of distinct object variables of appropriate size.

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

Then, for every semi-positive limit program $(\mathcal{P}, \tau)$ and every interpretation $\mathcal{I}$ limit-closed for $(\mathcal{P}, \tau)$, operator $\mathcal{S}_{(\mathcal{P},\tau)}$ is such that $\mathcal{S}_{(\mathcal{P},\tau)}(\mathcal{I}) = \mathcal{S}_\mathcal{P}(\mathcal{I}) \cup \mathcal{S}_{\mathcal{A}_\tau}(\mathcal{S}_\mathcal{P}(\mathcal{I}))$. Thus, $\mathcal{S}_{(\mathcal{P},\tau)}$ merely imposes a restriction on the order in which the rules are applied when computing the materialisation of $\mathcal{P} \cup \mathcal{A}_\tau$. Since we are interested in a transfinite number of applications, this order is immaterial; in particular, it is easily seen that $\mathcal{M}(\mathcal{P}, \tau) = \mathcal{M}(\mathcal{P} \cup \mathcal{A}_\tau)$ for each limit program $(\mathcal{P}, \tau)$, and hence we can see a limit program as an ordinary program where rules $\mathcal{A}_\tau$ are implicitly assumed.

To conclude this section, we next argue that each limit program can be equivalently rewritten as a homogeneous program without increasing the size of the program. This is achieved by replacing all min (or all max) predicates in the original program with fresh max (min, respectively) predicates and negating their numeric arguments. For the sake of generality, however, in our technical results we will not require limit programs to be homogeneous.

For each limit program $(\mathcal{P}, \tau)$ and each fact γ, we can compute in linear time a homogeneous program $(\mathcal{P}', \tau')$ and a fact γ' such that $(\mathcal{P}, \tau) \models \gamma$ if and only if $(\mathcal{P}', \tau') \models \gamma'$.

In this section, we have established a formal connection between the direct model-theoretic semantics of limit programs and their semantics via axiomatisation. To avoid notational clutter, in the rest of the paper we will abuse notation and write a limit program $(\mathcal{P}, \tau)$ as just $\mathcal{P}$, assuming that τ is given implicitly. Moreover, we will write $\preceq_C$ instead of $\preceq_C^\tau$, $\max_C S$ instead of $\max_C^\tau S$, $\mathcal{S}_\mathcal{P}$ instead of $\mathcal{S}_{(\mathcal{P},\tau)}$, and so on; this should not lead to any confusion since all programs mentioned from now onwards will be limit programs. Also, whenever we consider a union of several limit programs (or rules, considered as singleton programs), we silently assume that their respective τ coincide on shared predicates.

3.2 Undecidability and Limit-Linear Programs

We now start our investigation of the computational properties of limit *Datalog*$_\mathbb{Z}$. However, our first result is negative: without further restrictions, fact entailment remains undecidable.

Theorem 3.6. *The fact entailment problem for positive limit programs is undecidable.*

Proof. The proof is by reduction of Hilbert's tenth problem, which is to determine, given a polynomial $p(m_1, \dots, m_v)$ over variables $m_1, \dots, m_v$ with integer coefficients, whether the equation $p(m_1, \dots, m_v) = 0$ has an integer solution. For each such polynomial p, let $\mathcal{P}_p$ be the positive limit program consisting of single rule (5) with A a nullary object predicate.

$$(p(m_1, \dots, m_v) \doteq 0) \rightarrow A \tag{5}$$

By construction, $\mathcal{P}_p \models A$ if and only if $p(m_1, \dots, m_v) = 0$ has an integer solution. □

The main reason for undecidability in Theorem 3.6 is that, due to multiplication, checking whether the body of a rule matches an interpretation (i.e., checking rule applicability) amounts to finding integer roots of arbitrary polynomials. However, we will see that when all numeric terms are linear (i.e., if we prohibit multiplying variables), checking rule applicability boils down to IP satisfiability, which can be done, by Corollary 2.5, in NP and, by Proposition 2.6, in P if the number of variables is fixed. Thus, to ensure decidability, we next restrict limit programs so that each of their strata can be equivalently restated as a positive program with linear numeric terms.

Definition 3.7. A numeric variable m is *stratification-guarded*, or simply *guarded*,[2] 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 $\lceil C(\mathbf{t}, m) \rceil$. A *limit-linear rule* (*LL-rule*) is a rule where at most one argument of each multiplication mentions unguarded variables. A limit program is *limit-linear* (*LL-program*) if so is each of its rules.

The definition of guardedness can be expanded as follows: a numeric variable m is guarded in a rule ρ if the body of ρ contains either a literal of the form $B(\mathbf{t}, m)$ with B an exact predicate or a conjunction of the form $C(\mathbf{t}, m) \wedge \mathtt{not}\, C(\mathbf{t}, m + k)$ with C a limit predicate and $k = 1$ if C is max and $k = -1$ if C is min. Note, however, that atoms with any other numeric terms involving m, for instance $B(\mathbf{t}, m + 1)$, do not satisfy the requirements of the definition. Intuitively, guarded variables may only be assigned a finite number of possible values provided that we have computed the partial materialisation of all lower strata. Thus, each LL-rule has only finitely many nontautologous instances that can be obtained by grounding its guarded variables with values from a partial materialisation; moreover, all numeric terms other than ∞ in each such instance are linear (recall that, under our definitions, a linear numeric term is linear if at most one argument of each multiplication mentions a variable, which is more general that usual). As we will see, this property allows us to represent the result of applying each such rule instance to a partial materialisation as the optimal solution to an IOP—that is, LL-programs lack the property causing undecidability in Theorem 3.6.

In fact, each LL-program can be normalised to an LL-program in which all positive exact literals in rule bodies are function-free: we just need to replace each such literal $B(\mathbf{a}, s)$ where s is complex (i.e., uses functions) by the conjunction $B(\mathbf{a}, m) \wedge (m \doteq s)$, for m a fresh numeric variable. As a result, all the variables in such literals become guarded. Moreover, such normalisation can be done in linear time and independently for each rule. Hence, for convenience and without loss of generality, we can assume in the rest of the paper that all LL-programs are normalised in this way. (Note also that all exact atoms in rule heads are function-free by definition, because the predicates in all these atoms are EDB, and hence all such rules are facts; however, negative exact literals in rule bodies may still have complex numeric terms after normalisation.) Of course, this normalisation could be extended to limit literals and negative literals; doing so, however, would not be helpful and may even be harmful; for instance, the type-consistency condition, which is introduced in Section 6.3 and guarantees tractability of fact entailment, is not preserved under normalisation of limit literals.

LL-programs are our main object of study in this paper. In what follows, we will investigate their computational properties and expressive power.

3.3 Application Examples

The running example in the previous sections illustrates a simple declarative formulation of the single-source shortest paths problem using limit $Datalog_{\mathbb{Z}}$. In this section, we show a wider range of practical analytics tasks that can be naturally formulated using limit-linear $Datalog_{\mathbb{Z}}$ programs. In several examples we use the ordering sub-signature Σ_{ord} (introduced in Section 2.2); in these cases, we assume that input datasets are ordered.

Example 3.8 (All-pairs shortest paths). Assume that a directed graph with weighted edges is represented as a dataset $\mathcal{D}_{\mathrm{apsp}}$ over a ternary exact predicate *edge* and a unary object predicate *node* in the obvious way. The following LL-program $\mathcal{P}_{\mathrm{apsp}}$ encodes the all-pairs shortest paths problem, where the ternary min predicate *dist* encodes the distance from any node to any node as

[2]This notion is unrelated to the guarded fragment of first-order logic.

the length of a shortest path between them.

$$node(x) \to dist(x, x, 0) \tag{6}$$

$$dist(x, y, m) \wedge edge(y, z, n) \to dist(x, z, m + n) \tag{7}$$

Then, $\mathcal{P}_{\text{apsp}} \cup \mathcal{D}_{\text{apsp}} \models dist(a, a', k)$ if and only if the distance from node a to node a' is at most k.

Example 3.9 (Diffusion in social networks). Consider a social network where agents are connected by the 'follows' relation. A first agent a_s introduces ('tweets') a message, and each agent a 'retweets' the message if at least k_a agents that a follows tweet this message, where k_a is a positive threshold associated with a. 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 $\mathcal{D}_{\text{tw}}$, which contains object fact $tweet(a_s)$ saying that a_s introduces a message, object facts $follows(a, a')$ representing that a follows a', and exact facts $threshold(a, k_a)$ representing that the threshold of a is k_a. The LL-program $\mathcal{P}_{\text{tw}}$ consisting of rules (8)–(12) encodes message propagation. Here, acc is an 'accumulating' max predicate such that $acc(a, a', m)$ is true if there are at least m agents tweeting the message among the agents that a follows and that (inclusively) precede a' according to the dataset order. In particular, rules (8) and (9) initialise acc for the first agent in the order; note that acc is a max predicate, so if the first agent tweets the message, rule (9) 'overrides' rule (8). Rules (10) and (11) recurse over the order to compute acc as stated above.

$$follows(x, y') \wedge first(y) \to acc(x, y, 0) \tag{8}$$

$$tweet(y) \wedge follows(x, y) \wedge first(y) \to acc(x, y, 1) \tag{9}$$

$$acc(x, y', m) \wedge next(y', y) \to acc(x, y, m) \tag{10}$$

$$tweet(y) \wedge follows(x, y) \wedge acc(x, y', m) \wedge next(y', y) \to acc(x, y, m + 1) \tag{11}$$

$$threshold(x, m) \wedge acc(x, y, m) \to tweet(x) \tag{12}$$

Then, $\mathcal{P}_{\text{tw}} \cup \mathcal{D}_{\text{tw}} \models tweet(a)$ if and only if agent a tweets the message according to $\mathcal{D}_{\text{tw}}$.

Example 3.10 (Counting paths). Limit-linear $Datalog_{\mathbb{Z}}$ can also solve the problem of counting paths between pairs of nodes in a directed acyclic graph. We encode such a graph in the obvious way as a dataset $\mathcal{D}_{\text{cp}}$ that uses a unary object predicate $node$ and a binary object predicate $edge$. The LL-program $\mathcal{P}_{\text{cp}}$, consisting of rules (13)–(18) with max predicates $path\text{-}num$ and acc, then counts the paths. Intuitively, $acc(a, a', b, k)$ is true if the sum of the numbers of paths from each node b' preceding node b (according to the dataset order) to node a' for which there exists an edge from node a to b' is at least k. Rule (13) says that each node has one path to itself. Rule (14) initialises aggregation by saying that, for the first (in the order) node z, there are zero paths from any x to any y, and rule (15) overrides this if there exists an edge from x to z. Finally, rule (16) propagates the sum for x to the next z in the order, and rule (17) overrides this if there is an edge from x to z by adding the number of paths from z to y to the sum.

$$node(x) \to path\text{-}num(x, x, 1) \tag{13}$$

$$node(x) \wedge node(y) \wedge first(z) \to acc(x, y, z, 0) \tag{14}$$

$$edge(x, z) \wedge path\text{-}num(z, y, n) \wedge first(z) \to acc(x, y, z, n) \tag{15}$$

$$acc(x, y, z', m) \wedge next(z', z) \to acc(x, y, z, m) \tag{16}$$

$$edge(x, z) \wedge path\text{-}num(z, y, n) \wedge acc(x, y, z', m) \wedge next(z', z) \to acc(x, y, z, m + n) \tag{17}$$

$$acc(x, y, z, m) \to path\text{-}num(x, y, m) \tag{18}$$

Then, $\mathcal{P}_{\text{cp}} \cup \mathcal{D}_{\text{cp}} \models path\text{-}num(a, a', k)$ if and only if there are at least k paths from node a to node a'.

Example 3.11 (Counting paths under bandwidth constraints). Assume that in the graph from Example 3.10 each node a is associated with a 'bandwidth'—that is, a positive integer k_a limiting the number of distinct paths going out of a to at most k_a. To count the paths compliant with the bandwidth requirements, we extend $\mathcal{D}_{\mathrm{cp}}$ to the dataset $\mathcal{D}_{\mathrm{bcp}}$ that additionally contains an exact fact $\mathit{bandwidth}(a, k_a)$ for each node a, and define the LL-program $\mathcal{P}_{\mathrm{bcp}}$ by replacing rules (15) and (17) in $\mathcal{P}_{\mathrm{cp}}$ with the following rules.

$$\begin{aligned}\mathit{edge}(x,z) \wedge \mathit{path\text{-}num}(z,y,n) \wedge \mathit{first}(z) \wedge \\ \mathit{bandwidth}(z,m') \wedge (n \leq m') \rightarrow \mathit{acc}(x,y,z,n)\end{aligned}$$

$$\begin{aligned}\mathit{edge}(x,z) \wedge \mathit{path\text{-}num}(z,y,n) \wedge \mathit{acc}(x,y,z',m) \wedge \mathit{next}(z',z) \wedge \\ \mathit{bandwidth}(z,m') \wedge (n \leq m') \rightarrow \mathit{acc}(x,y,z,m+n)\end{aligned}$$

Then, $\mathcal{P}_{\mathrm{bcp}} \cup \mathcal{D}_{\mathrm{bcp}} \models \mathit{path\text{-}num}(a,a',k)$ if and only if there exist at least k paths from node a to node a', where the bandwidth requirement is satisfied for all nodes on each such path.

Example 3.12 (Bill of materials). Let a dataset $\mathcal{D}_{\mathrm{bm}}$ contain, for each direct or indirect part a needed to manufacture an end product, an object fact $\mathit{part}(a)$, and, for each direct subpart a' of each part a, an exact fact $\mathit{dirpart}(a,a',k)$ indicating that a uses k copies of a' as direct subparts. Clearly, the graph formed by the *dirpart* relation is acyclic and has positive edge weights. The program $\mathcal{P}_{\mathrm{bm}}$, consisting of rules (19)–(24) with max predicates *acc* and *subpart*, then computes how many copies of each subpart are used in total for each part. Intuitively, $\mathit{acc}(a,a',b,k)$ is true if the part a contains at least k copies of the subpart a' in all of the direct subparts of a that precede part b according to the dataset order. Rules (20) and (21) initialise aggregation for b the first part in the order. Rule (22) propagates the value to the next part in case b is not a direct subpart of a; if b is a direct subpart occurring $n_1 \geq 1$ times in a while a' occurs $n_2 \geq 0$ times in b, then rule (23) increments k by $n_1 \times n_2$ to account for all occurrences of a' in a in copies of b. Finally, rule (19) asserts that each part is a subpart of itself, while rule (24) defines k in $\mathit{subpart}(a,a',k)$ for a part a and a subpart a' as the maximum k in $\mathit{acc}(a,a',b,k)$ over all b.

$$\mathit{part}(x) \rightarrow \mathit{subpart}(x,x,1) \tag{19}$$

$$\mathit{part}(x) \wedge \mathit{part}(y) \wedge \mathit{first}(z) \rightarrow \mathit{acc}(x,y,z,0) \tag{20}$$

$$\mathit{dirpart}(x,z,n_1) \wedge \mathit{subpart}(z,y,n_2) \wedge \mathit{first}(z) \rightarrow \mathit{acc}(x,y,z,n_1 \times n_2) \tag{21}$$

$$\mathit{acc}(x,y,z',m) \wedge \mathit{next}(z',z) \rightarrow \mathit{acc}(x,y,z,m) \tag{22}$$

$$\mathit{dirpart}(x,z,n_1) \wedge \mathit{subpart}(z,y,n_2) \wedge \mathit{acc}(x,y,z',m) \wedge \mathit{next}(z',z) \rightarrow \mathit{acc}(x,y,z,m + n_1 \times n_2) \tag{23}$$

$$\mathit{acc}(x,y,z,m) \rightarrow \mathit{subpart}(x,y,m) \tag{24}$$

We have $\mathcal{P}_{\mathrm{bm}} \cup \mathcal{D}_{\mathrm{bm}} \models \mathit{subpart}(a,a',k)$ if and only if a contains at least k copies of a' in total. Note also that $\mathcal{P}_{\mathrm{bm}}$ is limit-linear because variable n_1 occurs in positive exact literals over *dirpart* and is hence guarded in rules (21) and (23).

All examples provided thus far are captured using positive limit-linear $\mathit{Datalog}_{\mathbb{Z}}$ programs. In the following two examples, we demonstrate the use of stratified negation.

Example 3.13 (Computing shortest paths). In the introduction, we discussed a program computing the length of a shortest path from a given source node to every node in a directed graph with weighted edges. We now extend this program to compute the paths themselves (concentrating on a distinguished target node as well). Given a dataset $\mathcal{D}_{\mathrm{csp}}$ encoding a directed graph with positive edge weights using a ternary exact predicate *edge* as before and encoding a source node a_s and a target node a_t using object facts $\mathit{source}(a_s)$ and $\mathit{target}(a_t)$, respectively, our LL-program $\mathcal{P}_{\mathrm{csp}}$ will compute a directed acyclic graph G with source a_s and target a_t, encoded using a binary object predicate *sp-edge*, such that every maximal path in G is a shortest path from a_s to a_t in the original graph. Program $\mathcal{P}_{\text{csp}}$ consists of rule (2) and rules (25)–(27), where rule (25) extends rule (1). The first stratum consists of rules (2) and (25), and computes the length of a shortest path from a_s to each other node using the min predicate *dist*; in particular, $\mathcal{P}_{\text{csp}} \cup \mathcal{D}_{\text{csp}} \models \lceil dist(a,k) \rceil$ if and only if k is the length of a shortest path from a_s to a. Then, the second stratum, consisting of rules (26) and (27), computes the predicate *sp-edge* such that $\mathcal{P}_{\text{csp}} \cup \mathcal{D}_{\text{csp}} \models \textit{sp-edge}(a,a')$ if and only if the edge (a,a') is part of a shortest path from a_s to a_t.

$$source(x) \rightarrow dist(x,0) \tag{25}$$

$$\lceil dist(x,m) \rceil \wedge edge(x,y,n) \wedge \lceil dist(y,m+n) \rceil \wedge target(y) \rightarrow \textit{sp-edge}(x,y) \tag{26}$$

$$\lceil dist(x,m) \rceil \wedge edge(x,y,n) \wedge \lceil dist(y,m+n) \rceil \wedge \textit{sp-edge}(y,z) \rightarrow \textit{sp-edge}(x,y) \tag{27}$$

Example 3.14 (Closeness centrality). The closeness centrality of a node in a strongly connected directed graph G with weighted edges is a measure of how central the node is in the graph [49]; variants of this measure are useful, for instance, for the analysis of market potential. Most commonly, closeness centrality of a node a is defined as $1 \big/ \sum_{a' \text{ node in } G} \Omega(a,a')$, where $\Omega(a,a')$ is the length of a shortest path from a to a'; the sum in the denominator is often called the *farness centrality* of a. We next give an LL-program computing a node of maximal closeness centrality in a given strongly connected directed graph with weighted edges. We encode such a graph as a dataset $\mathcal{D}_{\text{cc}}$ using, as before, a unary object predicate *node* and a ternary exact predicate *edge*. Program $\mathcal{P}_{\text{cc}}$ consists of rules (28)–(36), where *dist*, *fness* and *fness′* are min predicates, and *centre* and *centre′* are object predicates. The first stratum consists of rules (28)–(32). Rules (28) and (29) compute the distance (i.e., the length of a shortest path) between any two nodes. Rules (30)–(32) then compute the farness centrality of each node based on the aforementioned distances; for this, the program exploits the order predicates to iterate over the nodes in the graph while recording the best value obtained so far in the iteration using an auxiliary predicate *fness′*. In the second stratum (rules (33)–(36)), the program uses negation (abbreviated in the LUB expressions $\lceil fness(z,m) \rceil$ and $\lceil fness(y,n) \rceil$) to compute a node of minimal farness centrality (and hence of maximal closeness centrality), which is recorded using the *centre* predicate; the order is again exploited to iterate over nodes, and an auxiliary predicate *centre′* is used to record the current node of the iteration and the node with the best centrality encountered so far.

$$node(x) \rightarrow dist(x,x,0) \tag{28}$$

$$dist(x,y,m) \wedge edge(y,z,n) \rightarrow dist(x,z,m+n) \tag{29}$$

$$node(x) \wedge first(y) \wedge dist(x,y,n) \rightarrow fness'(x,y,n) \tag{30}$$

$$fness'(x,y,m) \wedge next(y,z) \wedge dist(x,z,n) \rightarrow fness'(x,z,m+n) \tag{31}$$

$$fness'(x,y,n) \wedge last(y) \rightarrow fness(x,n) \tag{32}$$

$$first(x) \rightarrow centre'(x,x) \tag{33}$$

$$centre'(x,z) \wedge next(x,y) \wedge \lceil fness(z,m) \rceil \wedge \lceil fness(y,n) \rceil \wedge (n < m) \rightarrow centre'(y,y) \tag{34}$$

$$centre'(x,z) \wedge next(x,y) \wedge \lceil fness(z,m) \rceil \wedge \lceil fness(y,n) \rceil \wedge (m \leq n) \rightarrow centre'(y,z) \tag{35}$$

$$centre'(x,z) \wedge last(x) \rightarrow centre(z) \tag{36}$$

4 DECIDABILITY AND COMPLEXITY UPPER BOUNDS FOR LL-PROGRAMS

In this section, we begin our study of limit-linear $Datalog_{\mathbb{Z}}$. In particular, we show that fact entailment for the language of LL-programs is decidable and provide complexity upper bounds for the full language, as well as for its positive and semi-positive fragments. We show that, for the full language, the problem is in Δ_2^{EXP} in combined and in Δ_2^{P} in data complexity, while, for the (semi-)positive fragment, it is in coNEXP in combined and in coNP in data complexity. Later on, in Section 5, we show that all these bounds are tight.

We start, in Section 4.1, by establishing a useful characterisation of fact entailment that allows us to manipulate only finite structures when computing materialisations. We next study, in Section 4.2, the case of positive LL-programs, where we show decidability via a reduction to the evaluation problem for Presburger sentences and then provide complexity upper bounds by analysing the shape of the Presburger formulas computed by our reduction. In Section 4.3, we first show that the bounds in Section 4.2 also apply to semi-positive LL-programs, and then propose a fact entailment algorithm for arbitrary LL-programs that relies on an entailment oracle for semi-positive LL-programs.

4.1 Characterising Entailment Using Pseudointerpretations

The definition of fact entailment is formulated in terms of the immediate consequence operator $\mathcal{S}$. Unlike usual Datalog, however, this definition cannot be used directly for deciding fact entailment, because all limit-closed interpretations containing at least one limit fact are infinite. In this section, we prove a characterisation of fact entailment in terms of *pseudointerpretations*—that is, certain sets of pseudofacts (recall that pseudofacts are 'facts' where ∞ may occur in place of integers). Pseudointerpretations have a one-to-one correspondence with limit-closed interpretations, and we will see that it is enough to consider only finite pseudointerpretations when deciding fact entailment. Our characterisation is directly applicable only to positive LL-programs, so in this section we assume that all programs are positive.

Positive LL-programs admit an important property analogous to the grounding property of usual (function-free) positive Datalog: all object and guarded numeric (i.e., appearing in positive exact literals) variables may be grounded to constants occurring in the program without affecting the outcomes of fact entailment checks.

Definition 4.1. A positive LL-rule or LL-program is *object-and-guarded-ground* (*OG-ground*) if it has neither object nor guarded numeric variables. The (*result of the*) *canonical OG-grounding* of a positive LL-program $\mathcal{P}$, denoted by $\mathcal{G}(\mathcal{P})$, is the OG-ground program that consists of all OG-ground instances $\rho\sigma$ of rules ρ in $\mathcal{P}$ with σ a substitution mapping all object and guarded numeric variables of ρ to constants mentioned in $\mathcal{P}$.

The first immediate property of OG-ground programs is that all of their numeric terms are either linear or ∞. Moreover, since we consider only normalised programs, in which all positive exact body literals use no unguarded variables or function symbols, all numeric terms in exact body literals in the canonical OG-grounding are integers. As follows directly from the definitions and is formalised next in Lemmas 4.1 and 4.3, positive LL-programs and their canonical OG-groundings are semantically interchangeable, and the sizes of such OG-groundings can be bounded.

For every positive LL-program $\mathcal{P}$, $\mathcal{M}(\mathcal{P}) = \mathcal{M}(\mathcal{G}(\mathcal{P}))$.

The proof of this lemma is given in the appendix.

This lemma implies that fact entailment can be checked on the canonical OG-grounding instead of the original program.

Corollary 4.2. *For every positive LL-program $\mathcal{P}$ and fact γ, $\mathcal{P} \models \gamma$ if and only if $\mathcal{G}(\mathcal{P}) \models \gamma$.*

The following bounds on the canonical OG-grounding immediately follow from the definitions (recall that $[\![E]\!]$ is the size of an expression E assuming that each integer takes unit space).

Lemma 4.3. *The following holds for every positive LL-program $\mathcal{P}$ with c distinct constants, where $u = \max_{\rho\in\mathcal{P}}[\![\rho]\!]$:*

1. $\mathcal{G}(\mathcal{P})$ can be computed in time polynomial in $c^u + \|\mathcal{P}\|$; and

2. $\max_{\rho \in \mathcal{G}(\mathcal{P})} \llbracket \rho \rrbracket$ *is linearly bounded in u.*

Note that Lemma 4.3 immediately implies that $\|\mathcal{G}(\mathcal{P})\|$ and hence also $|\mathcal{G}(\mathcal{P})|$—that is, the number of rules in $\mathcal{G}(\mathcal{P})$—are polynomially bounded in $c^u + \|\mathcal{P}\|$. Lemmas 4.1 and 4.3 allow us to concentrate on OG-ground programs when studying positive LL-programs in this and the next section (keeping in mind the exponential blow-up in the number of rules).

We proceed to the definition of pseudointerpretations. Recall that if a limit-closed interpretation $\mathcal{I}$ contains a limit fact $C(\mathbf{a}, k)$, then either a limit integer $\ell \succeq_C k$ exists such that $C(\mathbf{a}, \ell) \in \mathcal{I}$ and $C(\mathbf{a}, k') \notin \mathcal{I}$ for all $k' \succ_C \ell$, or $C(\mathbf{a}, k') \in \mathcal{I}$ for all $k' \in \mathbb{Z}$. Thus, to characterise the value of C on a tuple of objects $\mathbf{a}$ in $\mathcal{I}$, we just need the corresponding limit integer or the information that no such integer exists.

Definition 4.4. A *pseudointerpretation* is a set $\mathcal{J}$ of pseudofacts such that $\ell_1 = \ell_2$ for all limit pseudofacts $C(\mathbf{a}, \ell_1)$ and $C(\mathbf{a}, \ell_2)$ in $\mathcal{J}$ with $\ell_1, \ell_2 \in \mathbb{Z} \cup \{\infty\}$.

Limit-closed interpretations correspond naturally and one-to-one to pseudointerpretations, so we can equivalently recast the notions of satisfaction and model using pseudointerpretations.

Definition 4.5. 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:

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

It is easily seen that the notion of correspondence defines a bijection—that is, each limit-closed interpretation corresponds to one and only one pseudointerpretation, and vice versa. Hence, we can transfer all the definitions and notations for (limit-closed) interpretations to pseudointerpretations: for example, a pseudointerpretation $\mathcal{J}$ *satisfies* a ground literal λ, written $\mathcal{J} \models \lambda$, if the corresponding limit-closed interpretation $\mathcal{I}$ satisfies λ, and $\mathcal{J}$ is a *pseudomodel* of a program $\mathcal{P}$, written $\mathcal{J} \models \mathcal{P}$, if $\mathcal{I} \models \mathcal{P}$. Also, we write $\mathcal{J}_1 \sqsubseteq \mathcal{J}_2$ for pseudointerpretations $\mathcal{J}_1$ and $\mathcal{J}_2$ if $\mathcal{I}_1 \subseteq \mathcal{I}_2$ for the corresponding limit-closed interpretations $\mathcal{I}_1$ and $\mathcal{I}_2$. When one or both $\mathcal{J}_i$ are singleton, we may sometimes abuse notation and omit the set braces: for example, for a pseudofact γ and pseudointerpretation $\mathcal{J}$, we write $\gamma \sqsubseteq \mathcal{J}$ instead of $\{\gamma\} \sqsubseteq \mathcal{J}$ (which holds if and only if $\mathcal{J} \models \gamma$).

Example 4.6. Let $\mathcal{I}$ be the limit-closed interpretation with facts $B(1)$, $B(2)$, $C(a_1, k)$ for all $k \leq 5$, and $C(a_2, k)$ for all $k \in \mathbb{Z}$, where B is an exact predicate, C is a max predicate, and a_1 and a_2 are objects. Then $\{B(1), B(2), C(a_1, 5), C(a_2, \infty)\}$ is the pseudointerpretation corresponding to $\mathcal{I}$.

We next show how the computation of the materialisations of OG-ground programs can be simulated on pseudointerpretations. In particular, we introduce the immediate consequence operator $\mathcal{T}_{\mathcal{P}}$ of an OG-ground program $\mathcal{P}$, which works on pseudointerpretations and is defined in terms of IP satisfiability. We then show that $\mathcal{T}_{\mathcal{P}}$ simulates the ordinary operator $\mathcal{S}_{\mathcal{P}}$ for limit-closed interpretations. In particular, the transfinite closure $\mathcal{N}(\mathcal{P})$ of $\mathcal{T}_{\mathcal{P}}$, called pseudomaterialisation, is the pseudointerpretation corresponding to the materialisation $\mathcal{M}(\mathcal{P})$. Importantly, we show that $\mathcal{N}(\mathcal{P})$ is always finite, and that its size can be bounded in terms of the size of $\mathcal{P}$.

Definition 4.7. For ρ an OG-ground (and thus positive and limit-linear) rule with body φ and $\mathcal{J}$ a pseudointerpretation, let $\psi(\rho, \mathcal{J})$ be the IP defined as the conjunction of the following comparison atoms:

1. $(0 < 0)$ if φ contains

- an object or exact atom that is not in $\mathcal{J}$,
- a limit atom $C(\mathbf{a}, s)$ such that $C(\mathbf{a}, \ell) \notin \mathcal{J}$ for each $\ell \in \mathbb{Z} \cup \{\infty\}$, or
- a limit atom $C(\mathbf{a}, \infty)$ such that $C(\mathbf{a}, \infty) \notin \mathcal{J}$;

2. $(s \preceq_C \ell)$ for each limit atom $C(\mathbf{a}, s)$ in φ with $C(\mathbf{a}, \ell) \in \mathcal{J}$, $s \neq \infty$, and $\ell \neq \infty$; and
3. each comparison atom in φ.

Rule ρ is *applicable* to pseudointerpretation $\mathcal{J}$ if the IP $\psi(\rho, \mathcal{J})$ is satisfiable.

For an OG-ground rule ρ with head α applicable to a pseudointerpretation $\mathcal{J}$, we write $\delta(\rho, \mathcal{J})$ for the pseudofact defined as follows:

$$\delta(\rho, \mathcal{J}) = \begin{cases} \alpha & \text{if } \alpha \text{ is an object or exact atom,} \\ C(\mathbf{a}, k) & \text{if } \alpha = C(\mathbf{a}, s) \text{ is a limit atom with } s \neq \infty \\ & \qquad \text{and the IOP } (\psi(\rho, \mathcal{J}), \max_C s) \text{ has } k \text{ as the optimal value,} \\ C(\mathbf{a}, \infty) & \text{if } \alpha = C(\mathbf{a}, s) \text{ is a limit atom with } s = \infty \\ & \qquad \text{or the IOP } (\psi(\rho, \mathcal{J}), \max_C s) \text{ is unbounded.} \end{cases}$$

For $\mathcal{P}$ an OG-ground program and $\mathcal{J}$ a pseudointerpretation, let $\mathcal{T}_\mathcal{P}(\mathcal{J})$ be the smallest (with respect to $\sqsubseteq$) pseudointerpretation satisfying the pseudofact $\delta(\rho, \mathcal{J})$ for each rule ρ in $\mathcal{P}$ applicable to $\mathcal{J}$. A *partial pseudomaterialisation* $\mathcal{N}^\kappa$ of $\mathcal{P}$, for κ an ordinal, is a pseudointerpretation defined by induction on κ as follows, where $\sup_{\kappa' < \kappa} \mathcal{N}^{\kappa'}$ is the supremum of all $\mathcal{N}^{\kappa'}$ with $\kappa' < \kappa$, with respect to $\sqsubseteq$:

$$\mathcal{N}^0 = \emptyset, \qquad \mathcal{N}^\kappa = \mathcal{T}_\mathcal{P}(\mathcal{N}^{\kappa'}) \quad \text{if } \kappa = \kappa' + 1, \qquad \mathcal{N}^\kappa = \sup_{\kappa' < \kappa} \mathcal{N}^{\kappa'} \quad \text{if } \kappa \text{ is a limit ordinal.}$$

The *pseudomaterialisation* $\mathcal{N}(\mathcal{P})$ of $\mathcal{P}$ is the pseudointerpretation $\mathcal{N}^{\omega_1}$.

Example 4.8. Let ρ be OG-ground rule (37), where C_1 and C_2 are max predicates.

$$C_1(m) \wedge (2 \leq m) \rightarrow C_2(m + 1) \tag{37}$$

Then, $\psi(\rho, \emptyset) = (0 < 0) \wedge (2 \leq m)$, where the first comparison atom is derived by condition 1 in Definition 4.7 from the literal $C_1(m)$, and the second comparison atom by condition 3 from $(2 \leq m)$. Clearly, IP $\psi(\rho, \emptyset)$ does not have a solution, and hence rule ρ is not applicable to the empty pseudointerpretation $\emptyset$. However, for the pseudointerpretation $\mathcal{J} = \{C_1(3)\}$, we have that $\psi(\rho, \mathcal{J}) = (m \leq 3) \wedge (2 \leq m)$, where the comparison atoms are derived from the corresponding body literals by conditions 2 and 3. IP $\psi(\rho, \mathcal{J})$ has two solutions, one assigning 2 to m and the other assigning 3; thus, rule ρ is applicable to $\mathcal{J}$. Finally, C_2 is max, and the optimal value of the IOP is $\max\{2 + 1, 3 + 1\} = 4$; hence $\delta(\rho, \mathcal{J}) = C_2(4)$. Thus, $\mathcal{T}_{\{\rho\}}(\mathcal{J}) = \{C_2(4)\}$.

Our next goal is to establish a correspondence between (partial) materialisations and (partial) pseudomaterialisations. The following lemma characterises rule applicability in terms of solutions to the corresponding IP.

For each OG-ground rule ρ with body φ and each pseudointerpretation $\mathcal{J}$, a grounding σ of ρ is a solution to the IP $\psi(\rho, \mathcal{J})$ if and only if $\mathcal{J} \models \varphi\sigma$.

The proof of this lemma is given in the appendix.

Using Lemma 4.1, we next show that, for each OG-ground program $\mathcal{P}$, operator $\mathcal{T}_\mathcal{P}$ on pseudointerpretations faithfully simulates the behaviour of operator $\mathcal{S}_\mathcal{P}$ on corresponding limit-closed interpretations.

For each OG-ground program $\mathcal{P}$ and each ordinal κ, the partial materialisation $\mathcal{M}_1^\kappa$ of $\mathcal{P}$ and the partial pseudomaterialisation $\mathcal{N}^\kappa$ of $\mathcal{P}$ correspond to each other.

The proof of this lemma is also given in the appendix.

Lemma 4.1 immediately implies that, for each OG-ground program $\mathcal{P}$, materialisation $\mathcal{M}(\mathcal{P})$ corresponds to pseudomaterialisation $\mathcal{N}(\mathcal{P})$. Therefore, $\mathcal{N}(\mathcal{P})$ is the smallest pseudomodel of

$\mathcal{P}$ with respect to $\sqsubseteq$, and $\mathcal{P} \models \gamma$ if and only if $\mathcal{N}(\mathcal{P}) \models \gamma$ for every fact γ. Also, Lemma 4.1 and Proposition 3.1 imply a useful monotonicity property of the operator $\mathcal{T}_{\mathcal{P}}$.

Corollary 4.9. *The following statements hold for each OG-ground program $\mathcal{P}$:*

1. *$\mathcal{J} \models \mathcal{P}$ if and only if $\mathcal{T}_{\mathcal{P}}(\mathcal{J}) \sqsubseteq \mathcal{J}$ for each pseudointerpretation $\mathcal{J}$, and*
2. *$\mathcal{T}_{\mathcal{P}}(\mathcal{J}_1) \sqsubseteq \mathcal{T}_{\mathcal{P}}(\mathcal{J}_2)$ for each two pseudointerpretations $\mathcal{J}_1$ and $\mathcal{J}_2$ such that $\mathcal{J}_1 \sqsubseteq \mathcal{J}_2$.*

We next discuss the computational properties of the immediate consequence operator and (partial) pseudomaterialisations, which will be useful in the following sections. We begin by establishing fine-grained complexity of applying the immediate consequence operator for pseudointerpretations and bounding the growth of the magnitudes of integers.

Lemma 4.10. *There exist polynomials p_1 and p_2 such that, for every OG-ground program $\mathcal{P}$ and finite pseudointerpretation $\mathcal{J}$ with the maximal magnitudes of integers bounded by $b > 1$ and $b' > 0$, respectively, the following holds, where $u = \max_{\rho \in \mathcal{P}} \llbracket \rho \rrbracket$:*

1. *the magnitudes of integers in $\mathcal{T}_{\mathcal{P}}(\mathcal{J})$ are bounded by $b^{p_1(u)} \cdot b'$; and*
2. *$\mathcal{T}_{\mathcal{P}}(\mathcal{J})$ can be computed in polynomial time with access to an* NP *oracle and in time polynomial in $2^{p_2(u)} + \|\mathcal{P}\| + \|\mathcal{J}\|$.*

Proof. We start by proving claim 1. We need to show that there exists a polynomial p_1 such that the magnitude of the optimal value k of the IOP $(\psi(\rho, \mathcal{J}), \max_C s)$ is bounded by $b^{p_1(u)} \cdot b'$ for each rule $\rho = \varphi \to C(\mathbf{a}, s)$ in $\mathcal{P}$ applicable to $\mathcal{J}$, assuming C is a limit predicate and the IOP is bounded. Consider such a rule $\rho \in \mathcal{P}$, the IP $\psi' = \psi(\rho, \mathcal{J}) \wedge (s \doteq m)$, for m a fresh variable, and the normal form $M\mathbf{n} \leq \mathbf{k}$ of ψ'. On the one hand, k is the optimal value of IOP $(M\mathbf{n} \leq \mathbf{k}, \max_C m)$ by construction. On the other hand, the maximal magnitudes of integers in M and $\mathbf{k}$ are bounded by b^u and $b^u + b'$, respectively, because all the integers in M depend only on the integers in ρ, while integers from $\mathcal{J}$ contribute only to $\mathbf{k}$ and do not occur in multiplications (i.e, do not contribute to the magnitude blow-up during normalisation). Hence, by Corollary 2.10, there is a polynomial q such that the magnitude of k is bounded by $(b^u)^{q(v)} \cdot (b^u + b')$, where $v = |\mathbf{n}|$ is the number of variables in ρ plus one. Since $v \leq \llbracket \rho \rrbracket + 1 \leq u + 1$, the claim follows.

We proceed to claim 2. Let $\mathcal{E}$ be the set of all pseudofacts $\delta(\rho, \mathcal{J})$ such that ρ is a rule in $\mathcal{P}$ applicable to $\mathcal{J}$. By Definition 4.7, $\mathcal{T}_{\mathcal{P}}(\mathcal{J})$ is the smallest pseudointerpretation with respect to $\sqsubseteq$ such that $\mathcal{T}_{\mathcal{P}}(\mathcal{J}) \models \mathcal{E}$, so it can be easily (in polynomial time) computed from $\mathcal{E}$ by removing subsumed limit pseudofacts. Hence, to complete the proof of the lemma, we next argue that set $\mathcal{E}$ can be computed within the required time bounds.

A rule ρ in $\mathcal{P}$ is applicable to $\mathcal{J}$ if and only if the IP $\psi(\rho, \mathcal{J})$ is satisfiable, and, by construction, $\psi(\rho, \mathcal{J})$ can be computed in time polynomial in $\|\rho\| + \|\mathcal{J}\|$, while $\|\psi(\rho, \mathcal{J})\|$ is linearly bounded in $\|\rho\| \cdot b'$. Moreover, IP $\psi(\rho, \mathcal{J})$ and rule ρ have the same variables. Therefore, by Corollary 2.5, applicability of ρ to $\mathcal{J}$ can be checked in NP. Also, by Proposition 2.6, there is a polynomial q such that satisfiability of $\psi(\rho, \mathcal{J})$ can be checked in time polynomial in $2^{q(v)} + \|\rho\| \cdot b'$, where v is the number of variables in ρ; since $v \leq \llbracket \rho \rrbracket$ and $b' \leq \|\mathcal{J}\|$, applicability of ρ to $\mathcal{J}$ can then be checked in time polynomial in $2^{q(\llbracket \rho \rrbracket)} + \|\rho\| \cdot \max(\|\mathcal{J}\|, 1)$ and hence in $2^{q(\llbracket \rho \rrbracket)} + \|\rho\| + \|\mathcal{J}\|$.

Now assume that a rule $\rho = \varphi \to \alpha$ is applicable to $\mathcal{J}$. According to Definition 4.7, computing $\delta(\rho, \mathcal{J})$ is either trivial, when α is object, exact or of the form $C(\mathbf{a}, \infty)$ with limit C, or requires checking the IOP $(\psi(\rho, \mathcal{J}), \max_C s)$ with s the numeric term from α for boundedness and, if it is bounded, computing its optimal value. By Proposition 2.7, the former can be done in time polynomial in $\|\rho\| + \|\mathcal{J}\|$ (recall that $\psi(\rho, \mathcal{J})$ is satisfiable since ρ is applicable), and, by Proposition 2.8, the latter can be done in time polynomial in $\|\rho\| + \|\mathcal{J}\|$ with satisfiability checks, each of which is in NP, and in time polynomial in $2^{q'(\llbracket \rho \rrbracket)} + \|\rho\| + \|\mathcal{J}\|$ for some polynomial q'. Thus, overall, $\delta(\rho, \mathcal{J})$ can be computed with the same bounds.

Finally, to compute $\mathcal{E}$, we need to check applicability of each rule ρ in $\mathcal{P}$ to $\mathcal{J}$ and, if successful, to compute $\delta(\rho, \mathcal{J})$. Since program $\mathcal{P}$ is OG-ground, each rule of $\mathcal{P}$ can contribute at most one fact to $\mathcal{E}$. Hence $|\mathcal{E}| \leq |\mathcal{P}|$, and the rules can be processed one by one, which gives us, together with the previous results, a procedure for computing the immediate consequence $T_{\mathcal{P}}(\mathcal{J})$ within the required time bounds. □

Next, we establish an important property of (partial) pseudomaterialisations: they are always finite, and the number of pseudofacts in each of them can be bounded.

Proposition 4.11. *For each partial pseudomaterialisation $\mathcal{N}^{\kappa}$ of an OG-ground program $\mathcal{P}$, with κ an ordinal, $|\mathcal{N}^{\kappa}| \leq |\mathcal{P}|$.*

Proof. Since $\mathcal{P}$ is OG-ground, all pseudofacts produced by a single rule coincide on their predicate and object arguments. The claim follows from the fact that $\mathcal{N}^{\kappa}$ is a pseudointerpretation and hence has at most one pseudofact per combination of an object or limit predicate and a tuple of objects of appropriate size, while all exact facts in $\mathcal{N}^{\kappa}$ are taken from $\mathcal{P}$. □

Note that the bound established in Proposition 4.11 does not yet imply decidability of fact entailment as it does not restrict the size of the binary representation of a (partial) pseudomaterialisation—it does not bound the magnitudes of the integers that may be involved.

4.2 Positive Programs

In this section, we establish decidability and complexity of fact entailment for positive LL-programs via an encoding of each OG-ground program $\mathcal{P}$ and fact γ as a Presburger sentence that holds if and only if $\mathcal{P} \models \gamma$. Our reduction is based on three main ideas.

1. For each limit atom $C(\mathbf{a}, s)$ in $\mathcal{P}$, we introduce a Boolean variable $def_{C\mathbf{a}}$ to indicate that an atom of the form $C(\mathbf{a}, \ell)$ exists in a pseudomodel of $\mathcal{P}$, a Boolean variable $fin_{C\mathbf{a}}$ to indicate whether ℓ is a (finite) integer or ∞, and a numeric variable $val_{C\mathbf{a}}$ to capture ℓ if it is finite.
2. Each rule of $\mathcal{P}$ is encoded as a universally quantified Presburger formula by replacing each standard atom with its encoding.
3. Entailment of γ by $\mathcal{P}$ is encoded as a Presburger sentence stating that, for every limit-closed interpretation $\mathcal{I}$, either $\mathcal{I}$ does not satisfy some rule in $\mathcal{P}$ or $\mathcal{I}$ satisfies γ.

We begin the description of our encoding by formally introducing the variables $def_{C\mathbf{a}}$, $fin_{C\mathbf{a}}$, and $val_{C\mathbf{a}}$ for limit atoms, as well as analogous variables for other types of atoms, and relating them to the satisfaction of the atoms in a pseudointerpretation.

Definition 4.12. For every v-tuple $\mathbf{a}$ of objects, consider the following variables: Boolean $def_{A\mathbf{a}}$ for each object predicate A of arity v; Boolean $def_{B\mathbf{a}k}$ for each exact predicate B of arity $v+1$ and each integer k; Boolean $def_{C\mathbf{a}}$ and $fin_{C\mathbf{a}}$, and numeric $val_{C\mathbf{a}}$ for each limit predicate C of arity $v+1$.

The *encoding* ξ_{ρ} of an OG-ground rule ρ is the Presburger formula (with the same quantifier block as ρ) that is obtained from ρ by replacing each atom α (both in the body and in the head) by the formula ξ_{α}, defined as follows:

– $\xi_{\alpha} = def_{A\mathbf{a}}$ if α is an object atom $A(\mathbf{a})$;
– $\xi_{\alpha} = def_{B\mathbf{a}k}$ if α is an exact atom $B(\mathbf{a}, k)$ (recall that this is the only possible form for exact atoms because ρ is normalised and OG-ground);
– $\xi_{\alpha} = def_{C\mathbf{a}} \wedge ((s \preceq_C val_{C\mathbf{a}}) \vee \neg fin_{C\mathbf{a}})$ if α is a limit atom $C(\mathbf{a}, s)$ with $s \neq \infty$;
– $\xi_{\alpha} = def_{C\mathbf{a}} \wedge \neg fin_{C\mathbf{a}}$ if α is a limit atom $C(\mathbf{a}, \infty)$; and
– $\xi_{\alpha} = \alpha$ if α is a comparison atom.

The *encoding* $\xi_{\mathcal{P}}$ of an OG-ground program $\mathcal{P}$ is the Presburger formula $\bigwedge_{\rho \in \mathcal{P}} \xi_{\rho}$.

Let $\mathcal{J}$ be a pseudointerpretation, and let σ be an assignment of all such variables $\mathit{def}_{A\mathbf{a}}$, $\mathit{def}_{B\mathbf{a}k}$, $\mathit{def}_{C\mathbf{a}}$, $\mathit{fin}_{C\mathbf{a}}$, and $\mathit{val}_{C\mathbf{a}}$ for the predicates and constants in $\mathcal{J}$. Then, $\mathcal{J}$ and σ *correspond* to each other if the following holds for all A, B, C, $\mathbf{a}$, and k as above:

- $\sigma(\mathit{def}_{A\mathbf{a}}) = \mathsf{true}$ if and only if $A(\mathbf{a}) \in \mathcal{J}$;
- $\sigma(\mathit{def}_{B\mathbf{a}k}) = \mathsf{true}$ if and only if $B(\mathbf{a}, k) \in \mathcal{J}$;
- $\sigma(\mathit{def}_{C\mathbf{a}}) = \mathsf{true}$ if and only if there exists $\ell \in \mathbb{Z} \cup \{\infty\}$ such that $C(\mathbf{a}, \ell) \in \mathcal{J}$; and
- $\sigma(\mathit{fin}_{C\mathbf{a}}) = \mathsf{true}$ and $\sigma(\mathit{val}_{C\mathbf{a}}) = \ell$ if and only if $C(\mathbf{a}, \ell) \in \mathcal{J}$, for every $\ell \in \mathbb{Z}$.

Note that ℓ in the last case of Definition 4.12 ranges over all integers but not ∞, and $\mathcal{J}$ is a pseudointerpretation and thus cannot contain both $C(\mathbf{a}, \infty)$ and $C(\mathbf{a}, \ell)$ for an integer $\ell \in \mathbb{Z}$; therefore, $\sigma(\mathit{def}_{C\mathbf{a}}) = \mathsf{true}$ and $\sigma(\mathit{fin}_{C\mathbf{a}}) = \mathsf{false}$ if and only if $C(\mathbf{a}, \infty) \in \mathcal{J}$.

Also note that each assignment corresponds to exactly one pseudointerpretation. However, each pseudointerpretation may correspond to several assignments since two assignments corresponding to the same pseudointerpretation may differ, for a limit predicate C and objects $\mathbf{a}$, on the value of $\mathit{val}_{C\mathbf{a}}$, if $\mathit{def}_{C\mathbf{a}}$ or $\mathit{fin}_{C\mathbf{a}}$ is set to false in both assignments.

Example 4.13. Consider the OG-ground program $\mathcal{P}$ consisting of the following rules, for B an exact and C a max predicate.

$$\rightarrow B(1) \qquad\qquad B(1) \land C(m) \land (1 \leq m) \rightarrow C(m+1)$$

Then $\xi_{\mathcal{P}}$ is the following Presburger formula:

$$\begin{aligned}&\mathit{def}_{B1} \land \\ &\quad \forall m.\, \mathit{def}_{B1} \land (\mathit{def}_C \land ((m \leq \mathit{val}_C) \lor \neg \mathit{fin}_C)) \land (1 \leq m) \rightarrow \mathit{def}_C \land ((m+1 \leq \mathit{val}_C) \lor \neg \mathit{fin}_C).\end{aligned}$$

The following lemma shows how the correspondence between pseudointerpretations and assignments transfers to the correspondence between programs and their encodings. (Note that in this lemma we abuse terminology and sometimes say that an assignment σ is a solution to a Presburger formula ξ meaning that the restriction of σ to the free variables of ξ is such a solution.)

Let $\mathcal{P}$ be an OG-ground program, and let $\mathcal{J}$ and σ be a pseudointerpretation and a variable assignment, respectively, such that $\mathcal{J}$ and σ correspond to each other. Then, $\mathcal{J} \models \mathcal{P}$ if and only if σ is a solution to $\xi_{\mathcal{P}}$.

The proof of this lemma is given in the appendix.

We are now ready to summarise the main properties of our Presburger encoding (we remind here that an IP is a conjunction of comparison atoms with linear numeric terms).

Lemma 4.14. *For $\mathcal{P}$ an OG-ground program and γ a fact, there exists a Presburger sentence $\xi = \forall \mathbf{n}.\, \exists \mathbf{m}.\, \bigvee_{i=1}^{h} \psi_i$ with each ψ_i an IP such that the following holds, for $v = |\mathbf{n}|$, $d = \max_{i \in [1,h]} [\![\psi_i]\!]$, $r = |\mathcal{P}|$, and $u = \max_{\rho \in \mathcal{P}} [\![\rho]\!]$:*

- *ξ holds if and only if $\mathcal{P} \models \gamma$;*
- *v, h, and d are linearly bounded in ru, $r \cdot 2^u$, and u, respectively; and*
- *each integer in ξ appears in $\mathcal{P}$ or γ.*

Proof. Since fact entailment for positive programs coincides with classical first-order entailment, Lemma 4.2 implies that $\mathcal{P} \models \gamma$ if and only if the Presburger sentence

$$\xi_0 = \forall \mathbf{n}.\, \neg \xi_{\mathcal{P}} \lor \xi_{\gamma}$$

holds, where $\mathbf{n}$ is the tuple of all variables $\mathit{def}_{A\mathbf{a}}$, $\mathit{def}_{B\mathbf{a}k}$, $\mathit{def}_{C\mathbf{a}}$, $\mathit{fin}_{C\mathbf{a}}$, and $\mathit{val}_{C\mathbf{a}}$ occurring in $\xi_{\mathcal{P}}$ and ξ_{γ}. Next, we transform, in several equivalence-preserving steps, ξ_0 to a Presburger sentence ξ that satisfies the required properties.

Let first ξ_1 be the Presburger sentence obtained from ξ_0 by converting each top-level conjunct ξ_ρ of $\xi_\mathcal{P}$, for $\rho \in \mathcal{P}$, into the form

$$\forall \mathbf{m}_\rho. \bigwedge_{i=1}^{h_\rho} \zeta_\rho^i$$

with each ζ_ρ^i a disjunction of literals, where $\mathbf{m}_\rho$ are all the (numeric) variables of ρ (i.e., the quantifier-free part of each ξ_ρ is converted to conjunctive normal form). By moving all quantifiers in ξ_1 to the front and pushing negations inwards, we obtain the sentence

$$\xi_2 = \forall \mathbf{n}.\, \exists \mathbf{m}. \left(\bigvee_{\rho \in \mathcal{P}} \bigvee_{i=1}^{h_\rho} \chi_\rho^i \right) \vee \xi'_\gamma,$$

where $\mathbf{m}$ is the (disjoint) union of all $\mathbf{m}_\rho$ (assuming without loss of generality that different rules in $\mathcal{P}$ use different variables), each χ_ρ^i is the negation normal form of $\neg\zeta_\rho^i$, and ξ'_γ is the disjunctive normal form of ξ_γ. Finally, let ξ be obtained from ξ_2 by rewriting each negated comparison atom to the equivalent (positive) comparison atom (e.g., $\neg(s_1 \leq s_2)$ is rewritten to $(s_2 < s_1)$).

We next claim that ξ satisfies all the required properties. First, ξ is of the required form, because $\mathcal{P}$ is an OG-ground program and hence all the terms are linear. Second, ξ holds if and only if $\mathcal{P} \models \gamma$ because ξ is equivalent to ξ_0 by construction. Third, $v = |\mathbf{n}|$ is linearly bounded in $ru = |\mathcal{P}| \cdot (\max_{\rho \in \mathcal{P}} [\![\rho]\!])$, because it is bounded by $[\![\mathcal{P}]\!]$ and $[\![\mathcal{P}]\!] \leq |\mathcal{P}| \cdot (\max_{\rho \in \mathcal{P}} [\![\rho]\!])$. Fourth, by construction, $\Sigma_{\rho \in \mathcal{P}}\, h_\rho$ plus the constant number of conjunctions in ξ'_γ—that is, h—is linearly bounded in $r \cdot 2^u$, while each $[\![\chi_\rho^i]\!]$ and each $[\![\chi]\!]$ for χ a conjunction in ξ'_γ—and hence d—is linearly bounded in u. Finally, also by construction, each integer in ξ appears in $\mathcal{P}$ or γ. □

The following lemma bounds the magnitudes of variable values in the Presburger sentence from Lemma 4.14.[3]

Lemma 4.15. *There exist polynomials p_1 and p_2 such that, for every Presburger sentence*

$$\xi = \forall \mathbf{n}.\, \exists \mathbf{m}. \bigvee_{i=1}^{h} \psi_i$$

with each ψ_i an IP and the maximal magnitude of an integer in ξ bounded by $b > 1$, ξ holds if and only if it holds over integers with magnitudes bounded by

$$b^{p_1(h+v) \cdot 2^{p_2(d)}},$$

where $v = |\mathbf{n}|$ and $d = \max_{i \in [1,h]} [\![\psi_i]\!]$.

Proof. Note first that the number of comparison atoms, the number of variables, and the number of integers in each comparison atom in each ψ_i with $i \in [1, h]$ are all bounded by d. So, as mentioned in Section 2.3, each IP ψ_i can be rewritten to an IP ψ'_i in normal form that has at most d inequalities, at most d variables, and the magnitudes of all integers bounded by b^d. By Proposition 2.9, the set of solutions to each ψ'_i—and hence ψ_i—is a semi-linear set of vectors from $\mathbb{Z}^d$ that can be represented as the union of 2^d linear sets $S(\boldsymbol{\ell}_j, L_j)$, $j \in [1, 2^d]$, where the magnitudes of integers in each $\boldsymbol{\ell}_j$ and L_j are bounded by $2^{O(d \log d)} \cdot b^{d^2}$ and hence by $b^{O(d^2)}$. Consequently, solutions to the disjunction $\bigvee_{i=1}^{h} \psi_i$ form a semi-linear set R made up of $h \cdot 2^d$ linear sets of vectors from $\mathbb{Z}^{v+|\mathbf{m}|}$, each of which can again be represented as a linear set $S(\boldsymbol{\ell}, L)$ with the magnitudes of integers in $\boldsymbol{\ell}$ and L bounded by $2^{O(d \log d)} \cdot b^{d^2}$ and hence by $b^{O(d^2)}$. Hence, solutions to the Presburger formula

[3]The proof of Lemma 4.15 was provided by Christoph Haase in private communication.

$\chi = \exists \mathbf{m}. \bigvee_{i=1}^{h} \psi_i$ are the projections of all vectors in R on variables $\mathbf{n}$, and hence are a union R' of $h \cdot 2^d$ linear sets of vectors from $\mathbb{Z}^v$, where the integers in the representation of each of these sets satisfy the same bounds as for R. Now, Theorem 2.11 implies that the set of all satisfying assignments to the formula $\neg\chi$—that is, the complement of R'—is a union R'' of linear sets in whose representations the magnitudes of all integers can be bounded by $(b^{O(d^2)})^{h \cdot 2^d \cdot O(v^4)}$. Since $\neg\chi$ has a satisfying assignment if and only if it has a satisfying assignment involving only numbers from the representation of R'', it follows that $\neg\xi$ holds if and only if it holds over integers with magnitudes bounded by $(b^{O(d^2)})^{h \cdot 2^d \cdot O(v^4)}$. This implies the claim since ξ is a sentence. □

We now combine the above results to establish upper bounds to the complexity of fact entailment. Lemmas 4.14 and 4.15 provide us with bounds on the size of counter-pseudomodels for entailment.

Lemma 4.16. *There exist polynomials p_1 and p_2 such that, for every OG-ground program $\mathcal{P}$ and fact γ with the maximal magnitude of an integer in $\mathcal{P}$ and γ bounded by $b > 1$, $\mathcal{P} \not\models \gamma$ if and only if there is a pseudomodel $\mathcal{J}$ of $\mathcal{P}$ such that $\mathcal{J} \not\models \gamma$, $|\mathcal{J}| \leq r$, and the magnitude of each integer in $\mathcal{J}$ is bounded by*

$$b^{p_1(r) \cdot 2^{p_2(u)}},$$

where $r = |\mathcal{P}|$ and $u = \max_{\rho \in \mathcal{P}} [\![\rho]\!]$.

Proof. The backward direction is trivial since $\mathcal{P}$ is positive, so we concentrate on the forward direction. To this end, consider an OG-ground program $\mathcal{P}$ and a fact γ such that $\mathcal{P} \not\models \gamma$. By Lemma 4.14, there exists a Presburger sentence $\xi = \forall \mathbf{n}. \exists \mathbf{m}. \bigvee_{i=1}^{h} \psi_i$ that does not hold and satisfies the following properties, for $v = |\mathbf{n}|$ and $d = \max_{i \in [1,h]} [\![\psi_i]\!]$:

- v, h, and d are linearly bounded by ru, $r \cdot 2^u$, and u, respectively; and
- each integer in ξ appears in $\mathcal{P}$ or γ.

The latter property guarantees that b is a bound on magnitudes of all integers in ξ. Therefore, by Lemma 4.15, there are polynomials q_1 and q_2, and an assignment σ to variables $\mathbf{n}$ such that σ is not a solution to $\exists \mathbf{m}. \bigvee_{i=1}^{h} \psi_i$ and the magnitudes of all numbers in the range of σ are bounded by

$$b^{q_1(h+v) \cdot 2^{q_2(d)}},$$

which immediately implies existence of polynomials p_1 and p_2 with the required properties.

We are left to show how to construct a pseudomodel $\mathcal{J}$ with $|\mathcal{J}| \leq r = |\mathcal{P}|$ and the same bound on integers. By construction of ξ in Lemma 4.14, σ is a solution to $\xi_{\mathcal{P}}$ and not a solution to ξ_{γ}. Now, let σ' be the extension of σ to all variables introduced in Definition 4.12 that assigns false to all Boolean and 0 to all integer variables outside of the domain of σ, and let $\mathcal{J}'$ be the pseudomodel corresponding to σ'. By Lemma 4.2, $\mathcal{J}' \models \mathcal{P}$ and $\mathcal{J}' \not\models \gamma$. Moreover, by definition, the bound on the magnitudes of integers propagates from σ to $\mathcal{J}'$. Let $\mathcal{J}$ be the subset of $\mathcal{J}'$ consisting of all pseudofacts that are either identical to the head of some rule in $\mathcal{P}$ or differ from such a head only in the numeric position of a limit atom. On the one hand, we still have $\mathcal{J} \models \mathcal{P}$ and $\mathcal{J} \not\models \gamma$. On the other hand, $|\mathcal{J}| \leq r$ by construction. So, $\mathcal{J}$ satisfies all the required properties. □

We are ready to present our first algorithm for fact entailment: by Corollary 4.2 and Lemma 4.16, nondeterministic Algorithm 1 decides fact entailment for positive LL-programs. Next, we show that the algorithm works within the announced upper complexity bounds for the problem. As already mentioned, in Section 5 (Theorem 5.4), we will see that these bounds are tight.

Theorem 4.17. *The fact entailment problem for positive LL-programs is in* coNEXP *in combined and in* coNP *in data complexity.*

ALGORITHM 1: Fact Entailment for Positive LL-Programs

```
Input: positive LL-program $\mathcal{P}$ and fact $\gamma$
Output: true if and only if $\mathcal{P} \models \gamma$
1 compute the canonical OG-grounding $\mathcal{G}(\mathcal{P})$ of $\mathcal{P}$
2 universally guess a pseudointerpretation $\mathcal{J}$ with bounds given in Lemma 4.16 for $\mathcal{G}(\mathcal{P})$ as $\mathcal{P}$
3 return true if $\mathcal{J} \not\models \mathcal{G}(\mathcal{P})$ or $\mathcal{J} \models \gamma$ and false otherwise
```

Proof. As discussed before the theorem, we are left to show that Algorithm 1 runs within the required complexity bounds.

We start with combined complexity. First, by claim 1 of Lemma 4.3, line 1 requires, for c the number of distinct constants in $\mathcal{P}$ and $u = \max_{\rho \in \mathcal{P}} [\![\rho]\!]$, time polynomial in $c^u + \|\mathcal{P}\|$—that is, exponential in $\|\mathcal{P}\|$—and hence produces an OG-ground program $\mathcal{G}(\mathcal{P})$ such that $\|\mathcal{G}(\mathcal{P})\|$ is exponentially bounded in $\|\mathcal{P}\|$. Moreover, since $\max_{\rho \in \mathcal{G}(\mathcal{P})} [\![\rho]\!]$ is linearly bounded in u by claim 2 of Lemma 4.3 and each integer in $\mathcal{G}(\mathcal{P})$ is inherited from $\mathcal{P}$, the magnitude of the integers in pseudointerpretation $\mathcal{J}$ guessed in line 2 is doubly-exponentially bounded in $\|\mathcal{P}\| + \|\gamma\|$ by Lemma 4.16; representing such integers requires exponentially many bits. Furthermore, since $|\mathcal{J}| \leq |\mathcal{G}(\mathcal{P})|$ also by Lemma 4.16, the size $\|\mathcal{J}\|$ can be exponentially bounded in $\|\mathcal{P}\| + \|\gamma\|$. Recall here that all the polynomials existence of which is claimed in this paper (e.g., in Lemmas 4.3 and 4.16), are computable in polynomial time. Finally, by claim 1 of Corollary 4.9, line 3 amounts to checking $\mathcal{T}_{\mathcal{G}(\mathcal{P})}(\mathcal{J}) \not\sqsubseteq \mathcal{J}$ and $\gamma \sqsubseteq \mathcal{J}$, which can both be done in time polynomial in $\|\mathcal{T}_{\mathcal{G}(\mathcal{P})}(\mathcal{J})\| + \|\mathcal{J}\|$ in a straightforward way (when $\mathcal{T}_{\mathcal{G}(\mathcal{P})}(\mathcal{J})$ is computed); moreover, $\mathcal{T}_{\mathcal{G}(\mathcal{P})}(\mathcal{J})$ can be computed in time polynomial in $2^{p(\max_{\rho \in \mathcal{G}(\mathcal{P})} [\![\rho]\!])} + \|\mathcal{G}(\mathcal{P})\| + \|\mathcal{J}\|$, for some polynomial p, by claim 2 of Lemma 4.10, which is exponential in $\|\mathcal{P}\| + \|\gamma\|$ by Lemma 4.3 and the above observation on $\|\mathcal{J}\|$. Hence, we conclude that line 3 requires exponential time in $\|\mathcal{P}\| + \|\gamma\|$. Therefore, overall, the algorithm works in coNEXP, as required.

We are left to show the coNP data complexity bound of the algorithm, in which case $\mathcal{P} = \mathcal{P}' \cup \mathcal{D}$, for a positive LL-program $\mathcal{P}'$ and a dataset $\mathcal{D}$, and only $\mathcal{D}$ and γ are considered as input. The bound can be shown in the same way as the combined complexity bound, except that $\|\mathcal{G}(\mathcal{P})\|$ and $\|\mathcal{J}\|$ can now be polynomially bounded in $\|\mathcal{D}\|$, because $u = \max_{\rho \in \mathcal{P}} [\![\rho]\!] = \max_{\rho \in \mathcal{P}'} [\![\rho]\!]$ does not depend on $\mathcal{D}$ and γ (recall that we assume that all predicates used in $\mathcal{D}$ and γ are also mentioned in $\mathcal{P}'$); in particular, since the only parameter in the second exponent in the bound of Lemma 4.16 is u, the magnitudes of integers in $\mathcal{J}$ are bounded only exponentially in $\|\mathcal{D}\| + \|\gamma\|$, and hence these integers can be represented using polynomially many bits. □

In the following corollary, which will be useful later when analysing programs with negation, we emphasise that we do not need to fix the whole program $\mathcal{P}'$ to obtain the coNP upper bound, since it is enough to bound only the maximal size of a rule.

Corollary 4.18. *There exists a polynomial p such that entailment of a fact γ by an OG-ground program $\mathcal{P}$ can be decided in* coNP, *provided the input consists of $\mathcal{P}$, γ, and the number $2^{p(u)}$ written in unary, for $u = \max_{\rho \in \mathcal{P}} [\![\rho]\!]$.*

To prove the complexity bounds for positive LL-programs in Theorem 4.17, it was enough to guarantee the existence of a pseudointerpretation of appropriate size that witnesses nonentailment. To obtain the bounds for arbitrary LL-programs in Section 4.3, however, we will also require that the smallest (with respect to $\sqsubseteq$) pseudomodel of an OG-ground program—that is, the pseudomaterialisation—possesses similar size bounds, and, moreover, that this pseudomodel can be effectively computed. Note that the former is not an immediate consequence of Lemma 4.16, because the pseudomaterialisation may have integers in places where the pseudointerpretation from Lemma 4.16 has ∞; furthermore, the pseudomaterialisation may have integers with magnitudes larger than those of the corresponding integers in the pseudointerpretation, even if they are smaller according to the order $\preceq_C$. Therefore, we devote the rest of this section to establishing the properties of pseudomaterialisations. We start by extending the bounds of Lemma 4.16.

Lemma 4.19. *There exist polynomials p_1 and p_2 such that, for every OG-ground program $\mathcal{P}$ with the maximal magnitude of an integer in $\mathcal{P}$ bounded by $b > 1$, the magnitudes of all integers in the pseudomaterialisation $\mathcal{N}(\mathcal{P})$ are bounded by*

$$b^{p_1(r)\cdot 2^{p_2(u)}},$$

where $r = |\mathcal{P}|$ and $u = \max_{\rho\in\mathcal{P}}[\![\rho]\!]$.

Proof. Consider an arbitrary OG-ground program $\mathcal{P}$. We first claim that $\mathcal{P}$ has a pseudomodel $\mathcal{J}$ with magnitudes of all integers bounded by

$$f(b, r, u) = b^{p_1'(r)\cdot 2^{p_2'(u)}},$$

for some polynomials p_1' and p_2', and such that for each pseudofact $C(\mathbf{a}, \infty)$ in $\mathcal{J}$ we have that $C(\mathbf{a}, k) \notin \mathcal{N}(\mathcal{P})$ for all $k \in \mathbb{Z}$ (i.e., either $C(\mathbf{a}, \infty) \in \mathcal{N}(\mathcal{P})$ or $C(\mathbf{a}, k) \notin \mathcal{N}(\mathcal{P})$ for all $k \in \mathbb{Z} \cup \{\infty\}$). This can be proven in a way that is very similar to the proof of Lemma 4.16. The only differences are as follows:

- to simulate fact entailment, we use a dummy fact $\gamma = A$ over a nullary predicate A that is not used in $\mathcal{P}$, which implies, in particular, $\mathcal{P} \not\models A$;
- to guarantee the requirement on pseudofacts with ∞, instead of the Presburger formula $\xi_{\mathcal{P}}$ as in Definition 4.12 we use the formula $\xi'_{\mathcal{P}}$ that is defined in exactly the same way as $\xi_{\mathcal{P}}$, except that ξ'_α, for a limit atom $\alpha = C(\mathbf{a}, s)$ with $s \neq \infty$ that occurs in the head of a rule, is defined not as $\xi_\alpha = \mathit{def}_{C\mathbf{a}} \wedge ((s \preceq_C \mathit{val}_{C\mathbf{a}}) \vee \neg \mathit{fin}_{C\mathbf{a}})$, but in a more restrictive way:

$$\xi'_\alpha = \begin{cases} \mathit{def}_{C\mathbf{a}} \wedge \mathit{fin}_{C\mathbf{a}} \wedge (s \preceq_C \mathit{val}_{C\mathbf{a}}) & \text{if } C(\mathbf{a}, \infty) \notin \mathcal{N}(\mathcal{P}),\\ \mathit{def}_{C\mathbf{a}} \wedge \neg \mathit{fin}_{C\mathbf{a}} & \text{otherwise.}\end{cases}$$

Since each assignment corresponding to $\mathcal{N}(\mathcal{P})$ is easily seen to be a solution to $\xi'_{\mathcal{P}}$, $\xi'_{\mathcal{P}}$ is satisfiable. Hence, the sentence $\xi'_0 = \forall \mathbf{n}.\, \neg\xi'_{\mathcal{P}} \vee \xi_\gamma$, defined analogously to ξ_0 in the proof of Lemma 4.14, does not hold, and, by an argument analogous to the one in the proofs of Lemmas 4.14 and 4.16, there is a solution to $\xi'_{\mathcal{P}}$ whose corresponding pseudointerpretation $\mathcal{J}$ satisfies the required bound on the magnitudes of integers as well as the requirement on pseudofacts with ∞. Finally, since $\xi'_{\mathcal{P}}$ is a strengthening of $\xi_{\mathcal{P}}$, by Lemma 4.2, $\mathcal{J}$ is a pseudomodel of $\mathcal{P}$. Note that, by Lemma 4.1 and monotonicity of positive programs, $\mathcal{N}(\mathcal{P}) \sqsubseteq \mathcal{J}$; thus, for each limit pseudofact $C(\mathbf{a}, k) \in \mathcal{N}(\mathcal{P})$ with $k \in \mathbb{Z}$ there is some $C(\mathbf{a}, k') \in \mathcal{J}$ such that $k' \in \mathbb{Z}$ and $k \preceq_C k'$. This implies that the magnitude of k is bounded by that of k', and hence by $f(b, r, u)$, but only if both numbers are on the same side of 0. In the rest of the proof, we show that we can also bound the magnitude of k if k and k' are on the opposite sides of 0.

Consider now the OG-ground program $\mathcal{P}'$ that extends $\mathcal{P}$ with all pseudofacts with ∞ in $\mathcal{N}(\mathcal{P})$ (recall that $\mathcal{N}(\mathcal{P})$ is a finite set by Proposition 4.11, so $\mathcal{P}'$ is indeed a program). On the one hand, $\mathcal{P}$ and $\mathcal{P}'$ are semantically equivalent—that is, $\mathcal{N}(\mathcal{P}) = \mathcal{N}(\mathcal{P}')$—so we can concentrate on $\mathcal{P}'$ when proving this lemma. On the other hand, $\mathcal{P}'$ has the following important property: for every partial pseudomaterialisation $\mathcal{N}^\kappa$ of $\mathcal{P}'$ with $\kappa > 0$, a pseudofact with ∞ is in $\mathcal{N}^\kappa$ if and only if it is in $\mathcal{P}'$. This property, together with the fact that $\mathcal{N}^\kappa \sqsubseteq \mathcal{N}(\mathcal{P}') \sqsubseteq \mathcal{J}$ and the restrictions on $\mathcal{J}$, implies that $k \prec_C 0$ for every limit fact $C(\mathbf{a}, k)$ in $\mathcal{N}^\kappa$ such that $k \in \mathbb{Z}$ and $|k| > f(b, r, u)$. Therefore, since $C(\mathbf{a}, k) \in \mathcal{N}^\kappa$ implies $k \preceq_C k'$ for all $C(\mathbf{a}, k') \in \mathcal{N}^{\kappa'}$ with $\kappa < \kappa'$ by monotonicity

ALGORITHM 2: Pseudomaterialisations of OG-Ground Programs

```
Input: OG-ground program $\mathcal{P}$
Output: pseudomaterialisation $\mathcal{N}(\mathcal{P})$
1  compute the bound $b'$ on the magnitudes of integers in $\mathcal{N}(\mathcal{P})$ as in Lemma 4.19
2  set $\mathcal{J} := \emptyset$
3  foreach rule $\varphi \to \gamma$ in $\mathcal{P}$ with object or exact fact $\gamma$ such that $\mathcal{P} \models \gamma$ do
4      add $\gamma$ to $\mathcal{J}$
5  foreach rule $\varphi \to C(\mathbf{a}, s)$ in $\mathcal{P}$ with $C$ a limit predicate do
6      search for the greatest, with respect to $\preceq_C$, integer $k \in [-b' - 1, b' + 1]$ such that $\mathcal{P} \models C(\mathbf{a}, k)$
7      if such $k$ exists then
8          if $k \in [-b', b']$ then add $C(\mathbf{a}, k)$ to $\mathcal{J}$
9          else add $C(\mathbf{a}, \infty)$ to $\mathcal{J}$
10 return $\mathcal{J}$
```

of $\mathcal{T}_{\mathcal{P}'}$, $f(b, r, u) < b^{\kappa} < b^{\kappa+1}$ for the maximal magnitudes b^{κ} and $b^{\kappa+1}$ of integers in $\mathcal{N}^{\kappa}$ and $\mathcal{N}^{\kappa+1}$, respectively, may happen only if there is a limit fact $C(\mathbf{a}, k)$ in $\mathcal{N}^{\kappa+1}$ such that $|k| = b^{\kappa+1}$ and there is no $\ell \in \mathbb{Z} \cup \{\infty\}$ with $C(\mathbf{a}, \ell) \in \mathcal{N}^{\kappa}$—that is, the maximal magnitude can grow beyond $f(b, r, u)$ in the sequence of partial pseudomaterialisations only when a new tuple of objects is introduced to a limit predicate by the immediate consequence operator (note here that all integers in $\mathcal{N}^{\kappa}$ with κ a limit ordinal are inherited from the previous partial pseudomaterialisations). However, the number of such introductions is bounded by $r = |\mathcal{P}|$, because each such predicate-tuple pair in $\mathcal{N}(\mathcal{P})$ is also present in the head of a rule in $\mathcal{P}$. Let κ_0 denote the last ordinal with $b^{\kappa_0} \leq f(b, r, u)$ and by κ_i, for $i \in [1, r']$ with $r' \leq r$, all the ordinals for which $b^{\kappa_{i-1}} < b^{\kappa_i}$. To conclude, by claim 1 of Lemma 4.10, there exists a polynomial q such that $b^{\kappa_i} \leq b^{q(u)} \cdot \max(b^{\kappa_{i-1}}, 1)$. Thus, the maximal magnitude of an integer in $\mathcal{N}(\mathcal{P})$ is bounded by

$$b^{\kappa_{r'}} \leq (b^{q(u)})^r \cdot f(b, r, u) = b^{r \cdot q(u)} \cdot b^{p_1'(r) \cdot 2^{p_2'(u)}},$$

which implies the claim of the lemma. □

We are ready to present Algorithm 2 for computing the pseudomaterialisation of an OG-ground program. Its correctness is immediate by Lemma 4.19 and the definition of pseudomaterialisations. Next, we establish the complexity bounds for this algorithm.

Theorem 4.20. *There exists a polynomial p such that, for every OG-ground program $\mathcal{P}$, the pseudomaterialisation $\mathcal{N}(\mathcal{P})$ can be computed in time polynomial in $2^{p(u)} + \|\mathcal{P}\|$ with access to an* NP *oracle, where $u = \max_{\rho \in \mathcal{P}} [\![\rho]\!]$.*

Proof. As already seen, Algorithm 2 computes the pseudomaterialisation $\mathcal{N}(\mathcal{P})$ of an input OG-ground program $\mathcal{P}$. Next, we show that the algorithm can be run within the stated time bounds. First, the computation of the bound $b' = b^{p_1(r) \cdot 2^{p_2(u)}}$ in line 1, for b the maximum of 2 and the maximal magnitude of an integer in $\mathcal{P}$, and for $r = |\mathcal{P}|$, can be done in time polynomial in $\log b \cdot p_1(r) \cdot 2^{p_2(u)}$—that is, polynomial in $2^{p_2(u)} + \|\mathcal{P}\|$—by a standard exponentiation method (e.g., using exponentiation by squaring). Moreover, the greatest k in line 6 can be found by binary search (if such k exists), which requires logarithmically many fact entailment checks in b'—that is, polynomially many in $\log b \cdot p_1(r) \cdot 2^{p_2(u)}$ and hence in $2^{p_2(u)} + \|\mathcal{P}\|$. Since this should be done once for each rule in $\mathcal{P}$ whose head is a limit atom, the algorithm boils down to polynomially many fact entailment checks in $2^{p_2(u)} + \|\mathcal{P}\|$, each of which can be done by an NP oracle by Corollary 4.18. □

4.3 Semi-Positive and Arbitrary LL-Programs

In this section, we first extend the upper bounds from the previous section to semi-positive LL-programs by showing that each such program can be transformed into an OG-ground (and hence positive) program while preserving fact entailment. Building on these results, we then develop a fact entailment algorithm for arbitrary LL-programs.

To this end, we next introduce the notion of a *reduct* of a semi-positive LL-program $\mathcal{P}$. This notion builds upon a direct extension of the canonical OG-grounding to semi-positive programs, which is also denoted by $\mathcal{G}(\mathcal{P})$. The reduct of $\mathcal{P}$ is then obtained by first computing $\mathcal{G}(\mathcal{P})$ and consequently eliminating all negative literals using entailment-preserving transformations.

Definition 4.21. For $\mathcal{P}$ a semi-positive LL-program, let $\mathcal{G}(\mathcal{P})$ be the set of all rules $\rho\sigma$ with $\rho \in \mathcal{P}$ and σ a substitution mapping all object and guarded numeric variables of ρ to constants in $\mathcal{P}$. The *reduct* $\mathcal{R}(\mathcal{P})$ of $\mathcal{P}$ is the OG-ground program obtained from $\mathcal{G}(\mathcal{P})$ by modifying it as follows, for each negative literal $\mathsf{not}\,\alpha$ in each rule ρ in $\mathcal{G}(\mathcal{P})$:

– if α is an object atom, then
  - if $\alpha \in \mathcal{G}(\mathcal{P})$ then delete ρ,
  - otherwise delete $\mathsf{not}\,\alpha$ from ρ;
– if $\alpha = B(\mathbf{a}, s)$ is an exact atom, then consider all integers $k_1 < \cdots < k_h$, $h \in \mathbb{N}$, such that $B(\mathbf{a}, k_i) \in \mathcal{G}(\mathcal{P})$ for each $i \in [1, h]$ and
  - if $h > 0$ then replace ρ with $h+1$ rules obtained from ρ by replacing $\mathsf{not}\,\alpha$ with the comparison atoms $(s < k_1)$, $(k_{i-1} < s < k_i)$ for $i \in [2, h]$, and $(k_h < s)$,
  - otherwise delete $\mathsf{not}\,\alpha$ from ρ; and
– if $\alpha = C(\mathbf{a}, s)$ is a limit atom, then
  - if $C(\mathbf{a}, \infty) \in \mathcal{G}(\mathcal{P})$ then delete ρ,
  - otherwise if $s \neq \infty$ and there is $C(\mathbf{a}, \ell) \in \mathcal{G}(\mathcal{P})$ with $\ell \in \mathbb{Z}$ then replace $\mathsf{not}\,\alpha$ in ρ with the comparison atom $(k \prec_C s)$ for $k = \max_C\{\ell \in \mathbb{Z} \mid C(\mathbf{a}, \ell) \in \mathcal{G}(\mathcal{P})\}$,
  - otherwise delete $\mathsf{not}\,\alpha$ from ρ.

Note that, while we could already call the semi-positive program $\mathcal{G}(\mathcal{P})$ OG-ground—after all, it is produced by a direct extension of the canonical OG-grounding to semi-positive programs—it will be more convenient to maintain the convention from Section 4.1 and restrict the term OG-ground to positive programs. Note also that $\mathcal{R}(\mathcal{P}) = \mathcal{G}(\mathcal{P})$ for each positive LL-program $\mathcal{P}$.

We next show that reducts play the same role for semi-positive LL-programs as canonical OG-groundings for positive LL-programs, in the sense that reducts are interchangeable with the original programs and can be used for checking fact entailment within the required complexity bounds.

The following lemma extends the result of Lemma 4.1 for positive LL-programs by showing that reducts preserve the semantics of semi-positive LL-programs.

For every semi-positive LL-program $\mathcal{P}$, $\mathcal{M}(\mathcal{P}) = \mathcal{M}(\mathcal{R}(\mathcal{P}))$.

The proof of this lemma is given in the appendix.

Similarly to Corollary 4.2 for positive LL-programs, the following corollary establishes that fact entailment can be checked on the reduct instead of the original semi-positive LL-program.

Corollary 4.22. *For every semi-positive LL-program $\mathcal{P}$ and fact γ, $\mathcal{P} \models \gamma$ if and only if $\mathcal{R}(\mathcal{P}) \models \gamma$.*

Having established a semantic connection between semi-positive LL-programs and their reducts, we next extend the results of Lemma 4.3, showing that reducts have the same computational properties as canonical OG-groundings.

Lemma 4.23. *The following holds for every semi-positive LL-program $\mathcal{P}$ with c distinct constants, where $u = \max_{\rho \in \mathcal{P}} [\![\rho]\!]$:*

1. $\mathcal{R}(\mathcal{P})$ can be computed in time polynomial in $c^u + \|\mathcal{P}\|$, and
2. $\max_{\rho \in \mathcal{R}(\mathcal{P})} [\![\rho]\!]$ is linearly bounded in u.

Proof. We start with claim 1. Note that $\|\mathcal{R}(\mathcal{P})\|$ is linearly bounded in $|\mathcal{R}(\mathcal{P})| \cdot \max_{\rho \in \mathcal{R}(\mathcal{P})} \|\rho\|$ and hence polynomially bounded in $|\mathcal{R}(\mathcal{P})| + \|\mathcal{P}\|$, since $\max_{\rho \in \mathcal{R}(\mathcal{P})} \|\rho\|$ is polynomially bounded in $\|\mathcal{P}\|$ by construction. We next show that $|\mathcal{R}(\mathcal{P})|$ is polynomially bounded in $c^u + |\mathcal{P}|$. Indeed, for every rule $\rho \in \mathcal{G}(\mathcal{P})$, reduct $\mathcal{R}(\mathcal{P})$ contains at most $(h+1)^d$ rules, where d is the number of negative exact literals in the body of ρ and h is the maximal number of distinct facts in $\mathcal{G}(\mathcal{P})$ over a fixed exact predicate and a fixed tuple of objects; note that h is bounded by c since each exact predicate has just one numeric position. Furthermore, $\mathcal{G}(\mathcal{P})$ has at most c^v rules for every rule in $\mathcal{P}$, where v is the maximal number of variables in such a rule, and hence $|\mathcal{G}(\mathcal{P})|$ is bounded by $|\mathcal{P}| \cdot c^v$. Since d and v are both bounded by u, $|\mathcal{R}(\mathcal{P})| \leq |\mathcal{P}| \cdot c^v \cdot (c+1)^d$ is polynomially bounded in $c^u + |\mathcal{P}|$, as required, and hence $\|\mathcal{R}(\mathcal{P})\|$ is polynomial in $c^u + \|\mathcal{P}\|$. Finally, we can compute $\mathcal{R}(\mathcal{P})$ in time polynomial in $\|\mathcal{R}(\mathcal{P})\|$ by directly applying the definition.

For claim 2, $\max_{\rho \in \mathcal{R}(\mathcal{P})} [\![\rho]\!]$ is linearly bounded in $\max_{\rho \in \mathcal{G}(\mathcal{P})} [\![\rho]\!]$, and hence in u, since each rule in $\mathcal{R}(\mathcal{P})$ is obtained from a rule in $\mathcal{G}(\mathcal{P})$ by either removing literals or substituting literals with (conjunctions of) comparison atoms with a fixed structure, while every rule in $\mathcal{G}(\mathcal{P})$ is an instance of a rule in $\mathcal{P}$. □

Corollary 4.22 and Lemma 4.23 imply that, essentially, we can use Algorithm 1 for deciding fact entailment for semi-positive LL-programs, and, moreover, both complexity bounds of Theorem 4.17 transfer to the semi-positive case.

Theorem 4.24. *The fact entailment problem for semi-positive LL-programs is in* coNEXP *in combined and in* coNP *in data complexity.*

Proof. The claim is shown in exactly the same way as Theorem 4.17; the only difference is that we use $\mathcal{R}(\mathcal{P})$ instead of $\mathcal{G}(\mathcal{P})$ in Algorithm 1, and apply Corollary 4.22 and Lemma 4.23 instead of Corollary 4.2 and Lemma 4.3, respectively, for justification (note that, same as in $\mathcal{G}(\mathcal{P})$, all the integers in the reduct $\mathcal{R}(\mathcal{P})$ are inherited from the input program $\mathcal{P}$). □

Before proceeding to arbitrary LL-programs, we briefly discuss the problem of entailment of an LUB expression by a semi-positive LL-program, which will be useful when expressing Ross and Sagiv's monotonic programs in Section 8.1, but is also interesting in its own right. In particular, in the following proposition, we provide a DEXP combined and a DP data complexity upper bound for this problem. Later on, in Section 5 (Theorem 5.5), we will show matching lower bounds that hold already for positive programs, which imply that extending the query language of facts with negation and conjunction leads to a significant complexity jump (under usual complexity-theoretic assumptions).

Proposition 4.25. *The problem of entailment of a ground LUB expression by a semi-positive LL-program is in* DEXP *in combined and in* DP *in data complexity.*

Proof. By definition, for a semi-positive LL-program $\mathcal{P}$, a limit predicate C, a tuple of objects $\mathbf{a}$, and a ground numeric term s, we have $\mathcal{P} \models \lceil C(\mathbf{a}, s) \rceil$ if and only if $\mathcal{P} \models C(\mathbf{a}, s)$ and $\mathcal{P} \not\models C(\mathbf{a}, s+k)$, where $k = 1$ if C is max and $k = -1$ if C is min. So, the claim follows from the fact that terms s and $s + k$ can be evaluated to integers in polynomial time and Theorem 4.24, which says that fact entailment for semi-positive LL-programs is in coNEXP in combined and in coNP in data complexity. □

We conclude this section by considering arbitrary LL-programs, for which we establish Δ_2^{EXP} and Δ_2^{P} upper bounds for the fact entailment problem in combined and data complexity, respectively.

ALGORITHM 3: Fact Entailment for LL-Programs

```
Input: LL-program $\mathcal{P}$ and fact $\gamma$
Output: true if and only if $\mathcal{P} \models \gamma$
1 set $\mathcal{J}_0 := \emptyset$
2 compute the number $h$ of strata admitted by $\mathcal{P}$
3 for $i := 1$ to $h$ do
4     set $\mathcal{J}_i := \mathcal{N}(\mathcal{R}(\mathcal{P}[i] \cup \mathcal{J}_{i-1}))$
5 return true if $\gamma \sqsubseteq \mathcal{J}_h$ and false otherwise
```

Later on, in Section 5 (Theorem 5.10), we will show that these bounds are tight and hence fact entailment becomes harder when generalising semi-positive to arbitrary LL-programs. Such a complexity jump should not come as a surprise in light of the discussion preceding Proposition 4.25: entailment of LUB expressions by positive programs can be easily reduced to entailment of facts by arbitrary programs, which is already DP-hard.

Algorithm 3 decides entailment of a fact γ by an arbitrary LL-program $\mathcal{P}$. In line 4, the algorithm calls a subroutine that computes the pseudomaterialisation $\mathcal{J}_i$ of the semi-positive program $\mathcal{R}(\mathcal{P}[i] \cup \mathcal{J}_{i-1})$, which is the reduct of the current stratum $\mathcal{P}[i]$ and the pseudomaterialisation $\mathcal{J}_{i-1}$ computed for the previous stratum; the existence of such a subroutine and its computational bounds are established by Algorithm 2 and Theorem 4.20. Therefore, the algorithm constructs the pseudointerpretation $\mathcal{J}_h$ corresponding to the materialisation of $\mathcal{P}$ stratum by stratum in a bottom-up fashion. Once $\mathcal{J}_h$ has been constructed, entailment of γ is checked directly over $\mathcal{J}_h$.

The following theorem justifies the correctness of the algorithm and its complexity bounds.

Theorem 4.26. *The fact entailment problem for LL-programs is in Δ_2^{EXP} in combined and in Δ_2^{P} in data complexity.*

Proof. We first show partial correctness of Algorithm 3. Since $\mathcal{M}(\mathcal{P})$ is the partial materialisation $\mathcal{M}_h^{\omega_1}$ of $\mathcal{P}$, for h the number of strata $\mathcal{P}$ admits, and $\mathcal{P} \models \gamma$ if and only if $\gamma \in \mathcal{M}(\mathcal{P})$, it is enough to show that $\mathcal{J}_h$ corresponds to $\mathcal{M}(\mathcal{P})$ in the end of the loop in lines 3–4. To this end, we prove, by induction on i, that the partial materialisation $\mathcal{M}_i^{\omega_1}$ and the pseudointerpretation $\mathcal{J}_i$ correspond to each other for each $i \in \mathbb{N}$. First, for $i = 0$, we have $\mathcal{M}_0^{\omega_1} = \mathcal{J}_0 = \emptyset$ by definition. Let now $\mathcal{M}_{i-1}^{\omega_1}$ and $\mathcal{J}_{i-1}$ correspond to each other for some $i \geq 1$. For convenience, in the reasoning below, we slightly abuse our definitions and view possibly infinite (partial) materialisations as programs (which should be finite by definition); this is done without loss of generality because all the relevant notions lift to infinite sets of rules without any changes. First, in the program $\mathcal{P}[i] \cup \mathcal{M}_{i-1}^{\omega_1}$, the first stratum is $\mathcal{M}_{i-1}^{\omega_1}$ and the second is $\mathcal{P}[i]$, so

$$\mathcal{M}_i^{\omega_1} = \mathcal{M}(\mathcal{P}[i] \cup \mathcal{M}_{i-1}^{\omega_1}).$$

Note that Definitions 3.5 (materialisation for limit programs) and 4.5 (correspondence between limit-closed interpretations and pseudo-interpretations) imply that $\mathcal{M}_{i-1}^{\omega_1} = \mathcal{M}(\mathcal{J}_{i-1})$, and hence

$$\mathcal{M}(\mathcal{P}[i] \cup \mathcal{M}_{i-1}^{\omega_1}) = \mathcal{M}(\mathcal{P}[i] \cup \mathcal{M}(\mathcal{J}_{i-1})).$$

However, $\mathcal{J}_{i-1}$ is in the first stratum of $\mathcal{P}[i] \cup \mathcal{J}_{i-1}$, and hence

$$\mathcal{M}(\mathcal{P}[i] \cup \mathcal{M}(\mathcal{J}_{i-1})) = \mathcal{M}(\mathcal{P}[i] \cup \mathcal{J}_{i-1}).$$

Next, by Lemma 4.3, we have

$$\mathcal{M}(\mathcal{P}[i] \cup \mathcal{J}_{i-1}) = \mathcal{M}(\mathcal{R}(\mathcal{P}[i] \cup \mathcal{J}_{i-1})).$$

Finally, by Lemma 4.1,

$$\mathcal{M}(\mathcal{R}(\mathcal{P}[i] \cup \mathcal{J}_{i-1})) \quad \text{corresponds to} \quad \mathcal{N}(\mathcal{R}(\mathcal{P}[i] \cup \mathcal{J}_{i-1})) = \mathcal{J}_i.$$

So, $\mathcal{M}_i^{\omega_1}$ and $\mathcal{J}_i$ correspond to each other. In particular, $\mathcal{M}(\mathcal{P})$ corresponds to $\mathcal{J}_h$, and hence $\mathcal{J}_h = \mathcal{N}(\mathcal{P})$, as required.

We next show that Algorithm 3 works within the required complexity bounds—that is, in Δ_2^{EXP} and in Δ_2^{P} in data complexity.

To this end, first note that the stratification $\Lambda_{\mathcal{P}}$ of $\mathcal{P}$, as well as the number of strata h in this stratification in line 2, can be computed in polynomial time.

Then, for each $i \in [1, h]$, the number of distinct constants in $\mathcal{P}[i] \cup \mathcal{J}_{i-1}$ is bounded by $c + c^v \cdot d$, where c, d, and v are the number of distinct constants, the number of distinct limit predicates, and the maximal arity of a predicate in $\mathcal{P}$, respectively, and hence it is linearly bounded in $(cd)^u$, where $u = \max_{\rho \in \mathcal{P}} [\![\rho]\!]$. Therefore, by claim 1 of Lemma 4.23, there are polynomials p_1 and p_2 such that each reduct $\mathcal{R}(\mathcal{P}[i] \cup \mathcal{J}_{i-1})$ can be computed in time

$$p_1((cd)^{p_2(u)} + \|\mathcal{P}[i] \cup \mathcal{J}_{i-1}\|); \tag{38}$$

moreover, the size $\|\mathcal{R}(\mathcal{P}[i] \cup \mathcal{J}_{i-1})\|$ of the reduct is bounded by the same number.

Note also that, by claim 2 of Lemma 4.23, $\max_{\rho \in \mathcal{R}(\mathcal{P}[i] \cup \mathcal{J}_{i-1})} [\![\rho]\!]$ is linearly bounded in u for each $i \in [1, h]$. Hence, by Theorem 4.20, there are polynomials q_1 and q_2 such that each pseudomaterialisation $\mathcal{J}_i = \mathcal{N}(\mathcal{R}(\mathcal{P}[i] \cup \mathcal{J}_{i-1}))$ can be computed in time

$$q_1(2^{q_2(u)} + \|\mathcal{R}(\mathcal{P}[i] \cup \mathcal{J}_{i-1})\|) \tag{39}$$

with an access to an NP oracle; moreover, the size $\|\mathcal{J}_i\|$ of the pseudomaterialisation is bounded by the same number.

Plugging (38) into (39), we conclude that there exist polynomials p and p' such that each $\mathcal{J}_i$ can be computed in line 4 of the algorithm in time

$$p((cd)^{p'(u)} + \|\mathcal{P}\| + \|\mathcal{J}_{i-1}\|)$$

with an access to an NP oracle, and the size of $\mathcal{J}_i$ is bounded by the same number. So, overall, the pseudomaterialisation $\mathcal{N}(\mathcal{P}) = \mathcal{J}_h$ can be computed by the loop in lines 3–4 in time bounded by

$$q^h((cd)^{p'(u)} + \|\mathcal{P}\|) \tag{40}$$

with an access to an NP oracle, where $q(k) = k + p(k)$ for every $k \in \mathbb{N}$ and q^h is q iterated h times (e.g., $q^3(k)$ is $q(q(q(k)))$). The other lines of the algorithm are easy to evaluate, so bound (40) implies both of the required complexity bounds (note that for data complexity, both h and u are fixed). □

5 EXPRESSIVE POWER AND COMPLEXITY LOWER BOUNDS FOR LL-PROGRAMS

In this section, we study the expressive power of LL-programs and establish lower bounds on the complexity of fact entailment matching the upper bounds established in Section 4.

Our main results on expressive power are that the language of semi-positive LL-programs captures the complexity class coNP and the language of all LL-programs captures $\Delta_2^{\mathsf{P}} = \mathsf{P}^{\mathsf{NP}}$, both over ordered datasets (Theorems 5.3 and 5.9, respectively). We also show that the full expressive power of the languages is available already when the programs do not mention ∞, $\times$ or $-$, when all the integers are represented in unary (in fact, the only integer used is 1), and, for capturing Δ_2^{P}, when the programs are restricted to admit at most three strata. Thus, semi-positive LL-programs have the same expressive power as usual answer set programming (ASP) formalisms based on stable models [50] and as the language SO$\forall$ of second-order Boolean queries allowing only for universal quantification over second-order variables [29], but one level above usual semi-positive Datalog, which captures over ordered datasets P [14]. Furthermore, our result for the language of all LL-programs places them in-between usual ASP formalisms and their disjunctive versions, such as disjunctive Datalog, which is known to capture $\Pi_2^{\mathsf{P}} = \mathsf{coNP}^{\mathsf{NP}}$ [17]. Note that logical characterisations of Δ_2^{P} have been described in the literature [15, 18, 55]; these characterisations, however, usually rely on rather artificial logical languages consisting of a P formalism (e.g., first-order logic with fixed-point) with access to an NP operator (e.g., Hamiltonian path quantifier).

Our complexity lower bounds—that is, coNEXP- and coNP-hardness for positive and semi-positive LL-programs (Theorem 5.4) and Δ_2^{EXP}- and Δ_2^{P}-hardness for arbitrary LL-programs (Theorem 5.10)—hold under the same restrictions as the expressivity results; moreover, the second result holds even for programs that admit at most two strata.

5.1 Positive and Semi-Positive LL-Programs

We start this section with the main ideas underlying our coNP capturing result for semi-positive LL-programs. In general, we need to check, by means of a fixed semi-positive LL-program, whether a polynomial-time non-deterministic Turing machine applied to a given input yields a computation tree where all branches are rejecting. This computation tree is of polynomial depth, but may have an exponential number of leaves. *Datalog*$_{\mathbb{Z}}$ is not equipped with any non-deterministic constructs, so the (single) evaluation of a simulating LL-program should essentially traverse, in search for an accepting branch, all the branches of the tree according to some strategy. Assuming for simplicity that all branches have the same length, the most straightforward strategy is to traverse the branches in parallel, from the leaves to the root, taking the logical OR of the acceptance bits in the branching nodes. This strategy, however, requires remembering a bit for every node of depth currently examined by the traversal, and the number of such nodes can be exponential in the size of the input (e.g., for the nodes immediately preceding the leaves). This is problematic for a fixed *Datalog*$_{\mathbb{Z}}$ program; indeed, there is a fixed number of predicates of fixed arity, and thus we cannot remember an exponential amount of information using polynomially many facts in a pseudointerpretation (note that exponentially-sized integers would not help here, because, by Lemma 4.19, if such an integer appears in a derivation, it can be treated as infinity). So, we adopt an alternative strategy where branches are explored one by one in a left-first depth-first manner. In particular, the (polynomial) depth of the currently explored node is encoded by a tuple of objects, while the branch number is encoded as an integer (of polynomial size) whose binary encoding represents the sequence of guesses made by the machine along the branch from the root to the current node. Thus, for a fixed depth, the integer encodings strictly increase along the exploration, which is necessary, since the predicates we use have to be limit predicates and so only their maximal values are meaningful. Then, we ensure correct simulation as follows: if a branch is rejecting we proceed to the next branch, and if it is accepting we block further derivation. The machine returns true if and only if the last branch is unsuccessfully explored.

To make these ideas precise, we begin by providing a definition of a Turing machine deciding an NP problem. Consider a problem $P \in \mathsf{NP}$ over alphabet $\{0, 1\}$. There exists a nondeterministic Turing machine M_P and a number $d \in \mathbb{N}$—called the *degree* of M_P—with the following properties. Machine M_P has a set of states Q that contains the initial state q^{init}, the set Q^{acc} of accepting states, and the set Q^{rej} of rejecting states; it has a work tape, infinite to the right, over alphabet $\Gamma = \{0, 1, \textvisiblespace\}$ with $\textvisiblespace$ the blank symbol, and a transition function

$$\delta : (Q \setminus (Q^{\mathsf{acc}} \cup Q^{\mathsf{rej}})) \times \Gamma \times \{0, 1\} \to Q \times \Gamma \times \{\mathsf{left}, \mathsf{right}\}.$$

The nondeterminism of M_P is realised by the third argument of the transition function—that is, in each step, computation of M_P splits into two branches, one for 0 as the argument and one for 1;

on each branch, M_P enters a new state, writes a new symbol into the tape cell under the head, and moves the head left or right, which is represented by symbols left and right, respectively. Without loss of generality, it is convenient to require that the two transitions for 0 and 1 always end in different states, and that M_P never moves the head to the left of the first cell of the tape and terminates (i.e., accepts or rejects by entering a state in Q^{acc} or in Q^{rej}, respectively) on an input w in exactly $(\max(|w|, 2))^d$ steps, for $|w|$ the length of w. A partial run of M_P on an input w is a sequence of configurations of M_P (i.e., triples of a state, a position of the head, and contents of the tape) starting in the initial configuration for w (i.e., a configuration with state q^{init}, the head over the left-most cell, and w on the tape) that agrees with δ at all steps. We have $w \in P$ if and only if there is an accepting run (i.e., a partial run ending in a configuration with a state in Q^{acc}). In the rest of this section, by a machine deciding a problem $P \in \mathsf{NP}$ we mean a Turing machine M_P with a degree d as described above.

Note that, since, by assumption, two transitions differing only in the guessed bit always lead to different states, each partial run $\Pi = C_0, \ldots, C_j$ of machine M_P on an input w has a one-to-one correspondence with the sequence of guesses from $\{0, 1\}$ of length j. Since $j \leq h$ for $h = (\max(|w|, 2))^d$, this sequence is the binary representation of a number in $[0, 2^h - 1]$; thus, let $\ell(\Pi)$ denote the number obtained from this representation by prepending 1 to it (i.e., adding 1 as the most significant bit). This plays a technical role: it ensures that each number represents a unique partial run; moreover, we can model a 0 guess by doubling the number, and a 1 guess by doubling and adding 1.

The following definition reflects the main idea for the simulation of a nondeterministic Turing machine M_P by a semi-positive LL-program: to decide whether an input w is in P, it is enough to go through the terminating runs Π of M_P on w in increasing order of $\ell(\Pi)$ until finding an accepting run or completing the pass at the rejecting run Π with $\ell(\Pi)$ consisting of $|\Pi| = (\max(|w|, 2))^d + 1$ ones; then, $w \in P$ if and only if the last explored run is accepting. Here and in what follows, $|\Pi|$ is the number of configurations in a partial run Π, which equals the number of guesses plus one.

Definition 5.1. For $P \in \mathsf{NP}$, a partial run Π of M_P is *searching* if one of the following holds:

- Π consists of a single initial configuration;
- the last digit of $\ell(\Pi)$ is 0 and $\Pi = \Pi', C$ for a searching partial run Π' and configuration C;
- the last digit of $\ell(\Pi)$ is 1, the partial run Π'' with $\ell(\Pi'') = \ell(\Pi) - 1$ is searching, and all terminating extensions of Π'' are rejecting.

The following lemma establishes that it is enough to go through all searching partial runs when looking for the accepting one.

Lemma 5.2. *For $P \in \mathsf{NP}$, whenever a searching partial run Π of M_P on an input w satisfies $\ell(\Pi) \geq \ell(\Pi')$ for each searching partial run Π' of M_P on w with $|\Pi| = |\Pi'|$, then there is an accepting extension of Π if and only if $w \in P$.*

Proof. The lemma follows from Definition 5.1. Indeed, if all extensions of a searching partial run Π' are rejecting, then either there is a searching partial run Π'' with $|\Pi''| = |\Pi'|$ and $\ell(\Pi'') > \ell(\Pi')$, or $\ell(\Pi')$ is the sequence of ones of length $|\Pi'|$—that is, Π' is the last partial run of this length (with respect to the order); conversely, if Π' has an accepting extension, then all partial runs Π'' with $|\Pi''| = |\Pi'|$ and $\ell(\Pi'') > \ell(\Pi')$ are not searching. □

The following theorem on the expressive power of semi-positive LL-programs is one of our main results. Roughly speaking, it says that adding 'limit' arithmetic and negation over EDB predicates to usual Datalog provides the same power as adding unrestricted, not necessary stratified, negation under the stable model semantics (which yields usual ASP based on stable models) in the sense that both languages capture coNP over ordered datasets [50]. Importantly, we cannot expect the same result for positive LL-programs as they lack any form of negation, which is necessary for capturing any reasonable complexity class. Note also that we explicitly require homogeneity in the formulation of the theorem (as well as analogous theorems below), because it makes the statement stronger even in light of Proposition 3.1—the reduction from nonhomogeneous to homogeneous programs may introduce subtraction, which is not allowed in the theorem.

Theorem 5.3. *The language of semi-positive LL-programs captures* coNP *over ordered datasets; the result already holds for the language of homogeneous semi-positive LL-programs that do not mention* ∞, $\times$ *or* $-$, *and use only* 1 *as a numeric constant.*

Proof. As shown in Theorem 4.24, fact entailment for semi-positive LL-programs is in coNP in data complexity. Therefore, the problem of evaluating a fixed semi-positive LL-program $\mathcal{P}$ on a dataset $\mathcal{D}$ from the family of ordered datasets D (i.e, checking whether $\mathcal{P} \cup \mathcal{D} \models goal$ for a special nullary predicate *goal*) is also in coNP, which means that condition 1 for capturing a complexity class given in Section 2.2 holds for the language of semi-positive LL-programs. Thus, it remains to show that condition 2 holds for the language of semi-positive LL-programs without ∞, $\times$, $-$, or numbers other than 1. This language is easily seen to be closed under first-order reductions following exactly the same reasoning as for usual semi-positive Datalog [14, 29] (note that first-order reductions between datasets in D involve no numeric predicates or arithmetic). Hence, as argued in Section 2.2, it suffices to show that condition 2 is satisfied for signature $\Sigma_{\text{graph}} = \{edge\} \cup \Sigma_{\text{ord}}$—that is, that for each problem $P \in$ coNP there is a semi-positive LL-program $\mathcal{P}_P$ with EDB predicates from Σ_{graph} that satisfies the aforementioned restrictions and such that, for every dataset $\mathcal{D} \in D$ over Σ_{graph}, $\mathcal{P}_P \cup \mathcal{D} \models goal$ if and only if $\text{code}(\mathcal{D}) \in P$.

Consider an arbitrary problem $P' \in$ coNP. Let M_P be a nondeterministic Turing machine with a degree d deciding the complement P of P' (which is in NP) as described above, and let the encoding $\text{code}(\mathcal{D})$ of a dataset $\mathcal{D} \in D$ representing a graph with c nodes be a binary string of length c^2 as defined in Section 2.2. Since $c \geq 2$ by the definition of family D, machine M_P running on $\text{code}(\mathcal{D})$ always terminates in exactly $h = c^{2d}$ steps and uses at most h cells on its tape. Therefore, given a partial run of M_P on $\text{code}(\mathcal{D})$, we can refer to each configuration in this run by a number requiring $2d$ digits in c-ary representation—that is, by a $2d$-tuple of objects in $\mathcal{D}$ (i.e., graph nodes); similarly, we can address each cell used by M_P by a $2d$-tuple of objects. As a result, we can encode the configuration C concluding each partial run Π of M_P on $\text{code}(\mathcal{D})$ using the following facts over $(4d + 1)$-ary max predicates $head_q$, for each state q of M_P, and $tape_u$, for each symbol $u \in \Gamma$:

- $head_q(\mathbf{a}_t; \mathbf{a}_x; \ell(\Pi))$, where q is the state in C, while $\mathbf{a}_t$ and $\mathbf{a}_x$ are the $2d$-tuples of objects in $\mathcal{D}$ encoding $|\Pi| - 1$ and the head position in C, respectively; and
- $tape_u(\mathbf{a}_t; \mathbf{a}_x; \ell(\Pi))$, for each $i \in [0, h - 1]$, where u is the symbol in the i-th tape cell in C, $\mathbf{a}_t$ is as above, and $\mathbf{a}_x$ is the $2d$-tuple encoding i.

Note that we use ';' instead of ',' to separate tuples of arguments when writing these atoms. We will use this notational convention in this and some following proofs to avoid ambiguity when concatenating several such tuples without explicitly mentioning their sizes.

The idea behind the semi-positive LL-program $\mathcal{P}_P$ constructed below based on M_P is to populate predicates $head_q$ and $tape_u$ so that they encode all and only searching partial runs on an input graph; then, we can apply Lemma 5.2 and extract the answer of M_P from the encoding of the searching partial run Π with the greatest $\ell(\Pi)$.

Program $\mathcal{P}_P$ consists of rules (41)–(54) given next. First, it is convenient to have a uniform way of accessing all objects in the dataset. This is achieved by the unary object predicate *object*, defined by rules (41) from EDB predicates *first* and *next*, which encode a total order on the objects in $\mathcal{D}$.

$$\mathit{first}(z_0) \rightarrow \mathit{object}(z_0) \qquad\qquad \mathit{next}(z,z') \rightarrow \mathit{object}(z') \tag{41}$$

Rules (42), where $i \in [0, 2d-1]$, define the $4d$-ary object predicate *succ* to denote the immediate successor relation on numbers encoded as $2d$-tuples of objects, as explained above, by a lexicographic extension of the order given by *next* to $2d$-tuples of objects.

$$\begin{multline}\mathit{object}(z_1) \wedge \cdots \wedge \mathit{object}(z_i) \wedge \mathit{next}(z,z') \wedge \mathit{first}(z_0) \wedge \mathit{last}(z_{\max}) \rightarrow \\ \mathit{succ}(z_1,\dots,z_i,z,z_{\max},\dots,z_{\max};\; z_1,\dots,z_i,z',z_0,\dots,z_0) \tag{42}\end{multline}$$

It is also convenient to have access to the transitive closure succ^+ of *succ*, defined by rules (43), and the the symmetric closure *differ* of succ^+—that is, the inequality relation,—defined by rules (44).

$$\mathit{succ}(\mathbf{z};\mathbf{z}') \rightarrow \mathit{succ}^+(\mathbf{z};\mathbf{z}') \qquad \mathit{succ}^+(\mathbf{z};\mathbf{z}') \wedge \mathit{succ}(\mathbf{z}';\mathbf{z}'') \rightarrow \mathit{succ}^+(\mathbf{z};\mathbf{z}'') \tag{43}$$

$$\mathit{succ}^+(\mathbf{z};\mathbf{z}') \rightarrow \mathit{differ}(\mathbf{z};\mathbf{z}') \qquad \mathit{succ}^+(\mathbf{z};\mathbf{z}') \rightarrow \mathit{differ}(\mathbf{z}';\mathbf{z}) \tag{44}$$

The simulation of M_P starts with rules (45)–(48), which read the input graph and initialise the state, head, and tape of M_P—that is, encode the partial run Π consisting of a single initial configuration. In rules (45)–(48), $\mathbf{z}_0$ and $\mathbf{z}'_0$ are tuples consisting of variable z_0 repeated $2d$ and $2d-2$ times, respectively. In particular, the rules initialise the first $2d$ arguments of the predicates $\mathit{head}_{q^{\mathsf{init}}}$ and tape_u, for $u \in \Gamma$, by the first object in $\mathcal{D}$, which encodes $0 = |\Pi| - 1$. Similarly, rule (45) initialises each component of the second $2d$-tuple in $\mathit{head}_{q^{\mathsf{init}}}$ by the first object in $\mathcal{D}$, thus encoding that the head of M_P initially points to the left-most tape cell. Rules (46) and (47) load the graph into the tape predicates tape_1 and tape_0: the two least significant positions of the second argument tuple in facts over tape_1 encode edges in the input graph, while the same positions in facts over tape_0 encode the absence of edges; rule (48) then pads all remaining tape cells with the blank symbol. Finally, the last argument of each predicate initialised by rules (45)–(48) is set to $\ell(\Pi) = 1$, as explained earlier.

$$\mathit{first}(z_0) \rightarrow \mathit{head}_{q^{\mathsf{init}}}(\mathbf{z}_0;\mathbf{z}_0;1) \tag{45}$$

$$\mathit{first}(z_0) \wedge \mathit{edge}(x_1,x_2) \rightarrow \mathit{tape}_1(\mathbf{z}_0;\mathbf{z}'_0,x_1,x_2;1) \tag{46}$$

$$\mathit{first}(z_0) \wedge \mathtt{not}\ \mathit{edge}(x_1,x_2) \wedge \mathit{object}(x_1) \wedge \mathit{object}(x_2) \rightarrow \mathit{tape}_0(\mathbf{z}_0;\mathbf{z}'_0,x_1,x_2;1) \tag{47}$$

$$\mathit{first}(z_0) \wedge \mathit{last}(z_{\max}) \wedge \mathit{succ}^+(\mathbf{z}'_0,z_{\max},z_{\max};\mathbf{x}) \rightarrow \mathit{tape}_{\llcorner\!\lrcorner}(\mathbf{z}_0;\mathbf{x};1) \tag{48}$$

Each transition $\delta(q,u,G) = (q',u',X)$ with $X \in \{\mathsf{left}, \mathsf{right}\}$ and $G \in \{0,1\}$ is simulated by rules (49) if $G = 0$ and (50) if $G = 1$; for brevity, we again use conjunctions in rule heads, which can be easily rewritten away as argued in the proof of Theorem 2.2 (given in the appendix). Rules (49) extend the encoding of a searching partial run Π' to the encoding of the searching partial run $\Pi = \Pi', C$ in the case when the last digit of $\ell(\Pi)$ is 0. Rules (49) have an instance for each $q \in Q \setminus Q^{\mathsf{acc}} \cup Q^{\mathsf{rej}}$, each $u \in \Gamma$, and each $v \in \Gamma$; furthermore, $S(\mathbf{x};\mathbf{x}')$ denotes $\mathit{succ}(\mathbf{x}';\mathbf{x})$ if X is left and $\mathit{succ}(\mathbf{x};\mathbf{x}')$ if X is right, and hence $\mathbf{x}'$ encodes the head position after the transition provided $\mathbf{x}$ does so before the transition. To encode the step from Π' to Π, all atoms in the head of (49) have variables $\mathbf{t}'$ as their first $2d$ arguments, which are related to variables $\mathbf{t}$ in the respective body atoms by $\mathit{succ}(\mathbf{t},\mathbf{t}')$. The atoms with variables $\mathbf{y}$, which must be different from the head position $\mathbf{x}$ by $\mathit{differ}(\mathbf{x};\mathbf{y})$, propagate the unaffected contents of the tape to both sides of the head. Finally, since $G = 0$, we have $\ell(\Pi) = 2 \cdot \ell(\Pi')$, which is reflected by doubling the numeric argument of the head atoms compared to the body.

$$\begin{multline}\mathit{head}_q(\mathbf{t};\mathbf{x};m) \wedge \mathit{tape}_u(\mathbf{t};\mathbf{x};m) \wedge \mathit{tape}_v(\mathbf{t};\mathbf{y};m) \wedge \\ \mathit{succ}(\mathbf{t};\mathbf{t}') \wedge S(\mathbf{x};\mathbf{x}') \wedge \mathit{differ}(\mathbf{x};\mathbf{y}) \wedge (m + m \doteq m') \rightarrow \\ \mathit{head}_{q'}(\mathbf{t}';\mathbf{x}';m') \wedge \mathit{tape}_{u'}(\mathbf{t}';\mathbf{x};m') \wedge \mathit{tape}_v(\mathbf{t}';\mathbf{y};m') \tag{49}\end{multline}$$

Similarly, rules (50) extend the encoding of a searching partial run Π' to the encoding of the searching partial run Π when the last digit of $\ell(\Pi)$ is 1. The main difference here is that, by Definition 5.1, we need to ensure that all terminating extensions of Π' starting with a step corresponding to a 0 guess are rejecting. This is achieved by the atom $\mathit{allreject}_0(\mathbf{t}; m)$, for $\mathit{allreject}_0$ a max predicate defined by rules (51)–(53) as explained below. Moreover, since $G = 1$, we have $\ell(\Pi) = 2 \cdot \ell(\Pi') + 1$, which is reflected by increasing the numeric argument of the head atoms accordingly.

$$\begin{aligned}\mathit{head}_q(\mathbf{t}; \mathbf{x}; m) \land \mathit{tape}_u(\mathbf{t}; \mathbf{x}; m) \land \mathit{tape}_v(\mathbf{t}; \mathbf{y}; m) \land \mathit{allreject}_0(\mathbf{t}; m) \land\\ \mathit{succ}(\mathbf{t}; \mathbf{t}') \land S(\mathbf{x}; \mathbf{x}') \land \mathit{differ}(\mathbf{x}; \mathbf{y}) \land (m + m + 1 \doteq m') \to\\ \mathit{head}_{q'}(\mathbf{t}'; \mathbf{x}'; m') \land \mathit{tape}_{u'}(\mathbf{t}'; \mathbf{x}; m') \land \mathit{tape}_v(\mathbf{t}'; \mathbf{y}; m') \quad (50)\end{aligned}$$

Predicate $\mathit{allreject}_0$ is defined by rules (51)–(53) so that a fact of the form $\mathit{allreject}_0(\mathbf{a}_t; \ell(\Pi))$ is derived if and only if all terminating extensions of Π' starting with a step corresponding to a 0—that is, all terminating runs $\Pi', C_1, \dots, C_j$ with $\ell(\Pi', C_1)$ even—are rejecting; this is done with the help of a max predicate $\mathit{allreject}$ such that $\mathit{allreject}(\mathbf{a}_t; \ell(\Pi))$ is derived if and only if all terminating extensions of Π (including Π itself, if it is terminating) are rejecting. Rule (51), which instantiated for each $q^{\mathsf{rej}} \in Q^{\mathsf{rej}}$, covers the case when the last configuration is rejecting, while rules (52) and (53) propagate information one step back in the run when the last guess is 0 and 1, respectively. It is important to note that a propagation along a 1 guess is only possible after the corresponding propagation along a 0 guess—that is, for each $\mathbf{a}_t$ and each k, fact $\mathit{allreject}_0(\mathbf{a}_t; k)$ is always derived before $\mathit{allreject}(\mathbf{a}_t; k)$.

$$\mathit{head}_{q^{\mathsf{rej}}}(\mathbf{t}; \mathbf{x}; m) \to \mathit{allreject}(\mathbf{t}; m) \quad (51)$$

$$\mathit{allreject}(\mathbf{t}'; m') \land \mathit{succ}(\mathbf{t}; \mathbf{t}') \land (m + m \doteq m') \to \mathit{allreject}_0(\mathbf{t}; m) \quad (52)$$

$$\mathit{allreject}(\mathbf{t}'; m') \land \mathit{succ}(\mathbf{t}; \mathbf{t}') \land (m + m + 1 \doteq m') \to \mathit{allreject}(\mathbf{t}; m) \quad (53)$$

Finally, rule (54) checks whether all runs are rejecting—that is, whether M_P rejects the input, and hence the input is in P'. The rule triggers if and only if we can derive $\mathit{allreject}$ for the run consisting of only the initial configuration, which, by Lemma 5.2 and the explanation above, happens precisely when all runs are explored and found rejecting.

$$\mathit{first}(\mathbf{z}_0) \land \mathit{allreject}(\mathbf{z}_0; 1) \to \mathit{goal} \quad (54)$$

Having completed the construction of $\mathcal{P}_P$, we next argue its correctness. To this end, for convenience, we assume that stratification $\Lambda_{\mathcal{P}_P}$ assigns 1 to all EDB predicates, *object*, *succ*, and succ^+, and 2 to all other IDB predicates; this can be achieved by adding semantically redundant rules involving negation, which we omit here. Consider an arbitrary ordered dataset $\mathcal{D} \in D$ over Σ_{graph} and the partial materialisations $\mathcal{M}_j^\kappa$ of $\mathcal{P}_P \cup \mathcal{D}$, for numbers $j \geq 1$ and ordinals κ. Note that, by assumption, $\mathcal{M}_1^{\omega_1}$ is the closure of $\mathcal{D}$ with respect to predicates *object*, *succ*, and succ^+, and, since the program does not use ∞, we have $\mathcal{M}(\mathcal{P}_P \cup \mathcal{D}) = \mathcal{M}_2^\omega$, for ω the first infinite ordinal. Hence, we can use ordinary induction over natural numbers for proving properties of $\mathcal{P}_P$ rather than transfinite induction. Next, by construction, for every number $i \in \mathbb{N}$, every tuple of objects $\mathbf{a}_t$, and every $\ell \in \mathbb{Z}$, there are $q \in Q$ and a tuple $\mathbf{a}_x$ such that $\mathcal{M}_2^i \models \lceil \mathit{head}_q(\mathbf{a}_t; \mathbf{a}_x; \ell) \rceil$ if and only if there are $u \in \Gamma$ and a tuple $\mathbf{a}'_x$ such that $\mathcal{M}_2^i \models \lceil \mathit{tape}_u(\mathbf{a}_t; \mathbf{a}'_x; \ell) \rceil$; moreover, the latter implies that for every tuple $\mathbf{a}''_x$, there is some $u \in \Gamma$ such that $\mathcal{M}_2^i \models \lceil \mathit{tape}_u(\mathbf{a}_t; \mathbf{a}''_x; \ell) \rceil$. In other words, the maximal facts encoding a configuration are either all derived by a partial materialisation or none of them is derived; moreover, the represented configuration is the last one in the partial run Π of M_P on code($\mathcal{D}$) represented by ℓ, and $|\Pi|$ is encoded by $\mathbf{a}_t$. Since, by construction, $\ell = \ell(\Pi)$ uniquely identifies Π, we can say that Π is *derived* by $\mathcal{M}_2^i$. Next, we prove that a partial run Π is derived by $\mathcal{M}_2^i$ for a number $i \in \mathbb{N}$ if and only if Π is searching.

We prove the forward direction of this claim—that is, show that if a partial run Π is derived by $\mathcal{M}_2^i$ for a number $i \in \mathbb{N}$ then Π is searching. The proof is by induction on $i > 0$ (the case $i = 0$ is trivial as $\mathcal{M}_2^0 = \mathcal{M}_1^{\omega_1}$ does not derive any partial run). For the base case, $\mathcal{M}_2^1$ derives, by rules (45)–(48), only the run consisting of the initial configuration of M_P on $\mathrm{code}(\mathcal{D})$, which is searching by the first case of Definition 5.1. For the inductive step, we prove the claim for $i+1$ assuming that it holds for i. Consider a partial run Π derived by $\mathcal{M}_2^{i+1}$. There are two cases: either the last guess of Π is 0 and the maximal facts representing Π are obtained by application of rules (49) to $\mathcal{M}_2^i$, or the last guess is 1 and the maximal facts are obtained by application of rules (50) to $\mathcal{M}_2^i$. In the first case, the rule bodies in (49) ensure that $\mathcal{M}_2^i$ derives the partial run Π' such that $\Pi = \Pi', C$, which, by the inductive hypothesis, implies that Π' is searching, and hence, by Definition 5.1, that Π is searching. In the second case, similar reasoning applies, except that, additionally, the rule bodies in (50) ensure that $\mathcal{M}_2^i \models \mathit{allreject}_0(\mathbf{a}'_t; \ell(\Pi'))$ for the tuple of objects $\mathbf{a}'_t$ encoding $|\Pi'| = |\Pi| - 1$, which means, by rules (51)–(53), that all terminating runs of M_P extending Π' first by a 0 guess and then in an arbitrary way until termination are rejecting, and hence, by Definition 5.1, Π is searching.

We next prove the backward direction of the claim—that is, show that if a partial run Π is searching then it is derived by $\mathcal{M}_2^i$ for a number $i \in \mathbb{N}$. The proof is by induction on the definition of searching partial runs (Definition 5.1). For the base case, consider a partial run Π consisting of only the initial configuration of machine M_P on $\mathrm{code}(\mathcal{D})$. Then, Π is derived by $\mathcal{M}_2^1$ according to the initialisation rules (45)–(48). For the inductive step, consider first a partial run Π such that the last digit of $\ell(\Pi)$ is 0 and $\Pi = \Pi', C$, for Π' a searching partial run and C a configuration. Since Π' is searching, by the inductive hypothesis, it is derived by $\mathcal{M}_2^i$ for some i. Hence, by rules (49), Π is derived by $\mathcal{M}_2^{i+1}$, as required. Finally, consider a partial run Π such that the last digit of $\ell(\Pi)$ is 1, the partial run Π'' with $\ell(\Pi'') = \ell(\Pi) - 1$ is searching, and all terminating extensions of Π'' are rejecting. Since Π'' is searching, by the inductive hypothesis, it is derived by $\mathcal{M}_2^{i'}$ for some i'; moreover, since all terminating extensions of Π'' are rejecting and each such extension is derived by some $\mathcal{M}_2^{i''}$ with $i'' \geq i'$, rules (51)–(53) ensure that $\mathcal{M}_2^i \models \mathit{allreject}_0(\mathbf{a}'_t; \ell(\Pi''))$ for the tuple of objects $\mathbf{a}'_t$ encoding $|\Pi''|$ and for some i greater than all i''. Hence, by rules (50), Π is derived by $\mathcal{M}_2^{i+1}$, as required.

We conclude that a partial run Π is derived by $\mathcal{M}_2^i$ for some $i \in \mathbb{N}$ if and only if Π is searching. Then, by Lemma 5.2, all terminating runs of M_P on $\mathrm{code}(\mathcal{D})$ are rejecting if and only if each of them is searching, and hence is derived by some partial materialisation. Therefore, $\mathit{allreject}(c_0, \ldots, c_0; 1)$ is in $\mathcal{M}(\mathcal{P}_P)$, for c_0 the first object in $\mathcal{D}$—and hence rule (54) derives *goal*—if and only if all terminating runs of M_P on $\mathrm{code}(\mathcal{D})$ are rejecting—that is, if and only if M_P does not accept $\mathrm{code}(\mathcal{D})$. □

By Theorem 5.3 and Proposition 2.3, fact entailment for homogeneous semi-positive LL-programs without ∞, $\times$ or $-$, and using only 1 as a numeric constant is coNP-hard in data complexity, which matches the upper bound in Theorem 4.24. Moreover, we can adapt the proof of Theorem 5.3 to show that even for positive LL-programs with the same restrictions, fact entailment is coNEXP-hard in combined and coNP-hard in data complexity, which matches the upper bounds in Theorems 4.17 and 4.24.

Theorem 5.4. *The fact entailment problem for homogeneous positive LL-programs that do not mention ∞, $\times$ or $-$, and use only* 1 *as a numeric constant is* coNEXP*-hard in combined and* coNP*-hard in data complexity.*

Proof. We start with the data complexity bound. First, note that if a problem P' over $\{0, 1\}$ is coNP-complete, then so is the problem

$$P = \{ww_0 \mid w \in P' \text{ and } w_0 \text{ is a tuple of } (|w| + 2)^2 - |w| \text{ zeros}\}. \tag{55}$$

Moreover, negation is used in the proof of Theorem 5.3 only in rule (47), where the absence of an edge between two nodes is propagated to predicate tape_0. So, we can reduce a coNP-complete problem P as above to the fact entailment problem for fixed homogeneous positive LL-programs without ∞, $\times$ or $-$, and using only numeric constant 1 as follows. First, the fixed positive LL-program $\mathcal{P}'_P$ is the same as $\mathcal{P}_P$ in the proof of Theorem 5.3 except that it uses the positive literal $\mathit{noedge}(x_1, x_2)$, for noedge a binary object predicate, instead of the negative literal $\mathtt{not}\ \mathit{edge}(x_1, x_2)$ in rule (47). Then, each instance w of P such that $|w| = c^2$ for $c \geq 2$ is reduced to an ordered dataset $\mathcal{D}_w$ over $\Sigma_{\mathsf{graph}} \cup \{\mathit{noedge}\}$ that extends the dataset $\mathcal{D}$ over Σ_{graph} satisfying $\mathrm{code}(\mathcal{D}) = w$ by a fact $\mathit{noedge}(a_1, a_2)$ for each pair of nodes a_1, a_2 such that $\mathit{edge}(a_1, a_2) \notin \mathcal{D}$ (i.e., noedge encodes the complement of the graph given by edge). Finally, each instance w with $|w| \neq c^2$, which is not in P by definition, is reduced to $\mathcal{D}_w = \emptyset$. By construction, $\mathcal{P}'_P$ satisfies all claimed syntactic restrictions and $\mathcal{P}'_P \cup \mathcal{D}_w \models \mathit{goal}$ if and only if $w \in P$, as required.

For the coNEXP combined complexity bound, we can use the same construction; the only difference is that a machine deciding a problem in NEXP, defined in exactly the same way as a machine deciding a problem in NP, terminates on input w in exactly $(\max(|w|, 2))^{(\max(|w|,2))^d}$ steps, for a fixed degree $d \in \mathbb{N}$, and uses at most this number of tape cells, so the arities of the relevant predicates become polynomially dependent on $|w|$ (e.g., the arity of head_q and tape_u becomes $4c^{2d} + 1 = 4|w|^d + 1$) and rules (42) have $2c^{2d} = 2|w|^d$ (i.e., polynomially many) instantiations. □

In addition to the above lower bounds, by further modifying the program in the proof of Theorem 5.3, we can show that entailment of ground LUB expressions by positive LL-programs is DEXP-hard in combined and DP-hard in data complexity, which matches the upper bounds established in Proposition 4.25.

Theorem 5.5. *The problem of entailment of a ground LUB expression by a homogeneous positive LL-program, where neither the expression nor the program mentions ∞, $\times$ or $-$ and both use only 1 as a numeric constant, is* DEXP*-hard in combined and* DP*-hard in data complexity.*

Proof. As in Theorem 5.4, we start with the data complexity bound. Once again, if a problem P' is DP-complete, then so is the problem P defined as in (55). Therefore, we can reduce any such DP-complete problem to the entailment problem as stated for a ground LUB expression and a fixed homogeneous positive LL-program without ∞, $\times$ or $-$, and using only numeric constant 1 as follows. Let P_1 and P_2 be problems in NP such that $w \in P$ if and only if $w \in P_1$ and $w \notin P_2$, which exist by the definition of the class DP. Let $\mathcal{P}_{P_1}$ and $\mathcal{P}_{P_2}$ be the (fixed) LL-programs constructed as in the proof of Theorem 5.3 for P_1 and P_2, respectively, and let $\mathcal{P}'_{P_1}$ and $\mathcal{P}'_{P_2}$ be the positive programs constructed from $\mathcal{P}_{P_1}$ and $\mathcal{P}_{P_2}$, respectively, in the same way as $\mathcal{P}'_P$ is constructed from $\mathcal{P}_P$ in the proof of Theorem 5.4. Then, let $\mathcal{P}''_{P_1}$ and $\mathcal{P}''_{P_2}$ be the programs obtained from $\mathcal{P}'_{P_1}$ and $\mathcal{P}'_{P_2}$, respectively, by renaming apart all common standard predicates except those in $\Sigma_{\mathsf{graph}} \cup \{\mathit{noedge}\}$ by adding to their names subscripts 1 and 2, respectively. Finally, let $\mathcal{P}''_P$ be the program consisting of all rules in $\mathcal{P}''_{P_1} \cup \mathcal{P}''_{P_2}$ as well as the following two rules, where lub is a unary max predicate.

$$\mathit{goal}_1 \rightarrow \mathit{lub}(1 + 1) \tag{56}$$

$$\mathit{goal}_2 \rightarrow \mathit{lub}(1) \tag{57}$$

Also, each input w to P is reduced to an ordered dataset $\mathcal{D}_w$ over $\Sigma_{\mathsf{graph}} \cup \{\mathit{noedge}\}$ in exactly the same way as in the proof of Theorem 5.4. As a result, $\mathcal{P}''_P$ satisfies all the required properties and $\mathcal{P}''_P \cup \mathcal{D}_w \models \lceil \mathit{lub}(1) \rceil$ if and only if $w \in P$: indeed, if $w \in P_1$ and $w \notin P_2$, then, as shown in the proof of Theorem 5.4, $\mathcal{P}''_P \cup \mathcal{D}_w \not\models \mathit{goal}_1$ and $\mathcal{P}''_P \cup \mathcal{D}_w \models \mathit{goal}_2$, which implies, by rule (57), that $\mathcal{P}''_P \cup \mathcal{D}_w \models \lceil \mathit{lub}(1) \rceil$; otherwise, either $\mathcal{P}''_P \cup \mathcal{D}_w \models \mathit{goal}_1$ or $\mathcal{P}''_P \cup \mathcal{D}_w \not\models \mathit{goal}_2$, which implies that $\mathcal{P}''_P \cup \mathcal{D}_w \not\models \lceil \mathit{lub}(1) \rceil$.

The DEXP bound then follows in the same way as the coNEXP bound in Theorem 5.4. □

5.2 Arbitrary LL-Programs

Our encoding of a Δ_2^{P} Turing machine (a deterministic polynomial time machine with access to an NP oracle) as an LL-program generalises the idea for NP problems in Section 5.1. The main difficulty is that we now need to simulate not just one run of an NP machine but several consecutive calls to such a machine. Similarly to the coNP case, the behaviour of the main machine on an input word with the runs of the oracle calls expanded can be seen as a single tree of polynomial depth but exponential size, where each branch of an oracle call feeds back to the main machine, assuming that the answer to the call is the answer for this particular branch. This representation thus considers not only the correct run of the main machine, but also all the 'runs' that have some answers of the calls wrong, which will be convenient for our encoding of the machine as an LL-program. As in the coNP case, our encoding explores this tree in a depth-first left-first manner, proceeding to the next branch in every call only if the current branch is rejecting. This again enables consecutive exploration of an exponential number of branches by means of (carefully incremented) 'limit' integers of polynomial size.

Similarly to Section 5.1, we start the formalisation of the ideas above with a precise definition of a Turing machine deciding a problem in Δ_2^{P}, which we rely on in this section. To this end, consider a problem $P \in \Delta_2^{\mathsf{P}}$ over alphabet $\{0, 1\}$. There exists a deterministic Turing machine M_P deciding P with access to a nondeterministic Turing machine O as an oracle and with a degree $d \in \mathbb{N}$ such that the following holds. Machine M_P has a set of states Q that contains the initial state q^{init}, the set Q^{acc} of accepting states, and the set Q^{rej} of rejecting states; the work tape is infinite to the right and has alphabet $\Gamma = \{0, 1, \llcorner\!\lrcorner\}$ with $\llcorner\!\lrcorner$ the blank symbol; finally, the transition function is as follows:

$$\delta : (Q \setminus (Q^{\mathsf{acc}} \cup Q^{\mathsf{rej}})) \times \Gamma \times \{\mathsf{yes}, \mathsf{no}\} \to Q \times \Gamma \times \{\mathsf{left}, \mathsf{right}\}.$$

Machine M_P starts in state q^{init} with the head over the left-most cell of the tape and with an input word w on the tape. Transitions of M_P are defined as usual except that they depend not only on the state in Q and the symbol in Γ in the cell under the head, but also on the answer to a call to the oracle machine O with the contents of the tape as input: the third argument of δ is the oracle answer—that is, yes or no; machine M_P then enters a new state, writes a new symbol on the tape under the head, and moves the head left or right, as determined by δ. Moreover, for M'_P the *nondeterministic version* of M_P—that is, the NP Turing machine obtained from M_P by replacing each oracle call with a nondeterministic guess of yes or no—we assume that M'_P never moves the head to the left of the first tape cell and terminates (i.e., accepts or rejects in a state in Q^{acc} or Q^{rej}, respectively) in exactly $h = (\max(|w|, 2))^d$ steps on each run. Essentially, M'_P captures the behaviour of M_P even when some of oracle answers are wrong, and, as will become clear later on, we need to state the assumption for M'_P rather than just for M_P because we simulate the behaviour of M_P even in such cases. Finally, $w \in P$ if and only if M_P accepts w.

The oracle machine O follows the definition of a machine deciding a problem in NP from Section 5.1: it has a set of states Q_O containing the initial state q_O^{init}, the set Q_O^{acc} of accepting states, and the set Q_O^{rej} of rejecting states, a work tape over alphabet Γ that is infinite to the right, and a transition function

$$\delta_O : (Q_O \setminus (Q_O^{\mathsf{acc}} \cup Q_O^{\mathsf{rej}})) \times \Gamma \times \{0, 1\} \to Q_O \times \Gamma \times \{\mathsf{left}, \mathsf{right}\}.$$

The nondeterminism of O is again realised by the third argument of the transition function. We make the following assumption on the running time of the oracle: for each step of each run of M'_P on an input w, oracle O terminates on the tape contents of M'_P in this step (i.e., accepts or rejects the tape contents by entering a state in Q_O^{acc} or in Q_O^{rej}, respectively) in exactly h^d steps, for

$h = (\max(|w|, 2))^d$—note that, since M_P terminates on w in h steps, the input to O in all these cases is of size at most h (in other words, all calls to O by M_P terminate in h^d steps even if some oracle answers are wrong). The oracle answer is yes if there is an accepting run of O and no otherwise.

The definition of M_P is nonstandard—it is usually assumed that the main machine M_P calls the oracle only when it is in a special query state, and that the input to the oracle is composed on a separate write-only tape of M_P, which is flushed after each call. Our definition, however, will be more convenient for our proof, and is equivalent to the standard one in the sense that our machine is reducible to a standard oracle machine and vice versa. Therefore, in the rest of this section, by a machine deciding a problem $P \in \Delta_2^{\mathsf{P}}$ we mean a Turing machine M_P with an oracle O and degree d as described above.

To simulate several consecutive calls to an NP machine from the main machine, we will use max predicates, the values of whose numeric arguments are formed by concatenating the numbers representing all guesses made by all runs of O in a partial run of M_P. Importantly, we will continue simulation of M_P even if the answer returned by a particular run of O is wrong because a run with the correct answer has not yet been found—this is harmless as we will also continue searching for a run with the correct answer by increasing the numeric argument; the bits corresponding to a given oracle call are more significant than those corresponding to all subsequent calls, and hence derivations where the numeric argument encodes correct guesses of O will override derivations where the numeric argument encodes wrong guesses. The following notion generalises the usual notion of a run to several oracle calls by 'concatenating' their runs.

Definition 5.6. Given a machine M_P with an oracle O and degree d deciding a problem $P \in \Delta_2^{\mathsf{P}}$, a *partial* (*expanded*) *quasirun* of M_P on an input w is a sequence

$$(C_0, B_0^0), \dots, (C_0, B_0^{e_0}), (C_1, B_1^0), \dots, (C_1, B_1^{e_1}), \qquad \dots \qquad (C_j, B_j^0), \dots, (C_j, B_j^{e_j}), \tag{58}$$

where $j \in [0, h]$ for $h = (\max(|w|, 2))^d$, $e_0 = \dots = e_{j-1} = h^d$, $e_j \in [0, h^d]$, and

- each C_i, for $i \in [0, j]$, is a configuration of M_P, where C_0 is the initial configuration for input w;
- each sequence $B_i^0, \dots, B_i^{e_i}$, for $i \in [0, j-1]$, is a terminating run of O on the contents of the tape of M_P in C_i, while C_{i+1} is the result of the transition from C_i according to δ assuming the third argument of δ is Y if $B_i^{e_i}$ is accepting and N otherwise;
- sequence $B_j^0, \dots, B_j^{e_j}$ is a partial run of O on the contents of the tape of M_P in C_j.

A partial quasirun of form (58) is *terminating* if both C_j and $B_j^{e_j}$ are terminating.

Note that, by the properties of M_P and O, configuration C_i in each partial quasirun Ψ of form (58) on an input w is terminating if and only if $i = j = h = (\max(|w|, 2))^d$, while $B_j^{e_j}$ is terminating if and only if $e_j = h^d$. Note also that, by definition, a terminating quasirun includes a run of the oracle O immediately after each step of the main machine M_P, including the last one, when the main machine is in a terminating state and the result of the overall quasirun is already known; this redundant oracle call is harmless and is included for uniformity.

As in Section 5.1, each partial run $B_i^0, \dots, B_i^{e_i}$ of O with $i \in [0, j]$ in Ψ corresponds to a sequence of guesses from $\{0, 1\}$ of length e_i, which is the binary representation of a number in $[0, 2^{h^d} - 1]$. Since M_P is deterministic and transitions of O for 0 and 1 lead to different states, each partial quasirun Ψ of form (58) has a one-to-one correspondence with the tuple consisting of w, $|\Psi|$, and the concatenation of the guess sequences for all $B_i^0, \dots, B_i^{e_i}$ of total length $j \cdot h^d + e_j$ (for fixed M_P and O). This concatenation is the binary representation of a number in $[0, 2^{j \cdot h^d + e_j} - 1]$, and, same as in Section 5.1, $\ell(\Psi)$ denotes the number obtained from this representation by prepending 1 to it.

We next adapt Definition 5.1 of searching partial runs to quasiruns as follows.

Definition 5.7. For $P \in \Delta_2^{\mathsf{P}}$, a partial quasirun Ψ of M_P of form (58) is *searching* if one of the following holds:

- $j = e_j = 0$ (i.e., $\Psi = (C_0, B_0^0)$);
- $e_j > 0$, the last digit of $\ell(\Psi)$ is 0, and $(C_0, B_0^0), \dots, (C_j, B_j^{e_j - 1})$ is searching;
- $e_j > 0$, the last digit of $\ell(\Psi)$ is 1, the partial quasirun Ψ' with $\ell(\Psi') = \ell(\Psi) - 1$ is searching, and all terminating extensions of the partial run $B_j^0, \dots, B_j^{e_j}$ are rejecting;
- $j > 0$, $e_j = 0$, and $(C_0, B_0^0), \dots, (C_{j-1}, B_{j-1}^{e_{j-1}})$ is searching.

The following lemma is an analogue of Lemma 5.2 for partial quasiruns. Here and in what follows, $|\Psi|$ is the number of pairs in a quasirun Ψ.

Lemma 5.8. *For $P \in \Delta_2^{\mathsf{P}}$, if a searching partial quasirun Ψ of a machine M_P on an input w has form* (58) *and satisfies $\ell(\Psi) \geq \ell(\Psi')$ for each searching partial quasirun Ψ' of M_P on w with $|\Psi'| = |\Psi|$, then $C_0, \dots, C_j$ is the (unique) partial run of M_P on w of length j.*

Proof. The claim follows by a straightforward induction on j. For the base case, $j = 0$ and the claim holds by definition. For the inductive step, let the claim hold for $j \in \mathbb{N}$. Then it follows for $j + 1$ by Lemma 5.2, Definition 5.7, and the fact that the bits for j in $\ell(\Psi)$ are more significant than the bits for $j + 1$. □

The following theorem establishes that the language of all LL-programs captures Δ_2^{P} over ordered datasets, complementing the result for semi-positive programs in Theorem 5.3.

Theorem 5.9. *The language of LL-programs captures Δ_2^{P} over ordered datasets; the result already holds for the language of homogeneous LL-programs that admit at most three strata, do not mention ∞, $\times$ or $-$, and use only* 1 *as a numeric constant.*

Proof. Note that condition 1 for capturing a complexity class given in Section 2.2 follows from Theorem 4.26 for the language of LL-programs and class Δ_2^{P} in exactly the same way as the corresponding claim in the proof of Theorem 5.3 follows from Theorem 4.24. Hence, it remains to show that condition 2 holds for the language of LL-programs with at most three strata and without ∞, $\times$, $-$, or numbers other than 1. This language is again closed under first-order reductions, and thus, as argued in Section 2.2, it suffices to show that it satisfies condition 2 for Σ_{graph}—that is, for each problem $P \in \Delta_2^{\mathsf{P}}$, there is an LL-program $\mathcal{P}_P$ with EDB predicates from Σ_{graph} that satisfies the aforementioned restrictions and such that, for each dataset $\mathcal{D} \in D$ over Σ_{graph}, $\mathcal{P}_P \cup \mathcal{D} \models \mathit{goal}$ if and only if $\mathsf{code}(\mathcal{D}) \in P$.

Consider an arbitrary problem $P \in \Delta_2^{\mathsf{P}}$. Let M_P be a machine with an oracle O and degree d deciding P as described above, and let the encoding $\mathsf{code}(\mathcal{D})$ of a dataset $\mathcal{D} \in D$ representing a graph with c nodes be a binary string of length c^2 as defined in Section 2.2. Since $c \geq 2$ by the definition of D, the nondeterministic version M_P' of M_P running on $\mathsf{code}(\mathcal{D})$ always terminates in exactly $h = c^{2d}$ steps and uses at most h cells on its tape; moreover, oracle O running on the contents of the tape of M_P' in each step of each run on $\mathsf{code}(\mathcal{D})$ terminates in exactly h^d steps and uses at most h^d cells on its tape. Therefore, there are $h \cdot h^d = c^{2(d+d^2)}$ possible configurations in a partial quasirun of M_P on $\mathsf{code}(\mathcal{D})$, and we can refer to each such pair by a number requiring $2(d + d^2)$ digits in c-ary representation—that is, by a $2(d + d^2)$-tuple of objects; similarly, we can refer to each used cell of M_P and O by $2(d + d^2)$-tuples of objects (only $2d$ and $2d^2$ such objects are required, respectively, but it is convenient to have all tuples be of the same size). Thus, we can encode the pair of configurations $(C_j, B_j^{e_j})$ concluding each partial quasirun Ψ, of form (58), of machine M_P on input $\mathsf{code}(\mathcal{D})$ using the following facts over $(4(d + d^2) + 1)$-ary max predicates

$head_q$, for each state q of M_P, $tape_u$, for each symbol $u \in \Gamma$, $ohead_{q_O}$, for each state q_O of O, and $otape_u$, for each symbol $u \in \Gamma$:

- $head_q(\mathbf{a}_t; \mathbf{a}_x; \ell(\Psi))$, where q is the state in C_j, while $\mathbf{a}_t$ and $\mathbf{a}_x$ are the $2(d+d^2)$-tuples of objects in $\mathcal{D}$ encoding $|\Psi| - 1 = j \cdot (h^d + 1) + e_j$ and the head position in C_j, respectively;
- $tape_u(\mathbf{a}_t; \mathbf{a}_x; \ell(\Psi))$, for each $i \in [0, h-1]$, where u is the symbol in the i-th tape cell in C_j, $\mathbf{a}_t$ is as above, and $\mathbf{a}_x$ is the $2(d+d^2)$-tuple encoding i;
- $ohead_{q_O}(\mathbf{a}_t; \mathbf{a}_x; \ell(\Psi))$, where q_O is the state in $B_j^{e_j}$, $\mathbf{a}_t$ is as above, and $\mathbf{a}_x$ is the $2(d+d^2)$-tuple encoding the head position in $B_j^{e_j}$; and
- $otape_u(\mathbf{a}_t; \mathbf{a}_x; \ell(\Psi))$, for each $i \in [0, h^d - 1]$, where u is the symbol in the i-th tape cell in $B_j^{e_j}$, $\mathbf{a}_t$ is as above, and $\mathbf{a}_x$ is the $2(d+d^2)$-tuple of objects encoding i.

The idea behind the LL-program $\mathcal{P}_P$ constructed below based on M_P is to populate predicates $head_q$, $tape_u$, $ohead_{q_O}$, and $otape_u$ so that they encode all searching partial quasiruns, where, for each rejecting quasirun Ψ with nonmaximal $\ell(\Psi)$, there is a searching quasirun Ψ' with $\ell(\Psi') > \ell(\Psi)$; then, we can apply Lemma 5.8 to extract the result of the terminating run of M_P from the encoding of the quasirun Ψ with the greatest $\ell(\Psi)$.

First, program $\mathcal{P}_P$ includes rules (41) from the proof of Theorem 5.3, populating predicate *object*, as well as rules populating $4(d+d^2)$-ary object predicates *succ*, $succ^+$, and *differ* that are the same as rules (42)–(44) except for the predicate arities.

Program $\mathcal{P}_P$ also includes rules (59)–(72) given next. The simulation of M_P starts with rules (59)–(62), which read the input graph and initialise the states, heads, and tapes of M_P and O (for the first call to O), thus encoding the partial quasirun $\Psi = (C_0, B_0^0)$. These rules are very similar to rules (45)–(48) in the proof of Theorem 5.3—the only difference is that the sizes of tuples $\mathbf{z}_0$ and $\mathbf{z}_0'$ (which repeat variable z_0) are $2(d+d^2)$ and $2(d+d^2)-2$, respectively.

$$first(z_0) \rightarrow head_{q^{\mathsf{init}}}(\mathbf{z}_0; \mathbf{z}_0; 1) \wedge ohead_{q_O^{init}}(\mathbf{z}_0; \mathbf{z}_0; 1) \tag{59}$$

$$first(z_0) \wedge edge(x_1, x_2) \rightarrow tape_1(\mathbf{z}_0; \mathbf{z}_0', x_1, x_2; 1) \wedge otape_1(\mathbf{z}_0; \mathbf{z}_0', x_1, x_2; 1) \tag{60}$$

$$\begin{aligned} &first(z_0) \wedge \texttt{not}\ edge(x_1, x_2) \wedge \\ &\quad object(x_1) \wedge object(x_2) \rightarrow tape_0(\mathbf{z}_0; \mathbf{z}_0', x_1, x_2; 1) \wedge otape_0(\mathbf{z}_0; \mathbf{z}_0', x_1, x_2; 1) \end{aligned} \tag{61}$$

$$\begin{aligned} &first(z_0) \wedge last(z_{\max}) \wedge \\ &\quad succ^+(\mathbf{z}_0', z_{\max}, z_{\max}; \mathbf{x}) \rightarrow tape_{\llcorner\!\lrcorner}(\mathbf{z}_0; \mathbf{x}; 1) \wedge otape_{\llcorner\!\lrcorner}(\mathbf{z}_0; \mathbf{x}; 1) \end{aligned} \tag{62}$$

Each transition $\delta_O(q_O, u_O, G) = (q_O', u_O', X)$ of O is simulated by rules (63) if $G = 0$ and (64) if $G = 1$. More formally, rules (63), where S is defined as in (49), extend the encoding of each searching partial quasirun $\Psi' = (C_0, B_0^0), \ldots, (C_j, B_j^{e_j - 1})$ to the encoding of the searching partial quasirun $\Psi = \Psi', (C_j, B_j^{e_j})$ provided the last digit of $\ell(\Psi)$ is 0. These rules are analogous to rules (49) but simulate a step of O rather than M_P, while the configuration C_j of M_P remains unchanged.

$$\begin{aligned} &head_q(\mathbf{t}; \mathbf{x}; m) \wedge tape_v(\mathbf{t}; \mathbf{y}; m) \wedge \\ &\quad ohead_{q_O}(\mathbf{t}; \mathbf{x}_O; m) \wedge otape_{u_O}(\mathbf{t}; \mathbf{x}_O; m) \wedge otape_{v_O}(\mathbf{t}; \mathbf{y}_O; m) \wedge \\ &\qquad succ(\mathbf{t}; \mathbf{t}') \wedge S(\mathbf{x}_O; \mathbf{x}_O') \wedge differ(\mathbf{x}_O; \mathbf{y}_O) \wedge (m + m \doteq m') \rightarrow \\ &head_q(\mathbf{t}'; \mathbf{x}; m') \wedge tape_v(\mathbf{t}'; \mathbf{y}; m') \wedge \\ &\quad ohead_{q_O'}(\mathbf{t}'; \mathbf{x}_O'; m') \wedge otape_{u_O'}(\mathbf{t}'; \mathbf{x}_O; m') \wedge otape_{v_O}(\mathbf{t}'; \mathbf{y}_O; m') \end{aligned} \tag{63}$$

Similarly, rules (64) extend the encoding of each searching partial quasirun Ψ' to the encoding of the searching partial quasirun Ψ as above provided the last digit of $\ell(\Psi)$ is 1. These rules are related to (50) in the same way as (63) are to (49). The atom over a max predicate $oallreject_0$, populated by rules (65)–(67) as explained below, ensures that all terminating extensions of the partial run $B_j^0, \dots, B_j^{e_j-1}$ of O in Ψ' starting with a 0 guess are rejecting.

$$\begin{aligned} &head_q(\mathbf{t}; \mathbf{x}; m) \wedge tape_v(\mathbf{t}; \mathbf{y}; m) \wedge \\ &\qquad ohead_{q_O}(\mathbf{t}; \mathbf{x}_O; m) \wedge otape_{u_O}(\mathbf{t}; \mathbf{x}_O; m) \wedge otape_{v_O}(\mathbf{t}; \mathbf{y}_O; m) \wedge oallreject_0(\mathbf{t}; m) \wedge \\ &\qquad\qquad succ(\mathbf{t}; \mathbf{t}') \wedge S(\mathbf{x}_O; \mathbf{x}'_O) \wedge differ(\mathbf{x}_O; \mathbf{y}_O) \wedge (m + m + 1 \doteq m') \rightarrow \\ &\quad head_q(\mathbf{t}'; \mathbf{x}; m') \wedge tape_v(\mathbf{t}'; \mathbf{y}; m') \wedge \\ &\qquad\qquad ohead_{q'_O}(\mathbf{t}'; \mathbf{x}'_O; m') \wedge otape_{u'_O}(\mathbf{t}'; \mathbf{x}_O; m') \wedge otape_{v_O}(\mathbf{t}'; \mathbf{y}_O; m') \end{aligned} \tag{64}$$

Rules (65)–(67), where (65) is instantiated for each $q_O^{\mathsf{rej}} \in Q_O^{\mathsf{rej}}$, populate predicate $oallreject_0$ in exactly the same way as rules (51)–(53), so that a fact of the form $oallreject_0(\mathrm{a_t}; \ell(\Psi))$ is derived if and only if all terminating runs of O extending the last such run in Ψ first by a 0 guess and then by an arbitrary sequence of guesses are rejecting; this is achieved, as in (51)–(53), with the help of a max predicate *oallreject* such that $oallreject(\mathrm{a_t}; \ell(\Psi))$ is derived if and only if all terminating extensions of the last run of O in Ψ (including the run itself, if it is terminating) are rejecting. Note that such a propagation along a 1 guess is only possible after the corresponding propagation along a 0 guess. Also, as we will see in the following, backward propagation beyond the initial configuration of the oracle call is harmless since the numeric argument becomes insignificantly small.

$$ohead_{q_O^{\mathsf{rej}}}(\mathbf{t}; \mathbf{x}_O; m) \rightarrow oallreject(\mathbf{t}; m) \tag{65}$$

$$oallreject(\mathbf{t}'; m') \wedge succ(\mathbf{t}; \mathbf{t}') \wedge (m + m \doteq m') \rightarrow oallreject_0(\mathbf{t}; m) \tag{66}$$

$$oallreject(\mathbf{t}'; m') \wedge succ(\mathbf{t}; \mathbf{t}') \wedge (m + m + 1 \doteq m') \rightarrow oallreject(\mathbf{t}; m) \tag{67}$$

Next, rules (68) (which have no analogue in the proof of Theorem 5.3) simulate each transition $\delta(q, u, A) = (q', u', X)$ of the main machine M_P with $A \in \{\text{yes}, \text{no}\}$ and $X \in \{\mathsf{left}, \mathsf{right}\}$ once a run of O terminates with a result, where, as before, $S(\mathbf{x}; \mathbf{x}')$ denotes $succ(\mathbf{x}'; \mathbf{x})$ if X is left and $succ(\mathbf{x}; \mathbf{x}')$ if X is right, and q_O^{ans} is instantiated by each state in Q_O^{acc} if A is yes and by each state in Q_O^{rej} if A is no. Note that the rules also reinitialise the oracle by setting the state to q_O^{init}, the head to the first cell, and the contents of the tape to a copy of the contents of the tape of M_P (where $\mathbf{z}_0$ is again a $2(d + d^2)$-fold repetition of z_0). Finally, this transition is deterministic, and hence the numeric arguments are the same in the body and the head (note that $\ell(\Psi) = \ell(\Psi')$ for a partial quasirun Ψ of the form $\Psi', (C_j, B_j^0)$). The latter guarantees that the backward propagation of $oallreject_0$ and *oallreject* beyond the initial configuration of an oracle call is harmless as the numeric arguments of the facts derived in this way are smaller than those of any fact encoding a partial quasirun with strictly fewer calls to O.

$$\begin{aligned} &head_q(\mathbf{t}; \mathbf{x}; m) \wedge tape_u(\mathbf{t}; \mathbf{x}; m) \wedge tape_v(\mathbf{t}; \mathbf{y}; m) \wedge ohead_{q_O^{ans}}(\mathbf{t}; \mathbf{x}_O; m) \wedge \\ &\qquad\qquad succ(\mathbf{t}; \mathbf{t}') \wedge S(\mathbf{x}; \mathbf{x}') \wedge differ(\mathbf{x}; \mathbf{y}) \wedge first(z_0) \rightarrow \\ &\qquad head_{q'}(\mathbf{t}'; \mathbf{x}'; m) \wedge tape_{u'}(\mathbf{t}'; \mathbf{x}; m) \wedge tape_v(\mathbf{t}'; \mathbf{y}; m) \wedge \\ &\qquad ohead_{q_O^{\mathsf{init}}}(\mathbf{t}'; \mathbf{z}_0; m) \wedge otape_{u'}(\mathbf{t}'; \mathbf{x}; m) \wedge otape_v(\mathbf{t}'; \mathbf{y}; m) \end{aligned} \tag{68}$$

Note that the transition is simulated not only for the real answer of the oracle call, but for all terminating runs of O whose guess sequence encodes a number that does not exceed the least number corresponding to an accepting run (or for all terminating runs if no accepting run exists); in other words, rules (68) extend the encoding of each searching partial quasirun $\Psi' = (C_0, B_0^0), \dots, (C_{j-1}, B_{j-1}^{e_{j-1}})$ where $B_{j-1}^{e_{j-1}}$ is a final configuration of O to a searching partial quasirun $\Psi = \Psi', (C_j, B_j^0)$. As a result, the oracle run with the greatest guess sequence among all runs simulated by $\mathcal{P}_P$ accepts if and only if the answer to the call is yes, or, in other words, a partial quasirun is simulated if and only if it is searching. By Lemma 5.8, the latter implies that we can extract the real run from the partial quasirun with the greatest guess sequence. This is realised by rules (69)–(72) over unary max predicates *accept* and *reject*, which extract and compare $\ell(\Psi_{\text{accept}})$ and $\ell(\Psi_{\text{reject}})$, for Ψ_{accept} the accepting and Ψ_{reject} the rejecting quasiruns with the maximal guess sequences over all searching quasiruns. The nullary predicate *goal* is derived by rule (72) if and only if $\ell(\Psi_{\text{reject}}) < \ell(\Psi_{\text{accept}})$—that is, by Lemma 5.8, if the run of M_P is accepting.

$$\rightarrow \mathit{reject}(0) \tag{69}$$

$$\mathit{head}_{q^{\text{rej}}}(\mathbf{t};\mathbf{x};m) \rightarrow \mathit{reject}(m) \tag{70}$$

$$\mathit{head}_{q^{\text{acc}}}(\mathbf{t};\mathbf{x};n) \rightarrow \mathit{accept}(n) \tag{71}$$

$$\lceil \mathit{reject}(m) \rceil \wedge \lceil \mathit{accept}(n) \rceil \wedge (m < n) \rightarrow \mathit{goal} \tag{72}$$

Next, we argue the correctness of the construction. Consider an arbitrary ordered dataset $\mathcal{D} \in D$ over Σ_{graph} and the partial materialisations $\mathcal{M}_j^\kappa$ of $\mathcal{P}_P \cup \mathcal{D}$, for numbers $j \geq 1$ and ordinals κ. As in Theorem 5.3, without loss of generality, we assume that stratification $\Lambda_{\mathcal{P}_P}$ maps *object*, *succ*, and *succ*$^+$ to 1, *goal* to 3, and all other IDB predicates to 2. Furthermore, we have $\mathcal{M}(\mathcal{P}_P[1] \cup \mathcal{P}_P[2] \cup \mathcal{D}) = \mathcal{M}_2^\omega$, and hence we can use ordinary induction over natural numbers for proving properties of $\mathcal{P}_P$. Next, by construction, for every $i \in \mathbb{N}$, every tuple of objects $\mathbf{a}_{\text{t}}$, and every $\ell \in \mathbb{Z}$, each of the following claims are equivalent:

- there are $q \in Q$ and a tuple $\mathbf{a}_{\text{x}}$ such that $\mathcal{M}_2^i \models \lceil \mathit{head}_q(\mathbf{a}_{\text{t}};\mathbf{a}_{\text{x}};\ell) \rceil$,
- there are $u \in \Gamma$ and a tuple $\mathbf{a}_{\text{x}}$ such that $\mathcal{M}_2^i \models \lceil \mathit{tape}_u(\mathbf{a}_{\text{t}};\mathbf{a}_{\text{x}};\ell) \rceil$,
- there are $q_O \in Q_O$ and a tuple $\mathbf{a}_{\text{x}}$ such that $\mathcal{M}_2^i \models \lceil \mathit{ohead}_{q_O}(\mathbf{a}_{\text{t}};\mathbf{a}_{\text{x}};\ell) \rceil$,
- there are $u \in \Gamma$ and a tuple $\mathbf{a}_{\text{x}}$ such that $\mathcal{M}_2^i \models \lceil \mathit{otape}_u(\mathbf{a}_{\text{t}};\mathbf{a}_{\text{x}};\ell) \rceil$;

moreover, the second claim implies that for each $\mathbf{a}_{\text{x}}$ there is $u \in \Gamma$ such that $\mathcal{M}_2^i \models \lceil \mathit{tape}_u(\mathbf{a}_{\text{t}};\mathbf{a}_{\text{x}};\ell) \rceil$, while the last claim implies that for each $\mathbf{a}_{\text{x}}$ there is $u \in \Gamma$ such that $\mathcal{M}_2^i \models \lceil \mathit{otape}_u(\mathbf{a}_{\text{t}};\mathbf{a}_{\text{x}};\ell) \rceil$. In other words, the maximal facts encoding a pair of configurations of M_P and O are either all derived by a partial materialisation or none of them is derived; moreover, this pair is the last one in the partial quasirun Ψ of M_P on code($\mathcal{D}$) represented by ℓ, and $|\Psi|$ is encoded by $\mathbf{a}_{\text{t}}$. Since $\ell(\Psi)$ uniquely identifies Ψ, we say that Ψ is *derived* by $\mathcal{M}_2^i$. Next, we prove that a partial quasirun Ψ is derived by $\mathcal{M}_2^i$ for a number $i \in \mathbb{N}$ if and only if Ψ is searching.

We prove the forward direction by induction on $i > 0$ (the case $i = 0$ is trivial as $\mathcal{M}_2^0 = \mathcal{M}_1^{\omega_1}$ does not derive any partial run). For the base case, we have that $\mathcal{M}_2^1$ derives, by rules (59)–(62), only the quasirun (C_0, B_0^0), for C_0 and B_0^0 the initial configurations of M_P and O, respectively, on code($\mathcal{D}$); this quasirun is searching by Definition 5.7. For the inductive step, we show the claim for $i + 1$ assuming that it holds for i. Consider a partial quasirun Ψ derived by $\mathcal{M}_2^{i+1}$. There are the following three cases, depending on the last transition in Ψ:

- it is an oracle transition guessing 0 and the maximal facts representing Ψ are obtained by application of rules (63) to $\mathcal{M}_2^i$,
- it is an oracle transition guessing 1 and the maximal facts are obtained by rules (64), or
- it is a transition of M_P and the maximal facts are obtained by rules (68).

The first two cases are essentially the same as the respective cases in Theorem 5.3 except that they use Definition 5.7 instead of Definition 5.1; note that, in the second case, the backward propagation of *oallreject*$_0$ and *oallreject* beyond the initial configuration of an oracle call decreases the numeric argument, and hence does not affect the maximal values of numeric arguments. In the third case, the bodies of rules (68) ensure that $\mathcal{M}_2^i$ derives the partial quasirun Ψ' such that $\Psi = \Psi', (C, B)$, which, by the inductive hypothesis, implies that Ψ' is searching, and hence, by the fourth case of Definition 5.7, that Ψ is searching.

We next prove the backward direction of the claim by induction on the definition of searching partial quasiruns (Definition 5.7). For the base case, consider a partial quasirun Ψ consisting of only the pair of the initial configurations of M_P and O on $\mathrm{code}(\mathcal{D})$. Then, Ψ is derived by $\mathcal{M}_2^1$ according to the initialisation rules (59)–(62). For the inductive step, consider first a partial quasirun Ψ such that the last transition of Ψ is an oracle transition, the last digit of $\ell(\Psi)$ is 0, and $\Psi = \Psi', (C_j, B_j^{e_j})$ for Ψ' a searching partial quasirun. Since Ψ' is searching, by the inductive hypothesis, it is derived by $\mathcal{M}_2^i$ for some i. Hence, by rules (63), Ψ is derived by $\mathcal{M}_2^{i+1}$, as required. Next, consider a partial quasirun Ψ such that the last transition of Ψ is an oracle transition, the last digit of $\ell(\Psi)$ is 1, the partial quasirun Ψ' with $\ell(\Psi') = \ell(\Psi) - 1$ is searching, and all terminating extensions of its last partial run of oracle O are rejecting. Since Ψ' is searching, by the inductive hypothesis, it is derived by $\mathcal{M}_2^{i'}$ for some i'; moreover, since all terminating extensions of the last partial run of O in Ψ' are rejecting and each such extension is derived by some $\mathcal{M}_2^{i''}$ with $i'' \geq i'$, rules (65)–(67) ensure that $\mathcal{M}_2^i \models \mathit{allreject}_0(\mathbf{b}_\mathrm{t}; \ell(\Psi'))$ for the tuple $\mathbf{b}_\mathrm{t}$ of objects encoding $|\Psi'|$ and for some i greater than all i''. Hence, by rules (64), Ψ is derived by $\mathcal{M}_2^{i+1}$, as required. Finally, consider a partial quasirun Ψ such that the last transition of Ψ is a transition of M_P, and the partial quasirun Ψ' with $\Psi = \Psi', (C_j, B_j^0)$ is searching. Since Ψ' is searching, by the inductive hypothesis, it is derived by $\mathcal{M}_2^i$ for some i and hence, by rules (68), Ψ is derived by $\mathcal{M}_2^{i+1}$, as required.

We conclude that a partial quasirun Ψ is derived by $\mathcal{M}_2^i$ for some $i \in \mathbb{N}$ if and only if Ψ is searching. However, by Lemma 5.8, the searching quasirun Ψ with maximal $\ell(\Psi)$ contains the run of M_P on $\mathrm{code}(\mathcal{D})$, and hence rules (69)–(72) derive *goal* if and only if the run is accepting. □

By Theorem 5.9 and Proposition 2.3, fact entailment for homogeneous LL-programs with at most three strata, without ∞, $\times$ or $-$, and using only 1 as a numeric constant is Δ_2^{P}-hard in data complexity, which matches the upper bound in Theorem 4.26. A tight Δ_2^{EXP} lower bound in combined complexity could then, in principle, be obtained by *complexity upgrade* techniques developed by Gottlob et al. [21]. Instead, however, we can simply adapt the proof of Theorem 5.9 in exactly the same way as the proof of Theorem 5.4 adapts the proof of Theorem 5.3, showing Δ_2^{EXP}- and Δ_2^{P}-hardness even for the language allowing for only two strata.

Theorem 5.10. *The fact entailment problem for homogeneous LL-programs that admit at most two strata, do not mention* ∞, $\times$ *or* $-$, *and use only* 1 *as a numeric constant is* Δ_2^{EXP}*-hard in combined and* Δ_2^{P}*-hard in data complexity.*

6 TRACTABILITY OF FACT ENTAILMENT

Tractability of reasoning in data complexity is important for problems involving large datasets. Therefore, in this section, we introduce a *stability* condition on LL-programs, which brings the complexity of fact entailment down to EXP in combined and to P in data complexity, thus matching the complexity bounds for usual positive Datalog. We then introduce an efficiently checkable syntactic sufficient condition for stability, which we call *type-consistency*.

6.1 Cyclic Dependencies in Limit-Linear Programs

The standard fact entailment algorithm for usual positive Datalog relies essentially on direct materialisation of the fixpoint, which can be done in polynomial time in the size of data. As illustrated by Example 2.1, however, a naive computation of the pseudomaterialisation of an LL-program $\mathcal{P}$ may not terminate since repeated application of $\mathcal{T}_\mathcal{P}$ can produce larger and larger numbers. Thus, we need a way to identify when the numeric argument of a limit atom *diverges*—that is, increases or decreases without a bound; moreover, to obtain a procedure that is tractable in data complexity, divergence should be detected after polynomially many steps. Example 6.1 illustrates that this can be achieved by analysing cyclic dependencies.

Example 6.1. Consider an LL-program $\mathcal{P}_{\mathrm{st}}$ with the following rules over max predicates C_1 and C_2.

$$\to C_1(0) \qquad\qquad C_1(m) \to C_2(m) \qquad\qquad C_2(m) \to C_1(m+1)$$

The second rule copies the value of C_1 into C_2, and the third rule increases the value of C_1; thus, both C_1 and C_2 diverge when computing $\mathcal{N}(\mathcal{P}_{\mathrm{st}})$. The existence of a cyclic dependency between C_1 and C_2, however, does not necessarily lead to divergence. Let $\mathcal{P}'_{\mathrm{st}}$ be obtained from $\mathcal{P}_{\mathrm{st}}$ by adding a fact $C(5)$, for C a max predicate, and replacing the second rule with the following one.

$$C_1(m) \wedge C(m) \to C_2(m)$$

While a cyclic dependency between C_1 and C_2 still exists, the increase in the values of C_1 and C_2 is bounded by the value of C, which is independent of the values of C_1 and C_2; thus, neither C_1 nor C_2 diverges.

In the rest of this section, we formalise the notion of dependency and establish some basic properties needed later on. For now, we focus on OG-ground (and hence positive) programs.

Definition 6.2. A numeric variable m *depends* on a numeric variable n in an OG-ground rule ρ if either m is n or m occurs in an atom in ρ with a variable that depends on n. A numeric term s_2 *depends* on a numeric term s_1 if s_2 mentions a variable that depends on a variable mentioned in s_1.

In other words, the dependency relation on the variables of a rule is reflexive, symmetric, and transitive; moreover, it is the transitive closure of the relation saying that two variables occur in the same atom.

The following proposition establishes a first immediate property of dependent terms (recall that $\delta(\rho, \mathcal{J})$ is the pseudofact resulting from the application of an OG-ground rule ρ to a pseudointerpretation $\mathcal{J}$, see Definition 4.7).

Proposition 6.3. *Let $\rho = \varphi \to C(\mathbf{a}, s)$ be an OG-ground rule with C a limit predicate and s a term not depending on the numeric term of any limit atom in φ. Then, for each two pseudointerpretations $\mathcal{J}$ and $\mathcal{J}'$ such that ρ is applicable to both, the pseudofacts $\delta(\rho, \mathcal{J})$ and $\delta(\rho, \mathcal{J}')$ coincide.*

The following definition formalises the intuitions behind Example 6.1, providing the basis for the notion of stability. It introduces graphs describing how integers propagate when evaluating the immediate consequence operator. As we will see, certain cycles in such a graph guarantee divergence of all participating numeric arguments. In this definition and the rest of the section, we extend the set $\mathbb{Z} \cup \{\infty\}$ with a new symbol $\bot$, intuitively representing that a fact holds for no integer. This symbol, of course, cannot appear in *Datalog*$_{\mathbb{Z}}$ programs or during their evaluation, but is merely introduced for convenience of presentation. To avoid bulky case analysis, we also assume that $\bot < k < \infty$ for all $k \in \mathbb{Z}$, and $\bot + \ell = \bot$ and $\infty + \ell = \infty$ for all $\ell \in \mathbb{Z} \cup \{\infty\}$ (in particular, $\bot + \infty = \bot$—that is, $\bot$ takes priority over ∞).

Definition 6.4. The *value propagation graph* of an OG-ground program $\mathcal{P}$ over a pseudointerpretation $\mathcal{J}$ is the weighted directed graph (V, E, Ω) defined as follows:

- the set of nodes V contains $\langle C\mathbf{a} \rangle$ for each limit atom $C(\mathbf{a}, s)$ in the head of a rule in $\mathcal{P}$;
- the set of edges E contains $(\langle C_1\mathbf{a}_1 \rangle, \langle C_2\mathbf{a}_2 \rangle)$ if there is a rule in $\mathcal{P}$ applicable to $\mathcal{J}$ *producing* the edge—that is, having an atom $C_1(\mathbf{a}_1, s_1)$ in the body and an atom $C_2(\mathbf{a}_2, s_2)$ in the head such that s_2 depends on s_1;

– the weight $\Omega(e)$ of each edge $e = (\langle C_1\mathbf{a}_1\rangle, \langle C_2\mathbf{a}_2\rangle)$ in E is an element from $\mathbb{Z} \cup \{\bot, \infty\}$ defined as

$$\Omega(e) = \max\{\Omega_\rho(e) \mid \rho \in \mathcal{P} \text{ produces } e\},$$

where, for $\ell \in \mathbb{Z} \cup \{\infty\}$ such that $C_1(\mathbf{a}_1, \ell) \in \mathcal{J}$ and for each $\rho = \varphi \to C_2(\mathbf{a}_2, s)$ in $\mathcal{P}$ producing e,

$$\Omega_\rho(e) = \begin{cases} \infty & \text{if the IOP } (\psi(\rho, \mathcal{J}), \max_{C_2} s) \text{ is unbounded,} \\ \bot & \text{if } (\psi(\rho, \mathcal{J}), \max_{C_2} s) \text{ is bounded and } \ell = \infty, \\ d_2 \cdot k - d_1 \cdot \ell & \text{if } (\psi(\rho, \mathcal{J}), \max_{C_2} s) \text{ has optimal value } k \text{ and } \ell \in \mathbb{Z} \\ & \quad \text{where, for each } i \in \{1,2\},\ d_i \text{ is 1 if } C_i \text{ is max and } -1 \text{ if } C_i \text{ is min.} \end{cases}$$

The *weight* $\Omega(\Pi)$ of a path Π in a value propagation graph is the sum of the weights of all the edges in Π; path Π *has positive weight* if $\Omega(\Pi)$ is either a positive integer or ∞.

Intuitively, the value propagation graph (V, E, Ω) of an OG-ground program $\mathcal{P}$ over a pseudointerpretation $\mathcal{J}$ describes how, for each limit predicate C_1 and objects $\mathbf{a}_1$ with a pseudofact $C_1(\mathbf{a}_1, \ell_1)$ in $\mathcal{J}$, operator $\mathcal{T}_\mathcal{P}$ propagates ℓ_1 to other pseudofacts. In particular, an edge $e = (\langle C_1\mathbf{a}_1\rangle, \langle C_2\mathbf{a}_2\rangle)$ in E indicates that at least one rule $\rho \in \mathcal{P}$ is applicable to $\mathcal{J}$ with atom $C_1(\mathbf{a}_1, s_1)$ in the body and $C_2(\mathbf{a}_2, s_2)$ in the head such that s_2 depends on s_1; moreover, applying $\mathcal{T}_\mathcal{P}$ to $\mathcal{J}$ yields a pseudofact $C_2(\mathbf{a}_2, \ell_2)$ with $\ell_1 + \Omega(e) \leq \ell_2$ if both C_1 and C_2 are max predicates, and analogously for C_1 or C_2 a min predicate. In other words, edge e indicates that the application of $\mathcal{T}_\mathcal{P}$ to $\mathcal{J}$ propagates the value of $\langle C_1\mathbf{a}_1\rangle$ to $\langle C_2\mathbf{a}_2\rangle$ while increasing it by at least $\Omega(e)$. The next lemma extends this property from edges to paths, which formalises the intuition that a positive-weight cycle indicates a possibly unbounded increase or decrease of the integers along the cycle.

Lemma 6.5. *For every path $\Pi = \langle C_1\mathbf{a}_1\rangle, \dots, \langle C_j\mathbf{a}_j\rangle$ in the value propagation graph (V, E, Ω) of an OG-ground program $\mathcal{P}$ over a pseudomodel $\mathcal{J}$ of $\mathcal{P}$ such that $\Omega(\Pi) \neq \bot$, the following holds:*

– *if $\Omega(\Pi) = \infty$ or $C_1(\mathbf{a}_1, \infty) \in \mathcal{J}$, then $C_j(\mathbf{a}_j, \infty) \in \mathcal{J}$;*
– *otherwise, either $\ell_j = \infty$ or $d_j \cdot (d_1 \cdot \ell_1 + \Omega(\Pi)) \leq_{C_j} \ell_j$, where, for each $i \in \{1, j\}$, ℓ_i is the integer such that $C_i(\mathbf{a}_i, \ell_i) \in \mathcal{J}$, and d_i is 1 if C_i is* max *and -1 if C_i is* min.

Proof. To avoid notational clutter, we focus on the case when $\mathcal{P}$ is a homogeneous program where all limit predicates are max, in which case the inequality in the claim boils down to $\ell_1 + \Omega(\Pi) \leq \ell_j$; the proof for the general case additionally manipulates several -1 factors in a straightforward way. We proceed by induction on the length of Π.

The base case, where Π consists of a single node and $\langle C_1\mathbf{a}_1\rangle = \langle C_j\mathbf{a}_j\rangle$, is immediate.

For the inductive step, assume that $\Pi = \Pi', \langle C_j\mathbf{a}_j\rangle$ with $\Pi' = \langle C_1\mathbf{a}_1\rangle, \dots, \langle C_{j-1}\mathbf{a}_{j-1}\rangle$ and the claim holds for Π'. Let e be the edge $(\langle C_{j-1}\mathbf{a}_{j-1}\rangle, \langle C_j\mathbf{a}_j\rangle)$, let $\rho = \varphi \to C_j(\mathbf{a}_j, s)$ be the rule such that $\Omega(e) = \Omega_\rho(e)$, and let ℓ_{j-1} be such that $C_{j-1}(\mathbf{a}_{j-1}, \ell_{j-1}) \in \mathcal{J}$. Since $\Omega(\Pi) \neq \bot$, by Definition 6.4, we know that $\ell_{j-1} \in \mathbb{Z}$ whenever $(\psi(\rho, \mathcal{J}), \max s)$ is bounded, and thus have the following possibilities:

– if $(\psi(\rho, \mathcal{J}), \max s)$ is unbounded, then $\Omega(e) = \infty$;
– if $(\psi(\rho, \mathcal{J}), \max s)$ has optimal value k, then $\ell_{j-1} \in \mathbb{Z}$ and $\Omega(e) = k - \ell_{j-1}$.

In the first case, $\Omega(\Pi) = \infty$; moreover, since $\mathcal{J}$ is a pseudomodel of $\mathcal{P}$, we have $\mathcal{T}_\mathcal{P}(\mathcal{J}) \sqsubseteq \mathcal{J}$ by claim 1 of Corollary 4.9, and therefore $C_j(\mathbf{a}_j, \infty) \in \mathcal{J}$ by the definition of operator $\mathcal{T}_\mathcal{P}$, which implies $\ell_1 + \Omega(\Pi) \leq \ell_j$, as required.

In the second case, $\ell_1 + \Omega(\Pi') \leq \ell_{j-1}$ by the inductive hypothesis, and hence

$$\ell_1 + \Omega(\Pi) = \ell_1 + \Omega(\Pi') + \Omega(e) = \ell_1 + \Omega(\Pi') + k - \ell_{j-1} \leq k.$$

As before, we have $\mathcal{T}_\mathcal{P}(\mathcal{J}) \sqsubseteq \mathcal{J}$, and hence $C_j(\mathbf{a}_j, \ell_1 + \Omega(\Pi)) \sqsubseteq C_j(\mathbf{a}_j, k) \sqsubseteq C_j(\mathbf{a}_j, \ell_j)$ by the definition of operator $\mathcal{T}_\mathcal{P}$, which again implies $\ell_1 + \Omega(\Pi) \leq \ell_j$. □

We conclude this section with the following monotonicity property of value propagation graphs, which is immediate by construction and monotonicity of $\mathcal{T}_{\mathcal{P}}$ (i.e., claim 2 of Corollary 4.9).

Proposition 6.6. *For every OG-ground program $\mathcal{P}$ and pseudointerpretations $\mathcal{J}$ and $\mathcal{J}'$, let (V, E, Ω) and (V', E', Ω') be the value propagation graphs of $\mathcal{P}$ over $\mathcal{J}$ and over $\mathcal{J}'$, respectively. Then $V = V'$, and $\mathcal{J} \sqsubseteq \mathcal{J}'$ implies $E \subseteq E'$.*

6.2 Stable Programs

As already argued, the presence of a positive-weight cycle in the value propagation graph may cause an infinite sequence of rule applications. As Example 6.1 shows, however, such a cycle per se does not imply the divergence of the integer values along the cycle, since the cycle weight may decrease with applications of $\mathcal{T}_{\mathcal{P}}$, eventually becoming zero or negative. This motivates the following *stability*[4] condition, which guarantees that the edge weights in the value propagation graph may only increase with rule application. Hence, once the weight of a cycle becomes positive, it will remain positive, ensuring divergence of the numeric arguments in the pseudofacts corresponding to all nodes on the cycle. We first define the condition for OG-ground programs, leaving the general case to the end of this section; furthermore, the definition exploits monotonicity of value propagation graphs as established in Proposition 6.6.

Definition 6.7. An OG-ground program $\mathcal{P}$ is *stable* if $\Omega(e) \leq \Omega'(e)$ for all pseudointerpretations $\mathcal{J}$ and $\mathcal{J}'$ such that $\mathcal{J} \sqsubseteq \mathcal{J}'$ and all edges $e \in E$, where (V, E, Ω) and (V, E', Ω') are the value propagation graphs of $\mathcal{P}$ over $\mathcal{J}$ and over $\mathcal{J}'$, respectively.

Example 6.8. Program $\mathcal{P}_{\text{st}}$ in Example 6.1 is stable, while program $\mathcal{P}'_{\text{st}}$ is not. Indeed, consider pseudointerpretations $\mathcal{J} = \{C_1(0), C(0)\}$ and $\mathcal{J}' = \{C_1(1), C(0)\}$; then, $\mathcal{J} \sqsubseteq \mathcal{J}'$ but, for the edge $e = (\langle C_1 \rangle, \langle C_2 \rangle)$ in the corresponding value propagation graphs (V, E, Ω) and (V, E', Ω'), we have $\Omega(e) = 0$ and $\Omega'(e) = -1$.

The following lemma formulates a key property of stable OG-ground programs: a positive-weight cycle for a program $\mathcal{P}$ and pseudointerpretation $\mathcal{J}$ guarantees divergence of numeric arguments along the cycle by repeated application of $\mathcal{T}_{\mathcal{P}}$ to $\mathcal{J}$.

Lemma 6.9. *For each node $\langle C\mathbf{a} \rangle$ on a positive-weight cycle in the value propagation graph of a stable OG-ground program $\mathcal{P}$ over a pseudointerpretation $\mathcal{J}$ and for every pseudomodel $\mathcal{J}'$ of $\mathcal{P}$ such that $\mathcal{J} \sqsubseteq \mathcal{J}'$, we have $C(\mathbf{a}, \infty) \in \mathcal{J}'$.*

Proof. Let (V, E, Ω) and (V, E', Ω') be the value propagation graphs of $\mathcal{P}$ over $\mathcal{J}$ and $\mathcal{J}'$, respectively; these graphs have the same sets of nodes and $E \subseteq E'$ by Proposition 6.6. Consider a node $\langle C\mathbf{a} \rangle \in V$ on a positive-weight cycle Π in (V, E, Ω). Since $\mathcal{P}$ is stable and Π is a cycle in (V, E', Ω') as well, we have $\Omega'(\Pi) \geq \Omega(\Pi) > 0$. On the one hand, there exists $\ell \in \mathbb{Z} \cup \{\infty\}$ such that $C(\mathbf{a}, \ell) \in \mathcal{J}$ by Definition 6.4. On the other hand, the case $\ell \in \mathbb{Z}$ is not possible, because, by Lemma 6.5, this would imply $\Omega'(\Pi) \in \mathbb{Z}$ and $\ell + d \cdot \Omega'(\Pi) \preceq_C \ell$, where $d = 1$ if C is max and $d = -1$ if C is min, contradicting the positivity of $\Omega'(\Pi)$. □

While Lemma 6.9 ensures that a positive-weight cycle leads to divergence along this cycle, it does not bound the number of applications of the immediate consequence operator needed to obtain such a cycle. The following lemma closes this gap.

For each stable OG-ground program $\mathcal{P}$ and each pseudointerpretation $\mathcal{J}$, there is $i \in [1, |\mathcal{P}|]$ such that one of the following holds, where $\mathcal{J}_0 = \mathcal{J}$ and $\mathcal{J}_j = \mathcal{T}_{\mathcal{P}}(\mathcal{J}_{j-1})$ for each $j \geq 1$:

1. $\mathcal{J}_i = \mathcal{J}_{i-1}$,

[4]This notion is unrelated to stable models considered in the context of ASP.

ALGORITHM 4: Pseudomaterialisation of Stable OG-Ground Programs

```
Input: stable OG-ground program $\mathcal{P}$
Output: pseudomaterialisation $\mathcal{N}(\mathcal{P})$
1 set $\mathcal{J} := \emptyset$
2 repeat
3     set $\mathcal{J}_{\text{old}} := \mathcal{J}$
4     set $\mathcal{J} := T_{\mathcal{P}}(\mathcal{J})$
5     compute value propagation graph $(V, E, \Omega)$ of $\mathcal{P}$ over $\mathcal{J}$
6     foreach node $\langle C\mathbf{a} \rangle$ in $V$ on a positive-weight cycle in $(V, E, \Omega)$ do
7         replace $C(\mathbf{a}, \ell)$ in $\mathcal{J}$ with $C(\mathbf{a}, \infty)$
8 until $\mathcal{J} = \mathcal{J}_{\text{old}}$
9 return $\mathcal{J}$
```

2. there is a rule in $\mathcal{P}$ that is applicable to $\mathcal{J}_i$ but not to $\mathcal{J}_{i-1}$,
3. there is a node $\langle C\mathbf{a} \rangle$ on a positive-weight cycle in the value propagation graph of $\mathcal{P}$ over $\mathcal{J}_i$ such that $C(\mathbf{a}, \infty) \notin \mathcal{J}_i$.

The proof of this lemma is given in the appendix.

We are now ready to present Algorithm 4, which computes the pseudomaterialisation of a stable OG-ground program $\mathcal{P}$ in polynomial time in data complexity. The algorithm iteratively applies $T_{\mathcal{P}}$. After each application, however, it computes the corresponding value propagation graph (line 5) and replaces all integers by ∞ along all positive-weight cycles in the graph (lines 6–7); by Lemma 6.9, such replacements are always sound. Moreover, Lemma 6.2 guarantees termination of the algorithm—indeed, the lemma says that, until we reach the fixpoint, every $|\mathcal{P}|$ iterations either a new rule becomes applicable or a new pseudofact involving ∞ is introduced to the partial pseudomaterialisation; since $\mathcal{P}$ is OG-ground, each of the two cases can happen at most $|\mathcal{P}|$ times. We next make this argument formal in the proof of the following theorem.

Theorem 6.10. *There exists a polynomial p such that, for every stable OG-ground program $\mathcal{P}$, the pseudomaterialisation $\mathcal{N}(\mathcal{P})$ can be computed in time polynomial in $2^{p(u)} + \|\mathcal{P}\|$, where $u = \max_{\rho \in \mathcal{P}} [\![\rho]\!]$.*

Proof. We claim that Algorithm 4 computes $\mathcal{N}(\mathcal{P})$ for a stable OG-ground program $\mathcal{P}$ within the required complexity bounds. To this end, first note that Lemma 6.9 guarantees that $\mathcal{J} \sqsubseteq \mathcal{N}(\mathcal{P})$ throughout the computation; moreover, the algorithm terminates only if $\mathcal{J} = T_{\mathcal{P}}(\mathcal{J})$—that is, only if $\mathcal{J}$ is a pseudomodel of $\mathcal{P}$. Since the pseudomaterialisation $\mathcal{N}(\mathcal{P})$ is the minimal pseudomodel of $\mathcal{P}$ with respect to $\sqsubseteq$, these two observations imply that whenever the algorithm terminates, it outputs $\mathcal{N}(\mathcal{P})$. Thus, it remains to argue that the algorithm terminates within the stated time.

First, note that the pseudointerpretation $\mathcal{J}$ monotonically increases with respect to $\sqsubseteq$ during the execution of the main loop in lines 2–8 of the algorithm; hence, once a rule becomes applicable to $\mathcal{J}$, it remains applicable in all consequent iterations, and once a pseudofact $C(\mathbf{a}, \infty)$ is in $\mathcal{J}$, it remains in $\mathcal{J}$ in all consequent iterations. Then, Lemma 6.2 guarantees that after at most $r = |\mathcal{P}|$ iterations of the main loop, either the algorithm terminates by reaching a fixpoint, or a new rule becomes applicable, or a new pseudofact $C(\mathbf{a}, \infty)$ appears in $\mathcal{J}$. Since both the number of rules and the number of pseudofacts are bounded by r, the algorithm terminates after at most $2r^2$ iterations of the loop. Next, we analyse the complexity of each iteration.

Claim 1 of Lemma 4.10 implies that there are polynomials p_1 and p_2 such that the magnitudes of integers in $T_{\mathcal{P}}(\mathcal{J}')$ for every finite pseudointerpretation $\mathcal{J}'$ are bounded by $p_1(b)^{p_2(u)} \cdot \max(b', 1)$, where b and b' are the maximal magnitudes of integers in $\mathcal{P}$ and $\mathcal{J}'$, respectively. Therefore, the magnitudes of integers in $\mathcal{J}$ during the execution of the algorithm are bounded by $p_1(b)^{p_2'(r+u)}$, for some polynomial p_2'. Thus, since the number $|\mathcal{J}|$ of pseudofacts in $\mathcal{J}$ is bounded by r, $\|\mathcal{J}\|$ is polynomially bounded in $\|\mathcal{P}\|$ in each iteration. Therefore, by claim 2 of Lemma 4.10, computation of $\mathcal{T}_{\mathcal{P}}(\mathcal{J})$ in line 4 can be done in time polynomial in $2^{p(u)} + \|\mathcal{P}\|$, for some polynomial p.

Finally, computing the value propagation graph in line 5 boils down to checking applicability of r OG-ground rules, which is polynomial in $2^{p(u)} + \|\mathcal{P}\|$ by Proposition 2.6 since the IPs involved have no more than u variables, while detecting whether a node is on a positive-weight cycle in line 6 can be done in polynomial time using, for example, a variant of the Floyd-Warshall algorithm [28]. □

Algorithm 4 allows us to decide fact entailment for stable OG-ground programs in EXP in combined and in P in data complexity—that is, with the same complexity as for usual Datalog. In the rest of this section, we show how to extend this result to LL-programs that are not OG-ground. First of all, we generalise the notion of stability (see Definition 6.11 below). Our generalisation, however, depends on how the program is stratified: for example, the following LL-program over max predicates C_1 and C_2 is not stable if both rules belong to the same stratum, but becomes stable if they are split into two different strata.

$$\to C_1(0) \qquad\qquad C_1(m) \wedge (m \leq 5) \to C_2(m)$$

Definition 6.11. An LL-program $\mathcal{P}$ is *stable* for a stratification Λ of $\mathcal{P}$ if, for each stratum $\mathcal{P}[i]$ of $\mathcal{P}$ over Λ and for the pseudointerpretation $\mathcal{J}_{i-1}$ corresponding to the partial materialisation $\mathcal{M}_{i-1}^{\omega_1}$ of $\mathcal{P}$, the reduct $\mathcal{R}(\mathcal{P}[i] \cup \mathcal{J}_{i-1})$ is stable.

It is not difficult to see, however, that an LL-program stable for a stratification Λ is also stable for each stratification Λ' such that $\Lambda'(A) = \Lambda'(B)$ implies $\Lambda(A) = \Lambda(B)$. Thus, it is reasonable to consider stratifications with the maximal number of nonempty strata; although such stratifications are not unique, they are all linearisations of a unique partial order on predicates, and thus they either all yield stable programs or all yield nonstable ones. For example, nonrecursive programs (i.e., programs allowing for a distinct stratum for each standard predicate) are always stable for such 'maximal' stratifications. In the following, when talking about stable LL-programs without mentioning a stratification, we silently assume stability for stratifications with the maximal number of nonempty strata.

It is possible to check that the programs in Examples 3.8–3.10 and 3.12–3.14 are stable for all possible stratifications (we will see this formally in Section 6.3). However, the program in Example 3.11 (for a nonempty graph) is not stable for any stratification—comparisons $(n \leq m')$ bound the growth of numeric arguments in the same way as done by the atom $C(m)$ in Example 6.1.

We are now ready to establish the complexity of fact entailment for stable programs; the desired bounds follow by adapting Algorithm 3 to use Algorithm 4 as a subroutine instead of Algorithm 2 for computing the pseudomaterialisation of each stratum.

Theorem 6.12. *The fact entailment problem for stable LL-programs is* EXP*-complete in combined and* P*-complete in data complexity.*

Proof. The lower bounds are inherited from usual Datalog with stratified negation [14]. The upper bounds can be shown in the same way as Theorem 4.26 for arbitrary LL-programs; the only differences are that we now use Theorem 6.10 instead of Theorem 4.20 to compute the pseudomaterialisation of each stratum in line 4 of Algorithm 3 (thus avoiding calls to an NP oracle), and that we now consider a stratification with the maximal number of nonempty strata instead of $\Lambda_{\mathcal{P}}$. □

6.3 Type-Consistent Programs

Stability identifies a large subclass of LL-programs that covers most of our examples in Section 3.3 and enjoys favourable computational properties.

We next show, however, that checking stability is a computationally hard problem, which makes recognising stable programs challenging in practice. We start by showing intractability of stability checking already for OG-ground (and hence positive) programs.

Proposition 6.13. *The problem of checking stability of an OG-ground program is* coNP-*hard.*

Proof. We reduce the complement of the IP satisfiability problem, which is NP-complete by Corollary 2.5. To this end, for each IP ψ consider the OG-ground program $\mathcal{P}_\psi$ consisting of the following single rule over a max predicate C, where m is a numeric variable not mentioned in ψ.

$$\psi \wedge C(m) \wedge (m \leq 1) \rightarrow C(m+1)$$

It is immediate that if ψ is satisfiable, then $\mathcal{P}_\psi$ is not stable by the same reasons as program $\mathcal{P}'_{\text{st}}$ in Examples 6.1 and 6.8; if ψ is unsatisfiable, however, the rule never becomes applicable, and hence $\mathcal{P}_\psi$ is trivially stable. □

Furthermore, as we show next, it is undecidable to check whether a given LL-program is stable for all possible datasets, and hence stability would need to be rechecked every time data changes in an application.

Proposition 6.14. *The problem of checking, for a positive LL-program $\mathcal{P}$, whether $\mathcal{P} \cup \mathcal{D}$ is stable for every dataset $\mathcal{D}$ is undecidable.*

Proof. As in the proof of Theorem 3.6, we use a reduction of Hilbert's tenth problem. For each polynomial $p(m_1, \dots, m_v)$ over variables $m_1, \dots, m_v$, let $\mathcal{P}_p$ be the positive LL-program consisting of the following rule over exact predicates $B_1, \dots, B_v$ and a max predicate C.

$$B_1(m_1) \wedge \cdots \wedge B_v(m_v) \wedge (p(m_1, \dots, m_v) \doteq 0) \wedge C(m) \wedge (m \leq 1) \rightarrow C(m+1)$$

Note that program $\mathcal{P}_p$ is indeed limit-linear since variables $m_1, \dots, m_v$ are all guarded. On the one hand, if the equation $p(m_1, \dots, m_v) = 0$ has an integer solution, then on the basis of this solution we can construct a dataset $\mathcal{D}$ over $B_1, \dots, B_v$ such that $\mathcal{P} \cup \mathcal{D}$ is not stable by the same reasons as program $\mathcal{P}'_{\text{st}}$ in Examples 6.1 and 6.8. On the other hand, if the equation does not have a solution, then the rule is never applicable and hence $\mathcal{P} \cup \mathcal{D}$ is stable for every $\mathcal{D}$. □

To overcome these difficulties, we next propose in Definition 6.15 a sufficient condition for stability, which can be checked syntactically rule by rule and which is rich enough to capture a wide range of stable programs such as those in Examples 3.8–3.10 and 3.12–3.14 (recall that the program in Example 3.11 is not stable). Intuitively, the requirements in Definition 6.15 syntactically prevent certain interactions between numeric arguments matching min and max atoms, thus eliminating cases that can be problematic for stability. For instance, in the new rule of program $\mathcal{P}'_{\text{st}}$ from Example 6.1, stability is violated because there is a numeric variable, m, that occurs in the head and in two positive max literals in the body.

Definition 6.15. A *min-max typing* of variables in an LL-rule is a partitioning of all unguarded numeric variables that occur in positive limit literals in the rule into *max* and *min* types. Given a min-max typing, a numeric term is of type *max* if it is either ∞ or has the form

$$s + \left(\sum_{i=1}^{v} k_i \times m_i\right) - \left(\sum_{j=1}^{w} \ell_j \times n_j\right),$$

where s is a numeric term not mentioning any max or min variables, $v \in \mathbb{N}$, $w \in \mathbb{N}$, each m_i is a max variable with coefficient $k_i \geq 1$, and each n_j is a min variable with coefficient $\ell_i \geq 1$; a numeric term is of type *min* if the same holds except that each m_i is min and each n_j is max.

Rule $\rho = \varphi \rightarrow \alpha$ is *type-consistent* if it has a min-max typing with the following properties:

- each numeric variable in each negative exact literal in φ is guarded;
- the numeric term of each max and each min atom in ρ is of type max and min, respectively;
- each comparison atom in φ has the form $(s_1 < s_2)$ or $(s_1 \leq s_2)$, for term s_1 of type min and term s_2 of type max; and
- if $\alpha = C_2(\mathbf{a}_2, s_2)$ is a limit atom, then, for each positive limit literal $C_1(\mathbf{a}_1, s_1)$ in φ with s_2 depending on s_1, terms s_1 and s_2 have a common unguarded variable that has coefficient 1 in s_1 and does not appear in any other positive limit literals in φ, where *dependency* is defined as in Definition 6.2 except that only unguarded numeric variables are taken into account.

An LL-program is *type-consistent* if so are all of its rules.

Note that in our previous definition a numeric term that mentions only integers and guarded variables is both of type max and of type min.

Type consistency is a syntactic property that, by definition, can be checked rule by rule; moreover, it can be checked efficiently, as established by the following proposition.

Proposition 6.16. *Checking whether an LL-program is type-consistent is* L*-complete.*

Proof. We start with an L algorithm. By definition, type-consistency of an LL-program can be checked by considering each of its rules in separation. Given an LL-rule $\rho = \varphi \rightarrow \alpha$, checking whether a numeric variable is guarded in ρ can be done in L. The same applies to checking whether a numeric term in a positive limit literal has the right shape; if so, the shape of the term and the type of the respective predicate uniquely determine the types of all unguarded variables in the term. The rest of the type-consistency check boils down to checking dependency between unguarded variables, which is essentially reachability in an undirected graph (with unguarded variables as nodes and edges defined by co-occurrence of two such variables in an atom). This is doable in symmetric logarithmic space, and hence in L by the result of Reingold [47].

We next show L-hardness by reduction of the complement of the undirected graph reachability problem (which is L-complete [47]). Let G be an undirected graph, and let v_1 and v_2 be two distinct nodes in G. Consider the following LL-rule ρ, where C_1 and C_2 are max predicates, for each node v in G, m_v is a numeric variable uniquely associated with v, and φ is a conjunction of comparison atoms $(0 < m_v + m_{v'})$ for each edge $\{v, v'\}$ in G.

$$C_1(m_{v_1}) \wedge \varphi \rightarrow C_2(m_{v_2})$$

By construction, v_2 is reachable from v_1 in G if and only if m_{v_2} depends on m_{v_1} in ρ, and so ρ is type-consistent if and only if the reachability does not hold. □

It is easily seen that the LL-programs in Examples 3.8–3.10 and 3.12–3.14 are type-consistent. In the rest of the section, we show that all type-consistent LL-programs are stable, which, by Theorem 6.12, implies that such programs have the same complexity of fact entailment as usual Datalog programs. We first show the claim for OG-ground programs.

Lemma 6.17. *Each type-consistent OG-ground program is stable.*

Proof. As before, to avoid notational clutter, we focus on homogeneous limit programs all of whose limit predicates are max.

Consider an arbitrary type-consistent OG-ground program $\mathcal{P}$. It suffices to show that, for each rule $\rho = \varphi \rightarrow C_2(\mathbf{a}_2, s_2)$ in $\mathcal{P}$ with a literal $C_1(\mathbf{a}_1, s_1)$ in φ such that C_1 and C_2 are limit predicates and

s_2 depends on s_1, and for all pseudointerpretations $\mathcal{J}$ and $\mathcal{J}'$ such that $\mathcal{J} \sqsubseteq \mathcal{J}'$ and ρ is applicable to $\mathcal{J}$, we have $\Omega_\rho(e) \leq \Omega'_\rho(e)$, where $e = (\langle C_1\mathbf{a}_1\rangle, \langle C_2\mathbf{a}_2\rangle)$ is an edge in the value propagation graphs $G = (V, E, \Omega)$ and $G' = (V, E', \Omega')$ of $\mathcal{P}$ over $\mathcal{J}$ and over $\mathcal{J}'$, respectively (note that G and G' have the same set of nodes and $e \in E \subseteq E'$ by Proposition 6.6). To this end, consider arbitrary such ρ, $C_1(\mathbf{a}_1, s_1)$, $C_2(\mathbf{a}_2, s_2)$, $\mathcal{J}$, $\mathcal{J}'$, G, and G'. Note that, since ρ is OG-ground and type-consistent, and s_2 depends on s_1, terms s_1 and s_2 are different from ∞ and, moreover, either $s_1 = s'_1 + 1 \times m$ and $s_2 = s'_2 + k_m \times m$ for $k_m \geq 1$ and m a max variable that occurs neither in s'_1 nor in other limit atoms in ρ, or $s_1 = s'_1 - 1 \times n$ and $s_2 = s'_2 - \ell_n \times n$ for $\ell_n \geq 1$ and n a min variable that appears neither in s'_1 nor in other limit atoms in ρ. We next focus on the first case; the second case is symmetric.

First, note that if $C_1(\mathbf{a}_1, \infty) \in \mathcal{J}$, then the IOP $(\psi(\rho, \mathcal{J}), \max s_2)$ is unbounded. Indeed, if $C_1(\mathbf{a}_1, \infty) \in \mathcal{J}$ then, for each solution σ to $\psi(\rho, \mathcal{J})$ mapping m to some $k \in \mathbb{Z}$ and each integer $k' > k$, the function σ' mapping m to k' and all other variables to the same values as σ is also a solution, because

- m occurs only negatively on the left-hand side of each comparison atom in φ,
- m occurs only positively on the right-hand side of each comparison atom, and
- m does not occur in any standard literal in φ except $C_1(\mathbf{a}_1, s_1)$;

consequently, increasing the value of m to k' does not invalidate any literal in φ. Furthermore, since m does not occur in s'_2, we have $s_2\sigma' = s_2\sigma + k_m \cdot (k' - k) \geq s_2\sigma + (k' - k)$; clearly, this allows us, for each $\ell \in \mathbb{Z}$, to pick σ' such that $s_2\sigma' \geq \ell$. For the same reasons, $(\psi(\rho, \mathcal{J}'), \max s_2)$ is unbounded whenever $C_1(\mathbf{a}_1, \infty) \in \mathcal{J}'$. Therefore, $\Omega_\rho(e) \neq \bot$ and $\Omega'_\rho(e) \neq \bot$ for $e = (\langle C_1\mathbf{a}_1\rangle, \langle C_2\mathbf{a}_2\rangle)$, and we have two cases: either $\Omega_\rho(e) = \infty$ or $\Omega_\rho(e) \in \mathbb{Z}$.

If $\Omega_\rho(e) = \infty$ then $(\psi(\rho, \mathcal{J}), \max s_2)$ is unbounded. Since $\mathcal{J} \sqsubseteq \mathcal{J}'$, each solution to $\psi(\rho, \mathcal{J})$ is also a solution to $\psi(\rho, \mathcal{J}')$. Thus, $(\psi(\rho, \mathcal{J}'), \max s_2)$ is also unbounded and $\Omega'_\rho(e) = \infty$, as required.

Consider now the case $\Omega_\rho(e) \in \mathbb{Z}$—that is, $\Omega_\rho(e) = k - \ell$ for $\ell \in \mathbb{Z}$ with $C_1(\mathbf{a}_1, \ell) \in \mathcal{J}$ and k the optimal value of the IOP $(\psi(\rho, \mathcal{J}), \max s_2)$. If $C_1(\mathbf{a}_1, \infty) \in \mathcal{J}'$ then, as already argued, $(\psi(\rho, \mathcal{J}'), \max s_2)$ is unbounded; hence $\Omega'_\rho(e) = \infty$, and so $\Omega_\rho(e) \leq \Omega'_\rho(e)$, as required. Otherwise, let $C_1(\mathbf{a}_1, \ell') \in \mathcal{J}'$ for $\ell' \in \mathbb{Z}$ and consider the solution σ to $\psi(\rho, \mathcal{J})$ such that $s_2\sigma$ is the optimal value of the IOP—that is, such that $s_2\sigma = s'_2\sigma + k_m \cdot m\sigma = k$. In particular, we have $s_1\sigma \leq \ell$. Let σ' be a function mapping m to $m\sigma + \ell' - \ell$ and all other variables to the same values as σ. Function σ' is a solution to $\psi(\rho, \mathcal{J}')$, because

- $s_1\sigma' = s'_1\sigma' + m\sigma' = s'_1\sigma + m\sigma + \ell' - \ell = (s_1\sigma - \ell) + \ell' \leq \ell'$ and hence $\mathcal{J}' \models C_1(\mathbf{a}_1, s_1\sigma')$,
- pseudointerpretation $\mathcal{J}'$ satisfies all other standard literals in $\varphi\sigma'$ since they do not mention m and $\mathcal{J} \sqsubseteq \mathcal{J}'$, and
- $\mathcal{J}'$ satisfies all comparison literals in $\varphi\sigma'$ since m occurs only negatively in terms on the left and only positively in those on the right of each comparison.

Since $\mathcal{J} \sqsubseteq \mathcal{J}'$ and hence $\ell \leq \ell'$, the following holds, for k' the optimal value of $(\psi(\rho, \mathcal{J}'), \max s_2)$:

$$\begin{aligned}\Omega_\rho(e) = k - \ell = s'_2\sigma + k_m \cdot m\sigma - \ell = s'_2\sigma + k_m \cdot m\sigma - k_m \cdot \ell + (k_m - 1) \cdot \ell \leq \\ s'_2\sigma + k_m \cdot m\sigma - k_m \cdot \ell + (k_m - 1) \cdot \ell' = s'_2\sigma + k_m \cdot (m\sigma + \ell' - \ell) - \ell' = \\ s'_2\sigma + k_m \cdot m\sigma' - \ell' \leq s'_2\sigma' + k_m \cdot m\sigma' - \ell' = s_2\sigma' - \ell' \leq k' - \ell' = \Omega'_\rho(e);\end{aligned}$$

note that $s'_2\sigma \leq s'_2\sigma'$ because m is max and may appear in s'_2 only positively. □

Finally, we extend the result of Lemma 6.17 to arbitrary LL-programs.

Theorem 6.18. *Each type-consistent LL-program is stable.*

Proof. By Lemma 6.17, we only need to show that, for each stratum $\mathcal{P}[i]$ of a type-consistent LL-program $\mathcal{P}$ and for $\mathcal{J}_{i-1}$ the pseudointerpretation corresponding to the partial materialisation

$\mathcal{M}_{i-1}^{\omega_1}$ of $\mathcal{P}$, the reduct $\mathcal{R}(\mathcal{P}[i] \cup \mathcal{J}_{i-1})$ is type-consistent. Consider an arbitrary rule $\rho = \varphi \rightarrow \alpha$ in the reduct $\mathcal{R}(\mathcal{P}[i] \cup \mathcal{J}_{i-1})$, for $\mathcal{P}[i]$ and $\mathcal{J}_{i-1}$ as above, as well as the rule $\rho' = \varphi' \rightarrow \alpha$ in $\mathcal{G}(\mathcal{P}[i] \cup \mathcal{J}_{i-1})$ from which ρ was obtained. Note that ρ' is type-consistent because Definition 6.15 does not distinguish between constants, object variables, and guarded numeric variables. We next check that the four conditions of Definition 6.15 hold for ρ.

First, ρ is OG-ground and hence positive, so it does not have any negative exact literals. Therefore, the first condition holds vacuously.

Second, all limit atoms in ρ are inherited from ρ', so the second condition also holds.

Third, consider a comparison atom β in φ. If β is inherited from φ', then the third condition holds for β since ρ' is type-consistent. If β is obtained by replacing a negative exact literal in φ', then, since all numeric variables in negative exact literals in $\mathcal{P}$ are guarded by the first type-consistency condition, β is variable-free and hence also satisfies the third condition (recall that variable-free terms are both of type max and min). Finally, if $\beta = (k \prec_C s)$ is obtained by replacing a negative limit literal $\mathsf{not}\, C(\mathbf{a}, s)$ in φ', then s is of type max if C is max and of type min if C is min by the second type-consistency condition, and so, since k is a constant, β also satisfies the third condition.

Finally, by construction, no variable in ρ or ρ' is guarded and, for each literal in ρ, rule ρ' contains a literal with the same numeric variables. Therefore, if a term s_2 depends on a term s_1 in ρ then s_2 depends on s_1 in ρ' as well. So, the fourth condition holds because all positive limit literals in φ and the head α of ρ are inherited from ρ'. □

7 AGGREGATION

As illustrated in Examples 3.9–3.12, aggregation can often be simulated via recursion and arithmetic. This simulation is, however, rather low-level and may yield programs that are not very intuitive. Moreover, the encodings in our examples rely on the assumption that input datasets are ordered. In this section, we extend limit(-linear) $Datalog_{\mathbb{Z}}$ to a language that can handle aggregation with grouping natively, which we call limit(-linear) $Datalog_{\mathbb{Z}}^{\mathrm{agg}}$. On the one hand, limit(-linear) $Datalog_{\mathbb{Z}}^{\mathrm{agg}}$ will allow us to express programs such as those in our examples in a more intuitive way and without assuming an order on datasets; on the other hand, the language can be polynomially translated to limit(-linear) $Datalog_{\mathbb{Z}}$ over ordered datasets.

For aggregation to be meaningful, however, the definition of limit $Datalog_{\mathbb{Z}}^{\mathrm{agg}}$ should satisfy certain properties, which we summarise next.

1. Whenever a limit-closed interpretation contains a fact over a given limit predicate for a given tuple of objects, it contains infinitely many such facts; thus, it makes little sense to aggregate over all such facts as, for typical aggregate functions, the outcome will be either infinite or equal to the greatest or the least numeric value; hence, in our definition, it should suffice to consider a single ‘representative’ fact for each predicate and tuple of objects.
2. Our semantics assumes that the immediate consequence operator is monotone; hence, our definition should guarantee that Proposition 3.1 extends to $Datalog_{\mathbb{Z}}^{\mathrm{agg}}$.
3. When applying a rule, it should be possible to compare the result of an aggregation with a numeric term or use it in the head of a rule; so, such a result should always be an integer.

As demonstrated by the following example, the second property can be violated even by such seemingly innocuous aggregation operations as summation over a set of facts.

Example 7.1. Consider the pseudointerpretations $\mathcal{J} = \{C(a, 1)\}$, $\mathcal{J}' = \mathcal{J} \cup \{C(b, -1)\}$, and $\mathcal{J}'' = \mathcal{J} \cup \{C(b, 1)\}$, for C a max predicate. Clearly, it makes little sense to sum up the integers in all the infinitely many facts in the limit-closed interpretations corresponding to $\mathcal{J}$, $\mathcal{J}'$, and $\mathcal{J}''$, as in all three cases the result would be infinitely small (i.e., smaller than any integer). More meaningful, and more in line with the examples in Section 3.3, is to aggregate over all facts that are maximal with respect to $\sqsubseteq$. This, however, leads to nonmonotonicity of the immediate consequence operator: note that $\mathcal{J} \sqsubseteq \mathcal{J}'$ and $\mathcal{J} \sqsubseteq \mathcal{J}''$ but the sum k of the integers in all facts over C is 1 for $\mathcal{J}$, 0 for $\mathcal{J}'$, and 2 for $\mathcal{J}''$; thus, for a rule deriving a limit fact $C'(k)$, the immediate consequence operator would be nonmonotonic regardless of whether the predicate C' is min or max.

To avoid this issue, we introduce two aggregate functions for summation: one for positive integers (where negative numbers are ignored) and one for negative integers (where positive numbers are ignored). Similarly, the second property is easily seen to rule out recursive use of the average aggregate function, which is therefore not included in our language (nonrecursive use of integer average is unproblematic but is not discussed in this paper). The third property implies that we should avoid aggregating over infinite multisets; this is, however, not an issue if the first property is satisfied. The third property also implies that we should avoid taking the maximum or the minimum over an empty multiset, which will be ensured by our semantics.

We begin our formalisation of limit $\mathit{Datalog}_{\mathbb{Z}}^{\mathsf{agg}}$ by defining the following aggregate functions.

Definition 7.2. An *aggregate function* is one of $\mathtt{count}$, $\mathtt{max}$, $\mathtt{min}$, $\mathtt{sum}^{+}$, and $\mathtt{sum}^{-}$. It maps each finite multiset M of integers to an integer as follows, where duplicates in M are considered separately:

- $\mathtt{count}(M)$ is the number of elements in M;
- $\mathtt{max}(M)$ and $\mathtt{min}(M)$ are undefined when M is empty, and are otherwise the maximum and the minimum of all elements in M, respectively; and
- $\mathtt{sum}^{+}(M)$ and $\mathtt{sum}^{-}(M)$ are the sums of all positive and all negative elements in M, respectively.

Next, we define the syntax of limit and limit-linear $\mathit{Datalog}_{\mathbb{Z}}^{\mathsf{agg}}$. For simplicity, we allow aggregation with grouping only over a single limit atom; aggregation over other types of atoms (e.g., counting over object atoms) as well as over conjunctions of atoms can be handled in a similar way.

Definition 7.3. An *aggregate atom* has the form

$$(s \lhd \mathsf{agg}\, C(\mathbf{t}, \text{-})\ \mathtt{group\ by}\ \mathbf{t}'),$$

where s is a numeric term different from ∞; agg is an aggregate function; $\lhd$ is $\leq$ or $<$ when agg is $\mathtt{count}$, $\mathtt{max}$ or $\mathtt{sum}^{+}$, and $\geq$ or $>$ when agg is $\mathtt{min}$ or $\mathtt{sum}^{-}$; C is a v-ary numeric predicate; $\mathbf{t}$ is a $(v-1)$-tuple of object terms; $\mathbf{t}'$ is a subset of the terms in $\mathbf{t}$, called *grouping* terms, while all other terms in $\mathbf{t}$ are called *local*; and - is a special symbol (whose only role is to match the arity of C).

A $\mathit{Datalog}_{\mathbb{Z}}^{\mathsf{agg}}$ *rule* is defined as an ordinary $\mathit{Datalog}_{\mathbb{Z}}$ rule in equation (3) except that literals in the body can also be aggregate atoms, where each local variable in an aggregate atom appears only in that atom and is not mentioned in the universal quantifier of the rule. *Limit* and *limit-linear* $\mathit{Datalog}_{\mathbb{Z}}^{\mathsf{agg}}$ programs are defined in the same way as for $\mathit{Datalog}_{\mathbb{Z}}$, with aggregate atoms in rule bodies regarded as positive literals for the definitions of stratification and safety, and with the additional requirement that only limit predicates are allowed in aggregate atoms.

We emphasise that, in this definition, $\mathbf{t}$ is a tuple of object terms, while $\mathbf{t}'$ is a subset of the set of terms of $\mathbf{t}$—that is, does not have an order and repetitions (and may contain objects, which, as we will see, do not influence the result but are useful for the uniformity of the following definitions). Using similar notation here should not lead to any confusion, because the position of an expression $\mathbf{t}$ uniquely identifies whether $\mathbf{t}$ is a tuple or a set.

Having formally defined the syntax of limit $\mathit{Datalog}_{\mathbb{Z}}^{\mathsf{agg}}$, we next move to its semantics.

Definition 7.4. An aggregate atom is *ground* if all of its variables are local. An interpretation is *object-finite* if it mentions only a finite number of objects. A *sample* of a limit predicate C in a object-finite limit-closed interpretation $\mathcal{I}$ is a (finite) subset of $\mathcal{I}$ containing exactly one fact $C(\mathbf{a}, k)$ for each C and $\mathbf{a}$ such that $\{C(\mathbf{a}, k') \mid k' \in \mathbb{Z}\} \cap \mathcal{I}$ is nonempty. Interpretation $\mathcal{I}$ *satisfies* a ground aggregate atom $\beta = (s \lhd \mathsf{agg}\, C(\mathbf{t}, \text{-})\ \mathsf{group\ by}\ \mathbf{a})$, written $\mathcal{I} \models \beta$, if there exists a sample $\mathcal{K}$ of C in $\mathcal{I}$ such that $s \lhd \mathsf{agg}(M)$ evaluates to true, where M is the multiset of all integers ℓ for each of which there is a distinct substitution σ of all (local) variables in $\mathbf{t}$ such that $C(\mathbf{t}\sigma, \ell) \in \mathcal{K}$.

The immediate consequence operator $\mathcal{S}_{(\mathcal{P},\tau)}$ for a limit *Datalog*$_{\mathbb{Z}}^{\mathsf{agg}}$ program $(\mathcal{P}, \tau)$ is then defined in the same way as for ordinary *Datalog*$_{\mathbb{Z}}$ in Definition 3.4; we will continue to use the notation $\mathcal{S}_{\mathcal{P}}$ instead of $\mathcal{S}_{(\mathcal{P},\tau)}$, as discussed at the end of Section 3.1.

This definition requires several clarifications. First, note that the definition of a ground aggregate atom is an instantiation of our general definition of a ground expression in Section 2.1, under the assumption that local variables are not treated as usual. Second, grounding of the set of grouping terms used in the definition of operator $\mathcal{S}_{\mathcal{P}}$ implicitly assumes duplicate elimination: for example, for a substitution σ assigning object a to object variable x and the set $\{a, b, x\}$ with an object b, we have $\{a, b, x\}\sigma = \{a, b\}$. Third, note that objects in the set of grouping terms are essentially irrelevant: they simply ensure that the grounding of an aggregate atom remains an aggregate atom. Finally, since functions max and min are undefined on empty inputs, multisets involved in the evaluation of a ground aggregate atom β using max or min on an object-finite limit-closed interpretation $\mathcal{I}$ must be nonempty whenever $\mathcal{I} \models \beta$.

The following result generalises Proposition 3.1 to *Datalog*$_{\mathbb{Z}}^{\mathsf{agg}}$, establishing that the immediate consequence operator still has the desired monotonicity properties.

The following holds:

1. $\mathcal{I} \models \mathcal{P}$ if and only if $\mathcal{S}_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$, for each limit *Datalog*$_{\mathbb{Z}}^{\mathsf{agg}}$ program $\mathcal{P}$ and each object-finite limit-closed interpretation $\mathcal{I}$; and
2. $\mathcal{S}_{\mathcal{P}}(\mathcal{I}_1) \subseteq \mathcal{S}_{\mathcal{P}}(\mathcal{I}_2)$ for each semi-positive limit *Datalog*$_{\mathbb{Z}}^{\mathsf{agg}}$ program $\mathcal{P}$ and each object-finite limit-closed interpretations $\mathcal{I}_1$ and $\mathcal{I}_2$ that satisfy $\mathcal{I}_1 \subseteq \mathcal{I}_2$ and coincide on the EDB predicates.

The proof of this proposition is given in the appendix.

As for ordinary *Datalog*$_{\mathbb{Z}}$ programs, claim 2 of Proposition 7 and the Knaster-Tarski theorem guarantee the existence of the least fixpoint of the immediate consequence operator for each stratum of a limit *Datalog*$_{\mathbb{Z}}^{\mathsf{agg}}$ program, and hence we can define materialisations and fact entailment for such programs in the same way as before.

Definition 7.5. The *partial materialisations* and the *materialisation* $\mathcal{M}(\mathcal{P})$ of a limit *Datalog*$_{\mathbb{Z}}^{\mathsf{agg}}$ program $\mathcal{P}$ are defined in the same way as for limit *Datalog*$_{\mathbb{Z}}$ (see Section 2.1 and Definition 3.5). A limit *Datalog*$_{\mathbb{Z}}^{\mathsf{agg}}$ program $\mathcal{P}$ *entails* a fact γ, written $\mathcal{P} \models \gamma$, if and only if $\gamma \in \mathcal{M}(\mathcal{P})$.

Note that, according to this definition, the (partial) materialisations are object-finite since they mention only objects in the program.

We next illustrate how aggregates can be used to obtain more concise, natural and order-unaware formulations of the analysis tasks in Examples 3.9 and 3.10 (Examples 3.11 and 3.12 can be reformulated using aggregates in a similar way).

Example 7.6 (Diffusion in social networks). Consider again the setting of Example 3.9, where a dataset $\mathcal{D}_{\mathrm{tw}}$ contains an object fact $\mathit{tweet}(a_s)$, representing that agent a_s introduces a tweet, object facts $\mathit{follows}(a, a')$, representing that a follows a', and exact facts $\mathit{threshold}(a, k_a)$, encoding the threshold $k_a > 0$ of a, and where the goal is to check how the tweet propagates through the network. We can now restate program $\mathcal{P}_{\mathrm{tw}}$ in Example 3.9 in a much simpler way as a limit-linear *Datalog*$_{\mathbb{Z}}^{\mathsf{agg}}$ program $\mathcal{P}_{\mathrm{tw}}^{\mathsf{agg}}$ consisting of rules (73) and (74). Rule (73) defines a max predicate *sees* such that a fact $\mathit{sees}(a, a', 1)$ is true if an agent a follows an agent a' while a' tweets the message, and so a sees a' tweeting the message (the numeric value 1 is inessential, and any other can be used instead). Rule (74) then counts how many agents are seen tweeting the message by an agent and compares it to the agent's threshold.

$$\mathit{follows}(x, y) \wedge \mathit{tweet}(y) \rightarrow \mathit{sees}(x, y, 1) \tag{73}$$

$$\mathit{threshold}(x, m) \wedge (m \leq \texttt{count}\, \mathit{sees}(x, y, \text{-})\ \texttt{group by}\ x) \rightarrow \mathit{tweet}(x) \tag{74}$$

We have that $\mathcal{P}^{\mathsf{agg}}_{\mathsf{tw}} \cup \mathcal{D}_{\mathsf{tw}} \models \mathit{tweet}(a)$ if and only if agent a tweets the message.

Example 7.7 (Counting paths). Consider the setting of Example 3.10, where a dataset $\mathcal{D}_{\mathsf{cp}}$ encodes a directed acyclic graph using a unary object predicate *node* and a binary object predicate *edge*, and the goal is to compute the number of paths between each two nodes. We can restate program $\mathcal{P}_{\mathsf{cp}}$ in Example 3.10 as the limit-linear $\mathit{Datalog}^{\mathsf{agg}}_{\mathbb{Z}}$ program $\mathcal{P}^{\mathsf{agg}}_{\mathsf{cp}}$ consisting of rules (75)–(77), where *path-num* and *path-num-z* are max predicates. Intuitively, $\mathit{path\text{-}num\text{-}z}(a, a', b, k)$ is true if there are at least k paths from a node a to a node a' that begin with the edge (a, b); see rule (76). For a and a' distinct nodes, rule (77) then defines ℓ in $\mathit{path\text{-}num}(a, a', \ell)$ as the sum of all values k for which $\mathit{path\text{-}num\text{-}z}(a, a', b, k)$ holds, while, for the case $a = a'$, ℓ is defined by rule (75).

$$\mathit{node}(x) \rightarrow \mathit{path\text{-}num}(x, x, 1) \tag{75}$$

$$\mathit{edge}(x, z) \wedge \mathit{path\text{-}num}(z, y, m) \rightarrow \mathit{path\text{-}num\text{-}z}(x, y, z, m) \tag{76}$$

$$(m \leq \texttt{sum}^{+}\, \mathit{path\text{-}num\text{-}z}(x, y, z, \text{-})\ \texttt{group by}\ x, y) \rightarrow \mathit{path\text{-}num}(x, y, m) \tag{77}$$

We have $\mathcal{P}^{\mathsf{agg}}_{\mathsf{cp}} \cup \mathcal{D}_{\mathsf{cp}} \models \mathit{path\text{-}num}(a, a', k)$ if and only if there are at least k paths from a to a'.

Next, we establish the main theorem of this section, which states that limit $\mathit{Datalog}^{\mathsf{agg}}_{\mathbb{Z}}$ neither adds expressive power to $\mathit{Datalog}_{\mathbb{Z}}$ nor increases the complexity of reasoning.

Theorem 7.8. *For each limit $\mathit{Datalog}^{\mathsf{agg}}_{\mathbb{Z}}$ program $\mathcal{P}$, there exists a limit $\mathit{Datalog}_{\mathbb{Z}}$ program $\mathcal{P}'$, which is limit-linear if so is $\mathcal{P}$, such that $\mathcal{P} \models \gamma$ if and only if $\mathcal{P}' \models \gamma$ for every fact γ over a predicate in $\mathcal{P}$; furthermore, $\|\mathcal{P}'\|$ is polynomial in $\|\mathcal{P}\|$.*

Proof. Let $\mathcal{P}$ be a limit $\mathit{Datalog}^{\mathsf{agg}}_{\mathbb{Z}}$ program. Let $\mathcal{D}_{\mathsf{ord}}$ be a dataset encoding an order on all objects $a_1, \ldots, a_v$ in $\mathcal{P}$ using the following facts over Σ_{ord}.

$$\rightarrow \mathit{first}(a_1) \qquad \rightarrow \mathit{next}(a_1, a_2) \quad \cdots \quad \rightarrow \mathit{next}(a_{v-1}, a_v) \qquad \rightarrow \mathit{last}(a_v)$$

Let $\mathcal{P}_{\mathsf{succ}}$ be the program consisting of rules (41) from the proof of Theorem 5.3 and, for each number w such that $\mathcal{P}$ has an aggregate atom with w local variables, rules (42), but for each $i \in [0, w-1]$ and using a predicate succ_w of arity $2w$ instead of *succ*; program $\mathcal{P}_{\mathsf{succ}}$ extends the order in $\mathcal{D}_{\mathsf{ord}}$ to w-tuples of objects.

Then, for each aggregate atom $\beta = (s \lhd \mathsf{agg}\, C(\mathbf{t}, \text{-})\ \texttt{group by}\ \mathbf{t}')$ in $\mathcal{P}$, we construct a program $\mathcal{P}_\beta$. If $\mathsf{agg} = \texttt{count}$, then $\mathcal{P}_\beta$ is defined as follows, where $\mathbf{x} = x_1, \ldots, x_u$ and $\mathbf{y} = y_1, \ldots, y_w$ are the grouping and the local variables of β, respectively, while acc_β and agg_β are fresh max predicates of arities $u + w + 1$ and $u + 1$, respectively.

$$\mathit{object}(x_1) \wedge \cdots \wedge \mathit{object}(x_u) \wedge \mathit{first}(y_1) \wedge \cdots \wedge \mathit{first}(y_w) \rightarrow \mathit{acc}_\beta(\mathbf{x}, \mathbf{y}, 0) \tag{78}$$

$$\mathit{object}(x_1) \wedge \cdots \wedge \mathit{object}(x_u) \wedge \mathit{first}(y_1) \wedge \cdots \wedge \mathit{first}(y_w) \wedge C(\mathbf{t}, n) \rightarrow \mathit{acc}_\beta(\mathbf{x}, \mathbf{y}, 1) \tag{79}$$

$$\mathit{acc}_\beta(\mathbf{x}, \mathbf{y}', m) \wedge \mathit{succ}_w(\mathbf{y}', \mathbf{y}) \rightarrow \mathit{acc}_\beta(\mathbf{x}, \mathbf{y}, m) \tag{80}$$

$$\mathit{acc}_\beta(\mathbf{x}, \mathbf{y}', m) \wedge \mathit{succ}_w(\mathbf{y}', \mathbf{y}) \wedge C(\mathbf{t}, n) \rightarrow \mathit{acc}_\beta(\mathbf{x}, \mathbf{y}, m + 1) \tag{81}$$

$$\mathit{acc}_\beta(\mathbf{x}, \mathbf{y}, m) \wedge \mathit{last}(y_1) \wedge \cdots \wedge \mathit{last}(y_w) \rightarrow \mathit{agg}_\beta(\mathbf{x}, m) \tag{82}$$

If $\mathsf{agg} \in \{\texttt{max}, \texttt{min}\}$, let $\mathcal{P}_\beta$ be the program consisting of the following rule, where $\mathbf{x}$ is as above, while agg_β is a fresh predicate of arity $u + 1$ that has the same type (max or min) as C.

$$C(\mathbf{t}, n) \rightarrow \mathit{agg}_\beta(\mathbf{x}, n)$$

If $\mathsf{agg} \in \{\mathsf{sum}^+, \mathsf{sum}^-\}$, let $\mathcal{P}_\beta$ be the program consisting of rules (78), (80), (82), and the following rules in place of (79) and (81), where $\mathbf{x}$ and $\mathbf{y}$ are as before, acc_β and agg_β are fresh predicates of arities $u + w + 1$ and $u + 1$, respectively, that are max if $\mathsf{agg} = \mathsf{sum}^+$ and min otherwise.

$$\begin{aligned} & object(x_1) \land \cdots \land object(x_u) \land \\ & \quad first(y_1) \land \cdots \land first(y_w) \land C(\mathbf{t}, n) \land (0 \prec_{acc_\beta} n) \to acc_\beta(\mathbf{x}, \mathbf{y}, n) \\ & acc_\beta(\mathbf{x}, \mathbf{y}', m) \land succ_w(\mathbf{y}', \mathbf{y}) \land C(\mathbf{t}, n) \land (0 \prec_{acc_\beta} n) \to acc_\beta(\mathbf{x}, \mathbf{y}, m + n) \end{aligned}$$

Finally, let $\mathcal{P}'$ be obtained from $\mathcal{P}$ by adding $\mathcal{D}_{\mathrm{ord}}$, $\mathcal{P}_{\mathrm{succ}}$, and, for each aggregate atom β in $\mathcal{P}$, adding $\mathcal{P}_\beta$ and replacing β with the conjunction $agg_\beta(\mathbf{x}, m_\beta) \land (s \lhd m_\beta)$, with $\mathbf{x}$ as before, m_β a fresh numeric variable uniquely associated with β, and $\lhd$ the comparison predicate in β.

Having completed the construction of $\mathcal{P}'$, we next argue that it has the claimed properties. First, $\mathcal{P}'$ is limit-linear whenever so is $\mathcal{P}$, and $\|\mathcal{P}'\|$ is polynomial in $\|\mathcal{P}\|$ by construction. So, it is left to show that $\mathcal{P} \models \gamma$ if and only if $\mathcal{P}' \models \gamma$ for every fact γ over a predicate in $\mathcal{P}$. By construction, it suffices to prove that, for each aggregate atom $\beta = (s \lhd \mathsf{agg}\, C(\mathbf{t}, \text{-})\ \mathtt{group\ by}\ \mathbf{t}')$ in $\mathcal{P}$ with grouping variables $\mathbf{x}$ and each grounding σ of the nonlocal variables in β, we have $\mathcal{P} \models \beta\sigma$ if and only if there is an integer $k_\beta \in \mathbb{Z}$ such that $\mathcal{P}' \models agg_\beta(\mathbf{x}\sigma, k_\beta) \land (s\sigma \lhd k_\beta)$. For this, in turn, it suffices to show that, for each ground aggregate atom $\beta = (s \lhd \mathsf{agg}\, C(\mathbf{t}, \text{-})\ \mathtt{group\ by}\ \mathbf{a})$ and each (object-finite) limit-closed interpretation $\mathcal{I}$ over predicates and objects of $\mathcal{P}$, we have $\mathcal{I} \models \beta$ if and only if there is $k_\beta \in \mathbb{Z}$ such that $s \lhd k_\beta$ and $\mathcal{D}_{order} \cup \mathcal{P}_{succ} \cup \mathcal{P}_\beta \cup \mathcal{I} \models agg_\beta(\mathbf{a}, k_\beta)$ (where, as in the proof of Theorem 4.26, we abuse notation and treat $\mathcal{I}$ as an 'infinite program'); in other words, we claim that there is a sample $\mathcal{K}$ of C in $\mathcal{I}$, with M the multiset of all ℓ for which $C(\mathbf{t}\sigma, \ell) \in \mathcal{K}$ for some σ, such that $s \lhd \mathsf{agg}(M)$ if and only if there is some $k_\beta \in \mathbb{Z}$ with $s \lhd k_\beta$ and $agg_\beta(\mathbf{a}, k_\beta) \in \mathcal{M}(\mathcal{D}_{\mathrm{ord}} \cup \mathcal{P}_{\mathrm{succ}} \cup \mathcal{P}_\beta \cup \mathcal{I})$. To see this, consider β and $\mathcal{I}$ as above. We have five cases, depending on the aggregate function agg.

If $\mathsf{agg} = \mathtt{count}$, then $\mathsf{agg}(M)$ does not depend on the choice of $\mathcal{K}$ and the claim follows by construction since agg_β is max and $\lhd \in \{\leq, <\}$.

If $\mathsf{agg} = \mathtt{max}$, we have two cases: either C, and hence agg_β, are max or both are min. In the first case, the claim follows as both $\mathtt{max}(\mathcal{K})$ and k_β range from $-\infty$ to the largest ℓ for which there is a substitution σ with $C(\mathbf{t}\sigma, \ell) \in \mathcal{I}$, if such ℓ exists, or to $+\infty$ otherwise. The second case is symmetric.

If $\mathsf{agg} = \mathtt{min}$, the claim follows analogously to the previous case.

If $\mathsf{agg} = \mathsf{sum}^+$, the claim follows analogously to the case of max, except for the following: if C is max, then $\mathsf{sum}^+(\mathcal{K})$ and k_β range from 0 to the sum of all positive ℓ with $C(\mathbf{t}\sigma, \ell) \in \mathcal{I}$ for some σ when this sum exists (i.e., is finite), and to $+\infty$ otherwise; if C is min, both numbers range from the sum of all such negative ℓ when the sum exists, or from $-\infty$ otherwise, to 0.

If $\mathsf{agg} = \mathsf{sum}^-$, the claim follows analogously to the previous case. □

The immediate corollary of Theorem 7.8 is that the upper bounds of Theorems 4.24 and 4.26 extend to $Datalog_{\mathbb{Z}}^{\mathsf{agg}}$; note that the data complexity bounds transfer because, according to the construction in the proof of Theorem 7.8, all facts in a limit $Datalog_{\mathbb{Z}}^{\mathsf{agg}}$ program $\mathcal{P}$ are copied to its translation $\mathcal{P}'$ without influencing the rest of $\mathcal{P}'$. These upper bounds are tight by Theorems 5.4 and 5.10 since each limit-linear $Datalog_{\mathbb{Z}}$ program is also a limit-linear $Datalog_{\mathbb{Z}}^{\mathsf{agg}}$ program.

Corollary 7.9. *The fact entailment problem is*

1. *in* coNEXP *in combined and in* coNP *in data complexity for semi-positive limit-linear* $Datalog_{\mathbb{Z}}^{\mathsf{agg}}$ *programs, and*
2. *in* Δ_2^{EXP} *in combined and in* Δ_2^{P} *in data complexity for limit-linear* $Datalog_{\mathbb{Z}}^{\mathsf{agg}}$ *programs.*

Finally, note that the results of Section 6 straightforwardly extend to $Datalog_{\mathbb{Z}}^{\mathsf{agg}}$. In particular, a limit-linear $Datalog_{\mathbb{Z}}^{\mathsf{agg}}$ program $\mathcal{P}$ is *stable* or *type-consistent* if so is its $Datalog_{\mathbb{Z}}$ translation $\mathcal{P}'$; Theorems 6.12 and 6.18 (including tractability in data complexity) then immediately extend to $Datalog_{\mathbb{Z}}^{\mathsf{agg}}$. Proposition 6.16 also extends as the translation can be done in logarithmic space.

8 RELATED WORK

In this section, we compare limit $Datalog_{\mathbb{Z}}$ with other formalisms underpinning rule-based systems for data analysis. We first focus on the 'monotonic programs' proposed by Ross and Sagiv [48], which are the closest to our work; afterwards, we discuss other approaches in a less formal manner.

8.1 Ross and Sagiv's Monotonic Programs

We begin by defining the formalism of Ross and Sagiv [48]. To facilitate comparison with our language, we adapt some of their original notation. To avoid excessive notation, we consider only an essential core of the formalism and leave the discussion of additional features, such as negation and default values, for the end of the section. Finally, to avoid ambiguity, we rename some of their notions by prepending the prefix 'rs-' to their name (e.g., we speak of rs-atoms and rs-programs).

Rs-Programs. A *cost domain* is a complete lattice—that is, a partially ordered set having a supremum and an infimum for each subset. Each cost domain $(D, \sqsubseteq)$ with a carrier set D and partial order $\sqsubseteq$ is associated with built-in functions $D^v \to D$, for $v \in \mathbb{N}$, and aggregate functions $\mathbb{N}^D \to D$, where $\mathbb{N}^D$ is the set of all multisets over D. The cost domains relevant for our comparison are given next, where $+$, $-$, and $\times$ are binary functions with the usual semantics (extended to $+\infty$ and $-\infty$ in a way yet to be discussed), and all aggregate functions also have the usual semantics:

- $(\mathbb{Z} \cup \{-\infty, +\infty\}, \leq)$ with built-in functions $+$, $-$ and $\times$, and aggregate functions `count` and `max`;
- $(\mathbb{Z} \cup \{-\infty, +\infty\}, \geq)$ with built-in functions $+$, $-$ and $\times$, and aggregate functions `count` and `min`;
- $(\mathbb{N} \cup \{+\infty\}, \leq)$ with built-in functions $+$ and $\times$, and aggregate functions `count` and `sum`.

We consider a vocabulary consisting of *rs-constants*, *rs-functions*, and *rs-predicates*, which are as usual except the following: rs-constants include $\mathbb{Z} \cup \{-\infty, +\infty\}$, rs-functions are precisely the built-in functions of the cost domains, and rs-predicates include the binary equality predicate $\equiv$ and, for each cost domain, the binary *built-in* rs-predicates $\sqsubseteq$, $\sqsubset$ denoting the partial order in this cost domain and its strict version, respectively, as well as *cost* rs-predicates with positive arities associated with the cost domain. The last position of a cost rs-predicate is called *cost position*. An *rs-atom* is one of the following:

- a *standard* rs-atom of the form $A(\mathbf{t})$, where A is a v-ary non-built-in rs-predicate and $\mathbf{t}$ is a v-tuple of rs-constants and variables such that, if A is a cost rs-predicate and the last term in $\mathbf{t}$ is an rs-constant, then this rs-constant belongs to the cost domain associated with A;
- a *built-in* rs-atom of the form $(s_1 \equiv s_2)$, $(s_1 \sqsubseteq s_2)$, or $(s_1 \sqsubset s_2)$, where $\sqsubseteq$ and $\sqsubset$ are the built-in predicates of a cost domain $(D, \sqsubseteq)$ and s_1, s_2 are numeric terms constructed from elements of D, variables, and the built-in functions associated with $(D, \sqsubseteq)$;
- an *aggregate* rs-atom of the form $(s \overset{\mathsf{rs}}{=} \mathsf{agg}\, C(\mathbf{t}, \text{-})\ \texttt{group by}\ \mathbf{t}')$, where C is a v-ary cost rs-predicate, agg is an aggregate function associated with the cost domain $(D, \sqsubseteq)$ of C, the *aggregate rs-term* s is a variable or an rs-constant from D, $\mathbf{t}$ is a $(v-1)$-tuple of rs-constants and variables, tuple $\mathbf{t}'$ of *grouping rs-terms* is a subset of the set of terms in $\mathbf{t}$, and - is a special symbol (which is added just to match the arity of C); the variables in $\mathbf{t}$ that are not in $\mathbf{t}'$ are *local* and should be different from s.

An *rs-rule* is an expression of the form $\psi \to \beta$, where the *body* ψ is a conjunction of rs-atoms such that all local variables of each aggregate rs-atom appear only in this rs-atom, and the *head*

β is a standard rs-atom. An rs-rule is *range-restricted* if each of its nonlocal variables x is *quasi-limited*—that is, appears in the body in a standard rs-atom, in an equality built-in rs-atom $(x \equiv s)$ such that all variables in s are quasi-limited, or in an aggregate rs-atom as the aggregate rs-term. An *rs-program* is a finite set of range-restricted rs-rules. An *rs-fact* is an rs-rule whose body is empty; usually, we do not distinguish between rs-facts and the rs-atoms in their heads.

An *rs-interpretation* is a set of rs-facts that does not contain two rs-facts that differ only in their cost positions (thus, Ross and Sagiv's notion of interpretations is similar to our pseudointerpretations). The partial orders of the cost domains extend to a partial order on rs-interpretations: $\mathcal{H}_1 \sqsubseteq_{\text{rs}} \mathcal{H}_2$ for two rs-interpretations $\mathcal{H}_1$ and $\mathcal{H}_2$ if, for each rs-fact $\delta_1 \in \mathcal{H}_1$, there is an rs-fact $\delta_2 \in \mathcal{H}_2$ such that $\delta_1 \sqsubseteq_{\text{rs}} \delta_2$—that is, such that either $\delta_1 = \delta_2$ or $\delta_1 = C(\mathbf{a}, k_1)$ and $\delta_2 = C(\mathbf{a}, k_2)$ where C is a cost rs-predicate and $k_1 \sqsubseteq k_2$, for $\sqsubseteq$ the partial order of the cost domain of C. In fact, $\sqsubseteq_{\text{rs}}$ induces a complete lattice on rs-interpretations [48, Theorem 3.1].

An rs-atom (or rs-rule) is *ground* if all of its variables are local (which implies that the head of an rs-fact is a ground rs-atom). An rs-interpretation $\mathcal{H}$ *satisfies* a ground rs-atom β, written $\mathcal{H} \models_{\text{rs}} \beta$, if

- β is a standard rs-atom that is contained in $\mathcal{H}$;
- β is a built-in rs-atom that evaluates to true under the semantics of the cost domain and the standard semantics of equality;
- β is an aggregate rs-atom $(k \overset{\text{rs}}{=} \mathsf{agg}\, C(\mathbf{t}, \text{-})$ $\mathsf{group\ by}$ $\mathbf{a})$ such that $k = \mathsf{agg}(M)$, for M the multiset of all integers ℓ having distinct substitutions σ of the local variables in $\mathbf{t}$ such that $C(\mathbf{t}\sigma, \ell) \in \mathcal{H}$.

An rs-interpretation $\mathcal{H}$ *satisfies* a conjunction of ground rs-atoms ψ, written $\mathcal{H} \models_{\text{rs}} \psi$, if $\mathcal{H}$ satisfies each rs-atom in ψ. The *immediate consequence* $\mathcal{R}_Q(\mathcal{H})$ of an rs-interpretation $\mathcal{H}$ with respect to an rs-program Q is the set of all rs-facts δ for which there are ground instances $\psi \to \delta$ of rules in Q such that $\mathcal{H} \models_{\text{rs}} \psi$. An rs-program Q is *cost-consistent* if, for each rs-interpretation $\mathcal{H}$, $\mathcal{R}_Q(\mathcal{H})$ is also an rs-interpretation. A cost-consistent rs-program Q is *rs-monotonic* if $\mathcal{R}_Q(\mathcal{H}_1) \sqsubseteq_{\text{rs}} \mathcal{R}_Q(\mathcal{H}_2)$ for each two rs-interpretations $\mathcal{H}_1$ and $\mathcal{H}_2$ such that $\mathcal{H}_1 \sqsubseteq_{\text{rs}} \mathcal{H}_2$.

Example 8.1. Not every rs-program is cost-consistent. For instance, consider the following rs-program Q, where C is a cost rs-predicate with associated cost domain $(\mathbb{Z} \cup \{-\infty, +\infty\}, \leq)$.

$$\to C(0) \qquad\qquad C(m) \wedge (n \equiv m + 1) \to C(n)$$

Rs-program Q is not cost-consistent: for $\mathcal{J} = \{C(0)\}$, the set $\mathcal{R}_Q(\mathcal{J}) = \{C(0), C(1)\}$ is not an rs-interpretation.

Similarly, not every cost-consistent rs-program is rs-monotonic. For instance, the following cost-consistent rs-program, for C as above, is not rs-monotonic.

$$C(m) \wedge (n \equiv -m) \to C(n)$$

The main property of rs-monotonic rs-programs is that each such program Q has a unique minimal model $\mathcal{L}(Q)$ with respect to $\sqsubseteq_{\text{rs}}$, which is the least fixpoint of the immediate consequence operator $\mathcal{R}_Q$ [48, Corollary 3.5]. An rs-program Q *entails* an rs-fact δ, written $Q \models_{\text{rs}} \delta$, if $\delta \in \mathcal{L}(Q)$.

Checking cost consistency and rs-monotonicity of an rs-program can both be shown undecidable in general. Hence, Ross and Sagiv define sufficient conditions for both cost consistency and rs-monotonicity, which are, however, outside the scope of this summary. Nonetheless, knowing that an rs-program is rs-monotonic and thus has a least model does not guarantee decidability of fact entailment; in fact, an argument similar to the one in the proof of Theorem 3.6 can be used to show undecidability of fact entailment for rs-monotonic rs-programs with the cost domain $(\mathbb{Z} \cup \{-\infty, +\infty\}, \leq)$.

Expressibility by Limit Programs. We next show that a core class of rs-monotonic rs-programs can be captured by our limit $\mathit{Datalog}_{\mathbb{Z}}^{\mathsf{agg}}$ programs. We start by formally defining this class, after which we give a translation from rs-programs into the class to positive limit $\mathit{Datalog}_{\mathbb{Z}}^{\mathsf{agg}}$ programs.

Observe that, unlike limit $\mathit{Datalog}_{\mathbb{Z}}$ programs, rs-programs do not make a strict distinction between elements of cost domains and other rs-constants. To overcome this mismatch, we concentrate on *strict* rs-programs—that is, rs-programs satisfying the following restrictions:

- elements of cost domains do not appear in noncost positions;
- each variable of an rs-rule is either a noncost variable or a cost variable, where a variable is *noncost* if it occurs only in noncost positions of non-built-in rs-predicates, and it is a *cost* variable if it occurs only in cost positions and in built-in rs-atoms.

These restrictions ensure that all rs-terms can be partitioned in a similar way to how terms in $\mathit{Datalog}_{\mathbb{Z}}$ programs are partitioned into object and numeric terms.

Furthermore, we restrict our attention to *predicate-nonrepeating* rs-programs—that is, rs-programs in each of whose rules each pair of distinct rs-atoms in the rule body have distinct predicates. We discuss rs-programs with repeated predicates in rule bodies in the end of this section.

Finally, we only present the translation for programs over cost domains $(\mathbb{Z} \cup \{-\infty, +\infty\}, \leq)$ and $(\mathbb{N} \cup \{+\infty\}, \leq)$; as we will discuss later on, the cost domain $(\mathbb{Z} \cup \{-\infty, +\infty\}, \geq)$ can be easily simulated by the other two.

Before proceeding to the translation, it remains to complete the definitions of the remaining two cost domains by fixing the behaviour of arithmetic functions $+$, $-$, and $\times$ on elements $-\infty$ and $+\infty$. Ross and Sagiv do not specify the semantics of terms with $-\infty$ or $+\infty$; in particular, it is not clear from [48] how to interpret terms containing both $-\infty$ and $+\infty$, such as $(-\infty) + (+\infty)$. One of the ways would be to treat $-\infty$ and $+\infty$ symmetrically, in particular, having $(-\infty) + (+\infty) = 0$. This, however, would make addition not associative:

$$((+\infty) + (+\infty)) + (-\infty) = (+\infty) + (-\infty) = 0 \neq (+\infty) = (+\infty) + 0 = (+\infty) + ((+\infty) + (-\infty)).$$

Thus, to simplify our translation, we adopt a different, nonsymmetric extension of arithmetic to $-\infty$ and $+\infty$. Specifically, we assume that the expression $\ell_1 * \ell_2$ evaluates as follows, for every $* \in \{+, \times\}$ and $\ell_1, \ell_2 \in \mathbb{Z} \cup \{-\infty, +\infty\}$ such that $\{\ell_1, \ell_2\} \cap \{-\infty, +\infty\} \neq \emptyset$:

$$\ell_1 * \ell_2 = \begin{cases} +\infty & \text{if either } \ell_1 = \ell_2 = -\infty \text{ and } * \text{ is } \times, \text{ or at least one of } \ell_1, \ell_2 \text{ is } +\infty, \\ -\infty & \text{otherwise;} \end{cases}$$

furthermore, $\ell_1 - \ell_2 = \ell_1 + (-\ell_2)$ for all such ℓ_1 and ℓ_2. This extension has the following convenient property, which can be proved by a straightforward induction on the structure of rs-terms.

Proposition 8.2. *Each ground rs-term t mentioning $-\infty$ or $+\infty$ evaluates either to $-\infty$ or to $+\infty$; furthermore, the result of the evaluation does not depend on the integers in t—that is, the value of t equals the value of each term obtained from t by substituting any integer in t with any other integer.*

We are ready to define our translation from predicate-nonrepeating strict rs-programs to positive limit $\mathit{Datalog}_{\mathbb{Z}}^{\mathsf{agg}}$ programs. The main idea is to represent rs-facts $C(\mathbf{a}, -\infty)$, $C(\mathbf{a}, k)$ for $k \in \mathbb{Z}$, and $C(\mathbf{a}, +\infty)$ over a cost rs-predicate C by pseudofacts $C^{-\infty}(\mathbf{a}, 0)$, $C'(\mathbf{a}, k)$, and $C'(\mathbf{a}, \infty)$, respectively, where $C^{-\infty}$ and C' are fresh max predicates corresponding to C, and then to 'manually' simulate assignments of $-\infty$ and $+\infty$ to variables (predicate $C^{-\infty}$ needs to be max and not object only so we can count over it; the choice of 0 in $C^{-\infty}(\mathbf{a}, 0)$ is thus inessential).

Definition 8.3. Given a strict rs-program Q, let Q^{sum} be the strict rs-program obtained from Q by adding an rs-atom $(0 \leq m)$ to the body of each rs-rule in Q whose head has the form $C(\mathbf{t}, m)$, where C is a cost rs-predicate with cost domain $(\mathbb{N} \cup \{+\infty\}, \leq)$.

For each noncost rs-predicate A in $\mathcal{Q}$, let A' be an object predicate of the same arity as A, and, for each cost rs-predicate C in $\mathcal{Q}$, let $C^{-\infty}$ and C' be max predicates of the same arity as C. For each rs-rule π in $\mathcal{Q}^{\mathsf{sum}}$ and each pair $\mathbf{m}$, $\mathbf{n}$ of disjoint subsets of the cost variables in π, let $\rho_\pi^{\mathbf{m},\mathbf{n}}$ be the $Datalog_{\mathbb{Z}}^{\mathsf{agg}}$ rule obtained from π by first replacing each variable in $\mathbf{m}$ and $\mathbf{n}$ by $-\infty$ and $+\infty$, respectively, and then replacing each rs-atom β, both in the body and in the head, by the atom α_β, defined as follows:

- if β is $A(\mathbf{t})$ with A a noncost rs-predicate, then α_β is $A'(\mathbf{t})$;
- if β is $C(\mathbf{t}, s)$ with C a cost rs-predicate, then
  - if $s = -\infty$ then α_β is $C^{-\infty}(\mathbf{t}, 0)$,
  - if $s \notin \{-\infty, +\infty\}$ then α_β is $C'(\mathbf{t}, s)$,
  - if $s = +\infty$ then α_β is $C'(\mathbf{t}, \infty)$;
- if β is a built-in rs-atom, then
  - if β involves neither $-\infty$ nor $+\infty$ then $\alpha_\beta = \beta$,
  - if β involves $-\infty$ or $+\infty$ and evaluates to true (see Proposition 8.2) then α_β is $(0 \leq 0)$,
  - if β involves $-\infty$ or $+\infty$ and evaluates to false then α_β is $(0 < 0)$;
- if β is an aggregate rs-atom $(s \overset{\mathsf{rs}}{=} \mathsf{agg}\, C(\mathbf{t}, \text{-})\ \mathsf{group\ by}\ \mathbf{t}')$, then
  - if agg is aggregate function count, and $s \notin \{-\infty, +\infty\}$ or $s \in \{-\infty, +\infty\}$, then α_β is the atom $(s \leq \mathsf{count}\, C^{-\infty}(\mathbf{t}, \text{-})\ \mathsf{group\ by}\ \mathbf{t}')$ or $(0 < 0)$, respectively,
  - if agg is aggregate function max, and $s = -\infty$, $s \notin \{-\infty, +\infty\}$ or $s = +\infty$, then α_β is the atom $(0 \leq 0)$, $C'(\mathbf{t}, s)$ or $C'(\mathbf{t}, \infty)$, respectively,
  - if agg is aggregate function sum, and $s = -\infty$, $s \notin \{-\infty, +\infty\}$ or $s = +\infty$, then α_β is the atom $(0 < 0)$, $(s \leq \mathsf{sum}^+\, C'(\mathbf{t}, \text{-})\ \mathsf{group\ by}\ \mathbf{t}')$ or $C'(\mathbf{t}, \infty)$, respectively.

The *translation* $\mathcal{P}(\mathcal{Q})$ of $\mathcal{Q}$ is the positive $Datalog_{\mathbb{Z}}^{\mathsf{agg}}$ program containing a rule $\rho_\pi^{\mathbf{m},\mathbf{n}}$ for each rs-rule π in $\mathcal{Q}^{\mathsf{sum}}$ and each pair $\mathbf{m}$, $\mathbf{n}$ of disjoint subsets of the cost variables in π, as well as the following rule for each cost rs-predicate C, where $\mathbf{x}$ is a tuple of distinct object variables of appropriate arity.

$$C'(\mathbf{x}, m) \to C^{-\infty}(\mathbf{x}, 0) \tag{83}$$

It is immediate that this definition is syntactically correct in the sense that $\mathcal{P}(\mathcal{Q})$ is indeed a positive limit $Datalog_{\mathbb{Z}}^{\mathsf{agg}}$ program. Note also that $\mathcal{Q}$ and $\mathcal{Q}^{\mathsf{sum}}$ are equivalent; the atoms $(0 \leq m)$ merely preclude negative numbers to appear in the numeric atoms over C' when C has the cost domain $(\mathbb{N} \cup \{+\infty\}, \leq)$, which corresponds to the semantics of C.

The following theorem establishes correctness of the translation, where, for uniformity, we extend the entailment relation to pseudofacts and write, for $\mathcal{P}$ a $Datalog_{\mathbb{Z}}^{\mathsf{agg}}$ program and γ a pseudofact, $\mathcal{P} \models \gamma$ if $\mathcal{M}(\mathcal{P}) \models \gamma$ (note that until now we have used such notation only for facts).

Theorem 8.4. *The following holds for each predicate-nonrepeating strict rs-monotonic rs-program $\mathcal{Q}$ and each rs-fact δ:*

- *if δ is $A(\mathbf{a})$ with A a noncost rs-predicate, then $\mathcal{Q} \models_{\mathsf{rs}} \delta$ if and only if $\mathcal{P}(\mathcal{Q}) \models A'(\mathbf{a})$;*
- *if δ is $C(\mathbf{a}, \ell)$ with C a cost rs-predicate, then*
  - *if $\ell = -\infty$ then $\mathcal{Q} \models_{\mathsf{rs}} \delta$ if and only if $\mathcal{P}(\mathcal{Q}) \models C^{-\infty}(\mathbf{a}, 0)$ and $\mathcal{P}(\mathcal{Q}) \not\models C'(\mathbf{a}, k)$ for all $k \in \mathbb{Z}$,*
  - *if $\ell \in \mathbb{Z}$ then $\mathcal{Q} \models_{\mathsf{rs}} \delta$ if and only if $\mathcal{P}(\mathcal{Q}) \models \lceil C'(\mathbf{a}, \ell) \rceil$, and*
  - *if $\ell = +\infty$ then $\mathcal{Q} \models_{\mathsf{rs}} \delta$ if and only if $\mathcal{P}(\mathcal{Q}) \models C'(\mathbf{a}, \infty)$.*

The proof of this theorem is given in the appendix.

Theorem 8.4 establishes a connection between fact entailment for rs-programs and the techniques developed in this paper. In particular, note that all numeric terms with arithmetic functions in the $Datalog_{\mathbb{Z}}^{\mathsf{agg}}$ program $\mathcal{P}(\mathcal{Q})$ are inherited from the rs-program $\mathcal{Q}$. Hence, if all cost terms in $\mathcal{Q}$ are linear, then program $\mathcal{P}(\mathcal{Q})$ is limit-linear.

Proposition 8.5. *The rs-fact entailment problem for predicate-nonrepeating strict rs-monotonic rs-programs with linear cost terms is in* DEXP *in combined and in* DP *in data complexity; moreover, if the rs-fact is noncost or has the cost value* $+\infty$*, the problem is in* coNEXP *and in* coNP*, respectively.*

Proof. Consider an arbitrary predicate-nonrepeating strict rs-monotonic rs-program Q and the positive limit-linear $Datalog_{\mathbb{Z}}$ program $\mathcal{P}'$ obtained by first translating Q to the $Datalog_{\mathbb{Z}}^{\mathsf{agg}}$ program $\mathcal{P}(Q)$ and then translating $\mathcal{P}(Q)$ to $Datalog_{\mathbb{Z}}$ as in the proof of Theorem 7.8. By construction, the canonical OG-grounding $\mathcal{G}(\mathcal{P}')$ of $\mathcal{P}'$ possesses the same computational properties as canonical OG-groundings of positive LL-programs given in Lemma 4.3, for c the number of distinct rs-constants in Q and $u = \max_{\pi \in Q} [\![\pi]\!]$:

1. $\mathcal{G}(\mathcal{P}')$ can be computed in time polynomial in $c^u + \|Q\|$, and
2. $\max_{\rho \in \mathcal{G}(\mathcal{P}')} [\![\rho]\!]$ is linearly bounded in u.

Thus, we can once again use Algorithm 1, which decides fact entailment for positive LL-programs in coNP in data and in coNEXP in combined complexity (see Theorem 4.17), to decide fact entailment for $\mathcal{P}'$ within the same complexity bounds. Moreover, by Theorems 7.8 and 8.4 as well as Proposition 4.25, rs-fact entailment for Q boils down to fact entailment and, in some cases, fact nonentailment for $\mathcal{P}'$, which implies our claims. □

To the best of our knowledge, Proposition 8.5 provides the first decidability results for the formalism of Ross and Sagiv [48].

Discussion. The reduction in Theorem 8.4 makes several simplifying assumptions about Ross and Sagiv's formalism. In what follows, we discuss these assumptions in more detail.

First, Ross and Sagiv allow for stratified negation in rule bodies in essentially the same way as we do for $Datalog_{\mathbb{Z}}$ (it is also noted that any rule with nonstratified negation either never applies or violates rs-monotonicity of the program strata). We can extend our encoding in Definition 8.3 in a straightforward way to capture predicate-nonrepeating strict rs-programs with stratified negation and rs-monotonic strata. This would yield Δ_2^{EXP} upper bound for the rs-fact entailment problem in combined complexity and Δ_2^{P} upper bound in data complexity.

Second, entailment is only defined for the class of rs-monotonic—and hence cost-consistent—rs-programs, which do not include simple programs such as the ones in Example 8.1. To somewhat alleviate the restrictions caused by the cost consistency requirement, Ross and Sagiv allow *default value declarations*, which can be used to assign 'default' numeric values to predicates that can be 'overridden' by rules. For instance, the intention behind the first program in Example 8.1 can be realised by the following rs-program where, intuitively, the declaration on the left asserts that the value of C is initially 0, but is disregarded if some rule yields a different value.

$$\textit{declare default } C(0) \qquad\qquad C(m) \wedge (n \equiv m + 1) \rightarrow C(n)$$

Default value declarations can be easily simulated in limit $Datalog_{\mathbb{Z}}$ by extending the encoding in Definition 8.3: for instance, to simulate a declaration *declare default* $C(\mathbf{t}, 0)$, it suffices to iterate through all possible instances of $\mathbf{t}$ over the objects mentioned in the program (using a predicate *succ* defined as in the proof of Theorem 5.3) and assign 0 to C for each such instance.

Third, we only present a translation for the cost domains $(\mathbb{Z} \cup \{-\infty, +\infty\}, \leq)$ and $(\mathbb{N} \cup \{+\infty\}, \leq)$, leaving aside the cost domain $(\mathbb{Z} \cup \{-\infty, +\infty\}, \geq)$; thus, we intuitively focus on 'homogeneous' rs-programs. This is, however, not a major restriction since we can easily transform any rs-program over all three cost domains into an rs-program over the first two domains using a technique similar to that in Proposition 3.1. Furthermore, Ross and Sagiv also consider cost domains over real numbers, Booleans, and finite sets, which are outside the scope of our formalism.

Finally, we assume that rs-programs are predicate-nonrepeating, and the backward direction of Theorem 8.4 relies on this property. On the one hand, note that each rs-program can be transformed to a predicate-nonrepeating rs-program by introducing extra rs-predicates: for example, an rs-program consisting of a single rs-rule $C(3) \wedge C(4) \rightarrow A$ can be rewritten as the following rs-program.

$$C(3) \wedge C'(4) \rightarrow A \qquad\qquad C(m) \rightarrow C'(m)$$

On the other hand, while the original rs-program is vacuously rs-monotonic, the rewritten rs-program is not (to see this, consider rs-interpretations $\mathcal{H}_1 = \{C(3), C'(4)\}$ and $\mathcal{H}_2 = \{C(3), C'(5)\}$).

8.2 Other Related Work

Mazuran et al. [39] propose Datalog$^{\mathsf{FS}}$, which extends usual stratified Datalog with so-called *frequency support goals* and provides the formal underpinning for the DeALS system [62, 63]. The semantics of Datalog$^{\mathsf{FS}}$ is given by translation to logic programs with arithmetic, tuples, and lists. Mazuran et al. show that Datalog$^{\mathsf{FS}}$ can express various analytical queries involving arithmetic and aggregation. Frequency support goals in Datalog$^{\mathsf{FS}}$ provide the ability to count the number of distinct variable assignments satisfying a given goal atom; their semantics is similar to that of max predicates in limit $\mathit{Datalog}_{\mathbb{Z}}$ in that, whenever a frequency support goal holds for a given number, it also holds for all smaller numbers. Moreover, Mazuran et al. discuss an optimisation strategy for Datalog$^{\mathsf{FS}}$ programs, called *max-based optimisation*, which bears some resemblance to our stability condition. In contrast to LL-programs, and similarly to unrestricted $\mathit{Datalog}_{\mathbb{Z}}$, fact entailment in Datalog$^{\mathsf{FS}}$ is undecidable, and no practical decidable fragment has been identified, to the best of our knowledge. Identifying such fragments could potentially be accomplished by transferring some of the ideas in our work to the formalism of Mazuran et al. [39].

Zaniolo et al. [63] identify an extension of Datalog$_{1\mathrm{S}}$, a language proposed by Chomicki and Imielinski [10] in the context of temporal reasoning, with `min` and `max` aggregates, where each program can be rewritten into ordinary Datalog$_{1\mathrm{S}}$ with stratified negation as failure. Zaniolo et al. [63] show that the proposed language is powerful enough to encode some optimisation problems such as shortest paths from a given source node a_s, as illustrated by the following program in their language [63].

$$\mathit{edge}(a_s, x, m) \wedge (m \geq 0) \rightarrow \mathit{dist}(x, m) \tag{84}$$

$$\mathit{dist}(x, m) \wedge \mathit{edge}(x, y, n) \wedge (n \geq 0) \wedge (m' \doteq m + n) \rightarrow \mathit{dist}(y, m') \tag{85}$$

$$\mathit{dist}(x, m) \wedge \mathtt{min}((x), (m)) \rightarrow \mathit{s\text{-}dist}(x, m) \tag{86}$$

In this program, *edge* is an EDB predicate encoding an input graph as in the program of rules (1)–(2) in the introduction, *dist* and *s-dist* are ordinary (i.e., without any limit semantics) IDB predicates, and `min` is a special predicate whose extension is obtained from the extension of *dist* by grouping according to the first argument and then selecting the minimum value for each group. The intended meaning of this special predicate is then formalised by rewriting rule (86) as the following rules with negation.

$$\mathit{dist}(x, m) \wedge \mathit{dist}(x, n) \wedge (n < m) \rightarrow \mathit{lesser}(x, m) \tag{87}$$

$$\mathit{dist}(x, m) \wedge \mathtt{not}\ \mathit{lesser}(x, m) \rightarrow \mathit{s\text{-}dist}(x, m) \tag{88}$$

It is worth noting that rule (85) is not in Datalog$_{1\mathrm{S}}$ since it contains the term $m + n$ involving addition of two variables, whereas Datalog$_{1\mathrm{S}}$ only allows addition of a constant represented in unary (allowing arbitrary addition of variables would make Datalog$_{1\mathrm{S}}$ undecidable by Theorem 2.2). Since *edge* is an EDB predicate, rule (85) can, however, be rewritten into a set of Datalog$_{1\mathrm{S}}$ rules by grounding variable n against the input graph.

When compared to ours, the approach by Zaniolo et al. [63] has several disadvantages. First, bottom-up evaluation on the aforementioned program will not terminate if the input graph is cyclic, and Zaniolo et al. [63] do not provide a mechanism for ensuring termination in such cases; in contrast, the type-consistent and positive LL-program (1)–(2) encodes the problem and can be evaluated in polynomial time in the size of data for any input graph. Second, when evaluating special predicates they rely on rewriting via negation, whereas our formulations of problems such as shortest path are negation-free. Finally, our approach offers more favourable computational properties in general; indeed, as we have shown, reasoning in limit-linear $Datalog_{\mathbb{Z}}$ is in Δ_2^{P} in data complexity, whereas the computational properties for the language of Zaniolo et al. [63] is unknown (and even in positive Datalog_{1S} reasoning is PSPACE-hard in data complexity [10]).

The primary motivation for the work of Ross and Sagiv [48], Mazuran et al. [39], and Zaniolo et al. [63] was to provide a generic, uncontentious semantics for aggregation in recursive rules. Earlier research on the topic yielded solutions for restricted classes of programs that are either subject to strong monotonicity assumptions, only use *min* and *max* aggregate functions, or can be expressed as optimisation problems on closed semirings [11, 20, 42, 56]. Furthermore, well-founded and stable model semantics for aggregation in expressive logic programming languages with unrestricted arithmetic and undecidable fact entailment were proposed by Kemp and Stuckey [35] and further extended later on [6, 46, 57, 59]. Decidable recursive languages with aggregation—and hence with a restricted form of arithmetic—were studied in the context of answer set programming (ASP) by Faber et al. [19]. In contrast to limit-linear $Datalog_{\mathbb{Z}}$, however, Faber et al. achieve decidability by disallowing *numeric value invention*—that is, deriving any facts with numeric arguments that do not appear in the input program.

$Datalog_{\mathbb{Z}}$ is closely related to constraint logic programming (CLP) [3, 13]. Although a number of decidable CLP languages have been identified, including the formalism of Cox et al. [13], none of them allow for recursive numeric value invention, same as the language of Faber et al. [19].

Our formalism is also related to query languages studied in the field of constraint databases [24, 25, 33]. In contrast to our work, which is motivated by optimisation problems, the emphasis in constraint databases is on finitely representing relations with temporal and spatial data; such relations are naturally infinite, and constraint database formalisms represent them in a finite way using quantifier-free formulas over a suitable background structure (e.g., the rational numbers equipped with comparison and addition). It is well-known, however, that adding recursion to constraint query languages equipped with addition immediately leads to undecidability (e.g., see [25]). Decidability results in the context of recursive queries over constraint databases mostly apply to query languages without arithmetic operators. For instance, Toman and Chomicki [58] propose an extension of usual Datalog with comparison operators over the integers and periodicity constraints (but no arithmetic operators); such periodicity constraints are especially useful in the context of temporal constraint databases, where the proposed extended Datalog rules can be used to identify events happening with certain periodicity (e.g., flights between two cities departing twice a week, or buses running every ten minutes).

Finally, while this paper was under review, we studied the complexity and expressive power of limit Datalog programs extended with disjunction in the heads of rules and non-monotonic (non-stratified) negation under the stable model semantics [30]. We showed that allowing for unrestricted use of negation leads to undecidability of reasoning, while decidability can be restored by stratifying the use of negation over predicates carrying numeric values: the resulting language is Π_2^{EXP}complete in combined complexity and captures Π_2^{P} over ordered structures. In that paper, we have also studied several fragments of this language: we show that the complexity and expressive power of the full language are already reached for disjunction-free programs, while the semi-positive disjunctive programs are coNEXP-complete and capture coNP.

9 CONCLUSION AND FUTURE WORK

Using Datalog with stratified negation and integer arithmetic as a foundation, we have presented in this paper the formalism of limit *Datalog*$_{\mathbb{Z}}$ programs, which is flexible and powerful enough to capture in a natural way a wide range of data analysis tasks. We have then identified a number of classes of limit *Datalog*$_{\mathbb{Z}}$ programs with decidable fact entailment and established their computational complexity and expressive power. The complexity of fact entailment for these fragments ranges from EXP in combined and P in data complexity in the case of stable programs, to Δ_2^{EXP} in combined and Δ_2^{P} in data complexity in the case of unrestricted limit-linear programs. Furthermore, the language of limit-linear programs captures the complexity class Δ_2^{P} over ordered datasets in the sense of descriptive complexity, thus providing (to the best of our knowledge) the first logical characterisation of Δ_2^{P}.

We see many avenues for future work. For instance, from a theoretical perspective it would be very interesting to study the natural extension of limit-linear programs with disjunction in rule heads and unstratified negation under the well-founded semantics, thus complementing the resent results for stable model semantics [30]. Another direction for future research is to study the ability of limit *Datalog*$_{\mathbb{Z}}$ to actually solve various optimisation problems (rather than the associated decision problems) by identifying the relationship of the language with known optimisation complexity classes. We also believe that our languages, especially the fragments with tractable data complexity, are practical and can be exploited in data-intensive applications; therefore, we are planning to implement our materialisation-based algorithms as an extension of the RDFox Datalog engine [41], which already provides support for stratified aggregation and negation as failure.

ACKNOWLEDGMENTS

This work was supported by the EPSRC projects DBOnto, MaSI^3, and ED^3.

We thank Christoph Haase for discussing with us his results on Presburger arithmetic and semi-linear sets as well as for providing a proof for Lemma 4.15 based on his work. This paper has also benefited significantly from discussions with Michael Benedikt.

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases.* Addison-Wesley.
[2] Peter Alvaro, Tyson Condie, Neil Conway, Khaled Elmeleegy, Joseph M. Hellerstein, and Russell Sears. 2010. BOOM Analytics: Exploring Data-Centric, Declarative Programming for the Cloud. In *EuroSys*, Christine Morin and Gilles Muller (Eds.). ACM, 223–236. https://doi.org/10.1145/1755913.1755937
[3] Krzysztof R. Apt. 2003. *Principles of constraint programming.* Cambridge University Press.
[4] Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. 1988. Towards a Theory of Declarative Knowledge. In *Foundations of Deductive Databases and Logic Programming*, Jack Minker (Ed.). Morgan Kaufmann, 89–148.
[5] Catriel Beeri, Shamim A. Naqvi, Oded Shmueli, and Shalom Tsur. 1991. Set Constructors in a Logic Database Language. *J. Log. Program.* 10, 3&4 (1991), 181–232. https://doi.org/10.1016/0743-1066(91)90036-O
[6] Catriel Beeri, Raghu Ramakrishnan, Divesh Srivastava, and S. Sudarshan. 1992. The Valid Model Semantics for Logic Programs. In *PODS*. ACM Press, 91–104. https://doi.org/10.1145/137097.137115
[7] Leonard Berman. 1980. The Complexitiy of Logical Theories. *Theory Comput. Sci.* 11 (1980), 71–77. https://doi.org/10.1016/0304-3975(80)90037-7
[8] Brian Chin, Daniel von Dincklage, Vuk Ercegovac, Peter Hawkins, Mark S. Miller, Franz Josef Och, Christopher Olston, and Fernando Pereira. 2015. Yedalog: Exploring Knowledge at Scale. In *SNAPL (LIPIcs, Vol. 32)*, Thomas Ball, Rastislav Bodík, Shriram Krishnamurthi, Benjamin S. Lerner, and Greg Morrisett (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 63–78. https://doi.org/10.4230/LIPIcs.SNAPL.2015.63
[9] Dmitry Chistikov and Christoph Haase. 2016. The Taming of the Semi-Linear Set. In *ICALP (LIPIcs, Vol. 55)*, Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi (Eds.). Schloss Dagstuhl - Leibniz-Zentrum

für Informatik, 128:1–128:13. https://doi.org/10.4230/LIPIcs.ICALP.2016.128

[10] Jan Chomicki and Tomasz Imielinski. 1988. Temporal Deductive Databases and Infinite Objects. In *PODS*, Chris Edmondson-Yurkanan and Mihalis Yannakakis (Eds.). ACM, 61–73. https://doi.org/10.1145/308386.308416

[11] Mariano P. Consens and Alberto O. Mendelzon. 1993. Low-Complexity Aggregation in GraphLog and Datalog. *Theory Comput. Sci.* 116, 1 (1993), 95–116. https://doi.org/10.1016/0304-3975(93)90221-E

[12] Patrick Cousot and Radhia Cousot. 1979. Constructive Versions of Tarski's Fixed Point Theorems. *Pac. J. Math.* 82, 1 (1979), 43–57. https://doi.org/10.2140/pjm.1979.82.43

[13] Jim Cox, Ken McAloon, and Carol Tretkoff. 1992. Computational Complexity and Constraint Logic Programming Languages. *Ann. Math. Artif. Intell.* 5, 2–4 (1992), 163–189. https://doi.org/10.1007/BF01543475

[14] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. 2001. Complexity and Expressive Power of Logic Programming. *ACM Comput. Surv.* 33, 3 (2001), 374–425. https://doi.org/10.1145/502807.502810

[15] Anuj Dawar. 1995. Generalized Quantifiers and Logical Reducibilities. *J. Log. Comput.* 5, 2 (1995), 213–226. https://doi.org/10.1093/logcom/5.2.213

[16] Jason Eisner and Nathaniel Wesley Filardo. 2011. Dyna: Extending Datalog for Modern AI. In *Datalog (LNCS, Vol. 6702)*, Oege de Moor, Georg Gottlob, Tim Furche, and Andrew Jon Sellers (Eds.). Springer, 181–220.

[17] Thomas Eiter, Georg Gottlob, and Heikki Mannila. 1997. Disjunctive Datalog. *ACM Trans. Database Syst.* 22, 3 (1997), 364–418. https://doi.org/10.1145/261124.261126

[18] Thomas Eiter, Georg Gottlob, and Helmut Veith. 1997. Modular Logic Programming and Generalized Quantifiers. In *LPNMR (LNCS (LNAI), Vol. 1265)*, Jürgen Dix, Ulrich Furbach, and Anil Nerode (Eds.). Springer, 290–309.

[19] Wolfgang Faber, Gerald Pfeifer, and Nicola Leone. 2011. Semantics and Complexity of Recursive Aggregates in Answer Set Programming. *Artif. Intell.* 175, 1 (2011), 278–298. https://doi.org/10.1016/j.artint.2010.04.002

[20] Sumit Ganguly, Sergio Greco, and Carlo Zaniolo. 1995. Extrema Predicates in Deductive Databases. *J. Comput. System Sci.* 51, 2 (1995), 244–259. https://doi.org/10.1006/jcss.1995.1064

[21] Georg Gottlob, Nicola Leone, and Helmut Veith. 1999. Succinctness as a Source of Complexity in Logical Formalisms. *Ann. Pure Appl. Log.* 97, 1-3 (1999), 231–260. https://doi.org/10.1016/S0168-0072(98)00057-8

[22] Erich Grädel. 1988. Subclasses of Presburger Arithmetic and the Polynomial-Time Hierarchy. *Theory Comput. Sci.* 56 (1988), 289–301. https://doi.org/10.1016/0304-3975(88)90136-3

[23] Erich Grädel. 2007. Finite Model Theory and Descriptive Complexity. In *Finite Model Theory and Its Applications*, Erich Grädel, Phokion G. Kolaitis, Leonid Libkin, Maarten Marx, Joel Spencer, Moshe Y. Vardi, Yde Venema, and Scott Weinstein (Eds.). Springer, 125–230. https://doi.org/10.1007/3-540-68804-8

[24] Erich Grädel and Stephan Kreutzer. 1999. Descriptive Complexity Theory for Constraint Databases. In *CSL (LNCS, Vol. 1683)*, Jörg Flum and Mario Rodríguez-Artalejo (Eds.). Springer, 67–81.

[25] Stéphane Grumbach and Jianwen Su. 1997. Queries with Arithmetical Constraints. *Theory Comput. Sci.* 173, 1 (1997), 151–181. https://doi.org/10.1016/S0304-3975(96)00194-6

[26] Christoph Haase. 2014. Subclasses of Presburger Arithmetic and the Weak EXP Hierarchy. In *CSL-LICS*, Thomas A. Henzinger and Dale Miller (Eds.). ACM, 47:1–47:10. https://doi.org/10.1145/2603088.2603092

[27] Lane A. Hemachandra. 1989. The Strong Exponential Hierarchy Collapses. *J. Comput. System Sci.* 39, 3 (1989), 299–322. https://doi.org/10.1016/0022-0000(89)90025-1

[28] Stefan Hougardy. 2010. The Floyd-Warshall Algorithm on Graphs with Negative Cycles. *Inform. Process. Lett.* 110, 8-9 (2010), 279–281. https://doi.org/10.1016/j.ipl.2010.02.001

[29] Neil Immerman. 1999. *Descriptive Complexity*. Springer.

[30] Mark Kaminski, Bernardo Cuenca Grau, Egor V. Kostylev, and Ian Horrocks. 2020. Complexity and Expressive Power of Disjunction and Negation in Limit Datalog. In *AAAI*. AAAI Press, 2862–2869. https://doi.org/10.1609/aaai.v34i03.5676

[31] Mark Kaminski, Bernardo Cuenca Grau, Egor V. Kostylev, Boris Motik, and Ian Horrocks. 2017. Foundations of Declarative Data Analysis Using Limit Datalog Programs. In *IJCAI*, Carles Sierra (Ed.). ijcai.org, 1123–1130. https://doi.org/10.24963/ijcai.2017/156

[32] Mark Kaminski, Bernardo Cuenca Grau, Egor V. Kostylev, Boris Motik, and Ian Horrocks. 2018. Stratified Negation in Limit Datalog Programs. In *IJCAI*, Jérôme Lang (Ed.). ijcai.org, 1875–1881. https://doi.org/10.24963/ijcai.2018/259

[33] Paris C. Kanellakis, Gabriel M. Kuper, and Peter Z. Revesz. 1995. Constraint Query Languages. *J. Comput. System Sci.* 51, 1 (1995), 26–52. https://doi.org/10.1006/jcss.1995.1051

[34] Ravi Kannan. 1987. Minkowski's Convex Body Theorem and Integer Programming. *Math. Oper. Res.* 12, 3 (1987), 415–440. https://doi.org/10.1287/moor.12.3.415

[35] David B. Kemp and Peter J. Stuckey. 1991. Semantics of Logic Programs with Aggregates. In *ISLP*, Vijay A. Saraswat and Kazunori Ueda (Eds.). MIT Press, 387–401.

[36] Richard E. Ladner and Nancy A. Lynch. 1976. Relativization of Questions About Log Space Computability. *Math. Systems Theory* 10 (1976), 19–32. https://doi.org/10.1007/BF01683260

[37] Boon Thau Loo, Tyson Condie, Minos N. Garofalakis, David E. Gay, Joseph M. Hellerstein, Petros Maniatis, Raghu Ramakrishnan, Timothy Roscoe, and Ion Stoica. 2009. Declarative Networking. *Commun. ACM* 52, 11 (2009), 87–95. https://doi.org/10.1145/1592761.1592785

[38] Volker Markl. 2014. Breaking the Chains: On Declarative Data Analysis and Data Independence in the Big Data Era. *PVLDB* 7, 13 (2014), 1730–1733. https://doi.org/10.14778/2733004.2733075

[39] Mirjana Mazuran, Edoardo Serra, and Carlo Zaniolo. 2013. Extending the Power of Datalog Recursion. *VLDB J.* 22, 4 (2013), 471–493. https://doi.org/10.1007/s00778-012-0299-1

[40] Robert R. Meyer. 1974. On the Existence of Optimal Solutions to Integer and Mixed-Integer Programming Problems. *Math. Program.* 7, 1 (1974), 223–235. https://doi.org/10.1007/BF01585518

[41] Boris Motik, Yavor Nenov, Robert Piro, Ian Horrocks, and Dan Olteanu. 2014. Parallel Materialisation of Datalog Programs in Centralised, Main-Memory RDF Systems. In *AAAI*, Carla E. Brodley and Peter Stone (Eds.). AAAI Press, 129–137.

[42] Inderpal Singh Mumick, Hamid Pirahesh, and Raghu Ramakrishnan. 1990. The Magic of Duplicates and Aggregates. In *VLDB*, Dennis McLeod, Ron Sacks-Davis, and Hans-Jörg Schek (Eds.). Morgan Kaufmann, 264–277.

[43] Christos H. Papadimitriou. 1981. On the Complexity of Integer Programming. *J. ACM* 28, 4 (1981), 765–768. https://doi.org/10.1145/322276.322287

[44] Christos H. Papadimitriou. 1994. *Computational complexity.* Addison-Wesley.

[45] Christos H. Papadimitriou and Mihalis Yannakakis. 1984. The Complexity of Facets (and Some Facets of Complexity). *J. Comput. System Sci.* 28, 2 (1984), 244–259. https://doi.org/10.1016/0022-0000(84)90068-0

[46] Nikolay Pelov, Marc Denecker, and Maurice Bruynooghe. 2007. Well-Founded and Stable Semantics of Logic Programs with Aggregates. *Theory Pract. Log. Program.* 7, 3 (2007), 301–353. https://doi.org/10.1017/S1471068406002973

[47] Omer Reingold. 2008. Undirected Connectivity in Log-Space. *J. ACM* 55, 4 (2008), 17:1–17:24. https://doi.org/10.1145/1391289.1391291

[48] Kenneth A. Ross and Yehoshua Sagiv. 1997. Monotonic Aggregation in Deductive Databases. *J. Comput. System Sci.* 54, 1 (1997), 79–97. https://doi.org/10.1006/jcss.1997.1453

[49] Gert Sabidussi. 1966. The Centrality Index of a Graph. *Psychometrika* 31, 4 (1966), 581–603. https://doi.org/10.1007/BF02289527

[50] John S. Schlipf. 1995. The Expressive Powers of the Logic Programming Semantics. *J. Comput. System Sci.* 51, 1 (1995), 64–86. https://doi.org/10.1006/jcss.1995.1053

[51] Uwe Schöning. 1997. Complexity of Presburger Arithmetic with Fixed Quantifier Dimension. *Theory Comput. Sci.* 30, 4 (1997), 423–428. https://doi.org/10.1007/BF02679468

[52] Alexander Schrijver. 1999. *Theory of Linear and Integer Programming.* Wiley.

[53] Jiwon Seo, Stephen Guo, and Monica S. Lam. 2015. SociaLite: An Efficient Graph Query Language Based on Datalog. *IEEE Trans. Knowl. Data Eng.* 27, 7 (2015), 1824–1837. https://doi.org/10.1109/TKDE.2015.2405562

[54] Alexander Shkapsky, Mohan Yang, Matteo Interlandi, Hsuan Chiu, Tyson Condie, and Carlo Zaniolo. 2016. Big Data Analytics with Datalog Queries on Spark. In *SIGMOD*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 1135–1149. https://doi.org/10.1145/2882903.2915229

[55] Iain A. Stewart. 1993. Incorporating Generalized Quantifiers and the Least Fixed Point Operator. In *CSL (LNCS, Vol. 832)*, Egon Börger, Yuri Gurevich, and Karl Meinke (Eds.). Springer, 318–333. https://doi.org/10.1007/BFb0049340

[56] S. Sudarshan and Raghu Ramakrishnan. 1991. Aggregation and Relevance in Deductive Databases. In *VLDB*, Guy M. Lohman, Amílcar Sernadas, and Rafael Camps (Eds.). Morgan Kaufmann, 501–511.

[57] S. Sudarshan, Divesh Srivastava, Raghu Ramakrishnan, and Catriel Beeri. 1993. Extending the Well-Founded and Valid Semantics for Aggregation. In *ILPS*, Dale Miller (Ed.). MIT Press, 590–608.

[58] David Toman and Jan Chomicki. 1998. Datalog with Integer Periodicity Constraints. *J. Log. Program.* 35, 3 (1998), 263–290. https://doi.org/10.1016/S0743-1066(97)10008-5

[59] Allen Van Gelder. 1992. The Well-Founded Semantics of Aggregation. In *PODS*, Moshe Y. Vardi and Paris C. Kanellakis (Eds.). ACM Press, 127–138. https://doi.org/10.1145/137097.137854

[60] Joachim von zur Gathen and Malte Sieveking. 1978. A Bound on Solutions of Linear Integer Equalities and Inequalities. *Proc. AMS* 72, 1 (1978), 155–158. https://doi.org/10.1090/S0002-9939-1978-0500555-0

[61] Jingjing Wang, Magdalena Balazinska, and Daniel Halperin. 2015. Asynchronous and Fault-Tolerant Recursive Datalog Evaluation in Shared-Nothing Engines. *PVLDB* 8, 12 (2015), 1542–1553. https://doi.org/10.14778/2824032.2824052

[62] Mohan Yang, Alexander Shkapsky, and Carlo Zaniolo. 2017. Scaling up the Performance of More Powerful Datalog Systems on Multicore Machines. *VLDB J.* 26, 2 (2017), 229–248. https://doi.org/10.1007/s00778-016-0448-z

[63] Carlo Zaniolo, Mohan Yang, Ariyam Das, Alexander Shkapsky, Tyson Condie, and Matteo Interlandi. 2017. Fixpoint Semantics and Optimization of Recursive Datalog Programs with Aggregates. *Theory Pract. Log. Program.* 17, 5-6 (2017), 1048–1065. https://doi.org/10.1017/S1471068417000436

APPENDIX

This appendix contains the proofs of the statements that have been omitted in the main body.

Theorem 2.2. *The fact entailment problem is undecidable for positive programs that do not mention ∞, $\times$ and $-$, and that use standard predicates with no object positions and at most one numeric position.*

Proof. We prove the theorem by reduction of the halting problem for deterministic Turing machines on the empty tape. To this end, consider an arbitrary deterministic Turing machine M with the set of states Q containing an initial state q^{init} and a halting state q^{halt}, the tape alphabet Γ containing the blank symbol $\llcorner\!\lrcorner$, and the transition function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\mathsf{left}, \mathsf{right}\}$, where left and right encode the left and right directions of the head moves, respectively. We assume that M works on a tape that is infinite to the right, starts with the empty tape (i.e., with $\llcorner\!\lrcorner$ in all cells) and the head positioned on the leftmost cell, and never moves the head off the left edge of the tape.

We encode each time point $t \in \mathbb{N}$ as the integer 2^t. Thus, at time point t, each cell with number $i \geq 2^t$ is necessarily blank, so we can unambiguously encode a combination of a time point t and a nonblank cell with number i using a single integer $2^t + i$. We use this idea to encode the computation of M by the following facts, where the last (an the only) position of every predicate except nullary *halts* is numeric:

- $\mathit{time}(k)$ says that $k = 2^t$ and so k encodes a time point t;
- $\mathit{tape}_a(2^t + i)$ says that symbol $a \in \Gamma$ occupies the cell with number i of the tape at time point t;
- $\mathit{head}_q(2^t + i)$ says that the head is over the cell with number i and the machine is in state $q \in Q$ at time point t; and
- *halts* says that the machine has halted.

We next construct a program $\mathcal{P}_M$ that simulates the computation of M on the empty tape. First, $\mathcal{P}_M$ contains rules (89), which initialise *time* so that it holds for each integer $k = 2^t$ with $t \in \mathbb{N}$.

$$\to \mathit{time}(1) \qquad\qquad \mathit{time}(m) \to \mathit{time}(m + m) \tag{89}$$

Then $\mathcal{P}_M$ contains rules (90), which encode the initial configuration of M.

$$\to \mathit{tape}_{\llcorner\!\lrcorner}(1) \qquad\qquad \to \mathit{head}_{q^{\mathsf{init}}}(1) \tag{90}$$

Rule (91) in $\mathcal{P}_M$ derives *halts* if at any point the Turing machine enters the halting state q^{halt}.

$$\mathit{head}_{q^{\mathsf{halt}}}(m) \to \mathit{halts} \tag{91}$$

Program $\mathcal{P}_M$ also contains inertia rules (92) and (93), whose role will be clear later, where (92) is actually a family of rules, one for each for each symbol $a \in \Gamma$, each state $q \in Q$, and each $\lhd$ among $<$ and $>$ (note that q does not play any role except of being a part of head_q, which is used to retrieve to the head position).

$$\begin{aligned}&\mathit{time}(m) \wedge \mathit{tape}_a(n) \wedge \mathit{head}_q(n') \wedge \\ &\qquad (m \leq n < m + m) \wedge (m \leq n' < m + m) \wedge (n \lhd n') \to \mathit{tape}_a(m + n)\end{aligned} \tag{92}$$

$$\mathit{time}(m) \wedge (m + m + m \leq n + n < m + m + m + m) \to \mathit{tape}_{\llcorner\!\lrcorner}(n) \tag{93}$$

Moreover, for each alphabet symbol $a \in \Gamma$ and each state $q \in Q$ with $\delta(q, a) = (q', a', X)$ for a direction $X \in \{\mathsf{left}, \mathsf{right}\}$, program $\mathcal{P}_M$ contains rule (94), where β is $(n' + 1 \doteq m + n)$ if X is left and $(n' \doteq m + n + 1)$ if X is right; note that, for conciseness, in (94) we allow conjunctions in rule heads: each generalised rule $\varphi \to \psi$ with ψ a conjunction of atoms abbreviates the set $\{\varphi \to \alpha \mid \alpha \text{ atom in } \psi\}$ of conventional rules.

$$\mathit{time}(m) \wedge \mathit{tape}_a(n) \wedge \mathit{head}_q(n) \wedge (m \leq n < m + m) \wedge \beta \to \mathit{tape}_{a'}(m + n) \wedge \mathit{head}_{q'}(n') \tag{94}$$

Rules (92)–(94) encode the computation of M, and they are based on the following idea: if variable m encodes a time point t using integer 2^t, then variable n encodes a cell with number i at time point t whenever $m \leq n < m + m$; moreover, for such n, the cell at time point $t + 1$ is encoded as $2^{t+1} + i = 2^t + 2^t + i$ and can be obtained as $m + n$, while the encodings of the cells to the left and to the right of this cell at this time point can be obtained as $m + n - 1$ and $m + n + 1$, respectively. Since our goal is to prove undecidability using only +, we simulate subtraction by looking for a number n' such that $m + n = n' + 1$. With these observations in mind, one can see that rule (92) copies the unaffected part of the tape from time point t to time point $t + 1$. Moreover, rule (93) pads the tape by filling all cells with numbers $i \in [1.5 \cdot 2^t, 2 \cdot 2^t - 1]$ with blank symbols; since division is not supported in our language, we express this condition as $3 \cdot 2^t \leq 2 \cdot i < 4 \cdot 2^t$. Finally, rule (94) updates the state and the tape at the position of the head, and moves the head left or right according to the transition function. Thus, $\mathcal{P}_M \models \mathit{halts}$ if and only if M halts on the empty tape. □

The following holds:

1. $\mathcal{I} \models \mathcal{P}$ if and only if $\mathcal{S}_{(\mathcal{P},\tau)}(\mathcal{I}) \subseteq \mathcal{I}$, for each limit program $(\mathcal{P}, \tau)$ and each interpretation $\mathcal{I}$ that is limit-closed for $(\mathcal{P}, \tau)$; and
2. $\mathcal{S}_{(\mathcal{P},\tau)}(\mathcal{I}_1) \subseteq \mathcal{S}_{(\mathcal{P},\tau)}(\mathcal{I}_2)$ for each semi-positive limit program $(\mathcal{P}, \tau)$ and interpretations $\mathcal{I}_1$ and $\mathcal{I}_2$ that are limit-closed for $(\mathcal{P}, \tau)$, satisfy $\mathcal{I}_1 \subseteq \mathcal{I}_2$, and coincide on the EDB predicates.

Proof. For claim 1, consider an interpretation $\mathcal{I}$ limit-closed for $(\mathcal{P}, \tau)$. Assume first that $\mathcal{S}_{(\mathcal{P},\tau)}(\mathcal{I}) \subseteq \mathcal{I}$ and let $\rho = \varphi \to \alpha$ be a ground instance of a rule in $\mathcal{P}$. If $\mathcal{I} \models \varphi$, then the definition of $\mathcal{S}_{(\mathcal{P},\tau)}(\mathcal{I})$ ensures that $\mathcal{I} \models \rho$, as required. For the converse direction, assume that $\mathcal{S}_{(\mathcal{P},\tau)}(\mathcal{I}) \not\subseteq \mathcal{I}$. Since $\mathcal{I}$ is limit-closed for $(\mathcal{P}, \tau)$, there must be a ground instance $\varphi \to \alpha$ of a rule $\rho \in \mathcal{P}$ such that $\mathcal{I} \models \varphi$ but $\mathcal{I} \not\models \alpha$, meaning that $\mathcal{I} \not\models \rho$ and $\mathcal{I} \not\models \mathcal{P}$ as a result.

For claim 2, simply recall that $\mathcal{S}_{(\mathcal{P},\tau)}(\mathcal{I})$ is the limit-closure of $\mathcal{S}_{\mathcal{P}}(\mathcal{I})$ for every limit-closed interpretation $\mathcal{I}$, and that both $\mathcal{S}_{\mathcal{P}}$ and the closure operator are monotonic for interpretations coinciding on the EDB predicates in $\mathcal{P}$ with respect to set inclusion when $\mathcal{P}$ is semi-positive. □

For each limit program $(\mathcal{P}, \tau)$ and each fact γ, we can compute in linear time a homogeneous program $(\mathcal{P}', \tau')$ and a fact γ' such that $(\mathcal{P}, \tau) \models \gamma$ if and only if $(\mathcal{P}', \tau') \models \gamma'$.

Proof. Let $(\mathcal{P}, \tau)$ be an arbitrary limit program. We construct a limit program $(\mathcal{P}', \tau')$ with all limit predicates max (a program with all limit predicates min would be constructed in a similar way). Let, for each min predicate C in $(\mathcal{P}, \tau)$, program $(\mathcal{P}', \tau')$ use a fresh max predicate C' of the same arity and uniquely associated to C. Then, let $\mathcal{P}'$ be constructed from $\mathcal{P}$ by replacing each min atom $C(\mathbf{t}, s)$ by the max atom $C'(\mathbf{t}, -s)$, if $s \neq \infty$, and by $C'(\mathbf{t}, \infty)$ otherwise. Finally, let γ' be obtained from γ in the same way (assuming that -0 is 0 and $-(-k)$ is k for k a positive integer); in particular, unless γ is a min fact, we have $\gamma' = \gamma$.

Consider an interpretation $\mathcal{I}$ limit-closed for $(\mathcal{P}, \tau)$, and let $\mathcal{I}'$ be an interpretation limit-closed for $(\mathcal{P}', \tau')$ that is obtained from $\mathcal{I}$ by replacing each min fact $C(\mathbf{a}, k)$ with $C'(\mathbf{a}, -k)$; it is straightforward to see that $\mathcal{I} \models \mathcal{P}$ if and only if $\mathcal{I}' \models \mathcal{P}'$, and that $\mathcal{I} \models \gamma$ if and only if $\mathcal{I}' \models \gamma'$. By induction on the construction of $\mathcal{M}(\mathcal{P}, \tau)$, it follows that $(\mathcal{P}, \tau) \models \gamma$ if and only if $(\mathcal{P}', \tau') \models \gamma'$. □

For every positive LL-program $\mathcal{P}$, $\mathcal{M}(\mathcal{P}) = \mathcal{M}(\mathcal{G}(\mathcal{P}))$.

Proof. Consider the partial materialisations $\mathcal{M}_i^\kappa$ of a positive LL-program $\mathcal{P}$, for ordinals κ and numbers $i \geq 1$. Since $\mathcal{P}$ is positive, $\mathcal{M}(\mathcal{P}) = \mathcal{M}_1^{\omega_1}$, and it is enough to prove, by transfinite induction on κ, that each $\mathcal{M}_1^\kappa$ is also the partial materialisation of $\mathcal{G}(\mathcal{P})$ for the same ordinal κ and stratum 1. To this end, note that $\mathcal{M}_1^0 = \mathcal{M}_0^{\omega_1} = \emptyset$ is also the partial materialisation of $\mathcal{G}(\mathcal{P})$ for ordinal 0, and hence we need to show that $\mathcal{S}_{\mathcal{P}}(\mathcal{M}_1^\kappa) = \mathcal{S}_{\mathcal{G}(\mathcal{P})}(\mathcal{M}_1^\kappa)$ for every $\kappa > 0$. On the one hand, we have $\mathcal{S}_{\mathcal{P}}(\mathcal{M}_1^\kappa) \subseteq \mathcal{S}_{\mathcal{G}(\mathcal{P})}(\mathcal{M}_1^\kappa)$ because each object and each integer in an exact fact in $\mathcal{M}_1^\kappa$ is also mentioned in $\mathcal{P}$ (the latter holds since all exact predicates are EDB); on the other hand, $\mathcal{S}_{\mathcal{G}(\mathcal{P})}(\mathcal{M}_1^\kappa) \subseteq \mathcal{S}_{\mathcal{P}}(\mathcal{M}_1^\kappa)$ because each ground instance of a rule in $\mathcal{G}(\mathcal{P})$ is also an instance of a rule in $\mathcal{P}$. □

For each OG-ground rule ρ with body φ and each pseudointerpretation $\mathcal{J}$, a grounding σ of ρ is a solution to the IP $\psi(\rho, \mathcal{J})$ if and only if $\mathcal{J} \models \varphi\sigma$.

Proof. We first argue the forward direction of the claim. Let σ be a solution to $\psi(\rho, \mathcal{J})$ and let α be an atom in φ. We show that $\mathcal{J} \models \alpha\sigma$ by distinguishing the following cases.

First, if α is object or exact, then, since $\psi(\rho, \mathcal{J})$ has a solution, $\alpha \in \mathcal{J}$ and hence $\mathcal{J} \models \alpha$.

Next, if α is limit, then, since $\psi(\rho, \mathcal{J})$ has a solution,

- $\alpha = C(\mathbf{a}, s)$ for $s \neq \infty$ and there exists $\ell \in \mathbb{Z}$ with $(s \preceq_C \ell)$ in $\psi(\rho, \mathcal{J})$ and $C(\mathbf{a}, \ell) \in \mathcal{J}$,
- $\alpha = C(\mathbf{a}, s)$ for $s \neq \infty$ and $C(\mathbf{a}, \infty) \in \mathcal{J}$, or
- $\alpha = C(\mathbf{a}, \infty)$ and $\alpha \in \mathcal{J}$.

In the first case, we have $s\sigma \preceq_C \ell$ since σ is a solution, and hence $\mathcal{J} \models C(\mathbf{a}, s\sigma)$. In the second case, $\mathcal{J} \models C(\mathbf{a}, s\sigma)$ due to $C(\mathbf{a}, s\sigma) \sqsubseteq C(\mathbf{a}, \infty)$. In the third case, $\mathcal{J} \models \alpha$ by construction.

Finally, if α is a comparison atom, then α is in $\psi(\rho, \mathcal{J})$, and the claim trivially follows.

The converse direction of the lemma is analogous to the forward direction. □

For each OG-ground program $\mathcal{P}$ and each ordinal κ, the partial materialisation $\mathcal{M}_1^\kappa$ of $\mathcal{P}$ and the partial pseudomaterialisation $\mathcal{N}^\kappa$ of $\mathcal{P}$ correspond to each other.

Proof. The lemma follows by a simple transfinite induction based on the following claims.

1. Let $\mathcal{I}$ be a limit-closed interpretation and let $\mathcal{J}$ be its corresponding pseudointerpretation. Then the limit-closed interpretation $\mathcal{S}_{\mathcal{P}}(\mathcal{I})$ corresponds to the pseudointerpretation $\mathcal{T}_{\mathcal{P}}(\mathcal{J})$.
2. Let I be a set of limit-closed interpretations and let J be the set of their corresponding pseudointerpretations. Then the limit-closed interpretation $\bigcup I$ corresponds to the pseudointerpretation $\sup J$.

We start by showing claim 1. For this, consider a limit-closed interpretation $\mathcal{I}$, the corresponding pseudointerpretation $\mathcal{J}$, and a pseudofact γ. We distinguish the cases given next.

Let γ be an object fact. We show that $\gamma \in \mathcal{S}_{\mathcal{P}}(\mathcal{I})$ if and only if $\gamma \in \mathcal{T}_{\mathcal{P}}(\mathcal{J})$. Assume first that $\gamma \in \mathcal{S}_{\mathcal{P}}(\mathcal{I})$. Then, there exists a grounding σ of a rule $\rho = \varphi \to \gamma$ in $\mathcal{P}$ such that $\mathcal{I} \models \varphi\sigma$. Since $\mathcal{J}$ corresponds to $\mathcal{I}$, $\mathcal{J} \models \varphi\sigma$, and Lemma 4.1 ensures that σ is a solution to $\psi(\rho, \mathcal{J})$. Moreover, $\delta(\rho, \mathcal{J}) = \gamma$, and thus $\gamma \in \mathcal{T}_{\mathcal{P}}(\mathcal{J})$ as required. For the converse direction, assume that $\gamma \in \mathcal{T}_{\mathcal{P}}(\mathcal{J})$. Then, there exist a rule $\rho = \varphi \to \gamma$ in $\mathcal{P}$ and a solution σ to $\psi(\rho, \mathcal{J})$. Lemma 4.1 then ensures that $\mathcal{J} \models \varphi\sigma$, and so $\mathcal{I} \models \varphi\sigma$ as well. Therefore, $\gamma\sigma = \gamma \in \mathcal{S}_{\mathcal{P}}(\mathcal{I})$.

Let γ be an exact fact. It immediately follows that $\gamma \in \mathcal{S}_{\mathcal{P}}(\mathcal{I})$ if and only if $\gamma \in \mathcal{T}_{\mathcal{P}}(\mathcal{J})$, because both memberships hold if only if $\gamma \in \mathcal{P}$.

Let γ be a limit fact of the form $C(\mathbf{a}, k)$ with $k \in \mathbb{Z}$. We show that $\gamma \in \mathcal{S}_{\mathcal{P}}(\mathcal{I})$ if and only if $\gamma \sqsubseteq \mathcal{T}_{\mathcal{P}}(\mathcal{J})$. Assume first that $\gamma \in \mathcal{S}_{\mathcal{P}}(\mathcal{I})$. Then, there exists a grounding σ of a rule $\rho = \varphi \to C(\mathbf{a}, s)$ in $\mathcal{P}$ such that $\mathcal{I} \models \varphi\sigma$ and either $s = \infty$ or $k \preceq_C s\sigma$. Analogously to the case where γ is an object fact, Lemma 4.1 ensures that σ is a solution to $\psi(\rho, \mathcal{J})$. Therefore, in the case when $s \neq \infty$, either the IOP $(\psi(\rho, \mathcal{J}), \max_C s)$ is unbounded, or $s\sigma \preceq_C \ell$, for ℓ the optimal value of the IOP. Since $\rho \in \mathcal{P}$, $\gamma \sqsubseteq \delta(\rho, \mathcal{J}) = C(\mathbf{a}, \ell') \sqsubseteq \mathcal{T}_{\mathcal{P}}(\mathcal{J})$ as required, where $\ell' = \infty$ if $s = \infty$ or the IOP is unbounded, or $\ell' = \ell$ otherwise. For the converse direction, assume that $\gamma \sqsubseteq \mathcal{T}_{\mathcal{P}}(\mathcal{J})$. Then there exist a rule $\rho = \varphi \to C(\mathbf{a}, s)$ in $\mathcal{P}$ and a solution σ to the IP $\psi(\rho, \mathcal{J})$ such that $\gamma = C(\mathbf{a}, k) \sqsubseteq \delta(\rho, \mathcal{J}) = C(\mathbf{a}, \ell)$, where ℓ is either ∞ if $s = \infty$, or ℓ is the value of $s\sigma$ otherwise. Lemma 4.1 then ensures $\mathcal{J} \models \varphi\sigma$, and so $\mathcal{I} \models \varphi\sigma$ as well. Hence, either $C(\mathbf{a}, k') \in \mathcal{S}_{\mathcal{P}}(\mathcal{I})$ for all $k' \in \mathbb{Z}$ (if $s = \infty$) or $C(\mathbf{a}, \ell) \in \mathcal{S}_{\mathcal{P}}(\mathcal{I})$ (if $s \neq \infty$), and the fact that $\mathcal{S}_{\mathcal{P}}(\mathcal{I})$ is limit-closed ensures that $\gamma = C(\mathbf{a}, k) \in \mathcal{S}_{\mathcal{P}}(\mathcal{I})$.

Let γ be a limit pseudofact of the form $C(\mathbf{a}, \infty)$. We show that, for the set $\mathcal{K} = \{C(\mathbf{a}, k) \mid k \in \mathbb{Z}\}$, $\mathcal{K} \subseteq \mathcal{S}_{\mathcal{P}}(\mathcal{I})$ if and only if $\gamma \in \mathcal{T}_{\mathcal{P}}(\mathcal{J})$. Assume first that $\mathcal{K} \subseteq \mathcal{S}_{\mathcal{P}}(\mathcal{I})$. Since $\mathcal{P}$ is finite, the infinitely many facts of $\mathcal{K}$ in $\mathcal{S}_{\mathcal{P}}(\mathcal{I})$ can be produced in only two ways:

- by a rule $\rho = \varphi \to C(\mathbf{a}, \infty)$ in $\mathcal{P}$ and a grounding σ of ρ such that $\mathcal{I} \models \varphi\sigma$; or
- by a rule $\rho = \varphi \to C(\mathbf{a}, s)$ with $s \neq \infty$ in $\mathcal{P}$ and an infinite sequence σ_i, $i \in \mathbb{N}$, of groundings of ρ such that, for each i, we have $\mathcal{I} \models \varphi\sigma_i$ and $s\sigma_i \prec_C s\sigma_{i+1}$.

In the former case, $\gamma \in \mathcal{T}_{\mathcal{P}}(\mathcal{J})$ follows analogously to the case of γ an object fact. In the latter case, $\mathcal{J} \models \varphi\sigma_i$ and Lemma 4.1 ensures that each σ_i is a solution to $\psi(\rho, \mathcal{J})$. Moreover, since the sequence $s\sigma_i$, $i \in \mathbb{N}$, is strictly ascending with respect to $\prec_C$, the IOP $(\psi(\rho, \mathcal{J}), \max_C s)$ is unbounded. Since $\rho \in \mathcal{P}$, we then have $\gamma = \delta(\rho, \mathcal{J}) \in \mathcal{T}_{\mathcal{P}}(\mathcal{J})$, as required. For the converse direction, assume $C(\mathbf{a}, \infty) \in \mathcal{T}_{\mathcal{P}}(\mathcal{J})$. Then, a rule $\rho = \varphi \to C(\mathbf{a}, s)$ exists in $\mathcal{P}$ such that either $s = \infty$ or $(\psi(\rho, \mathcal{J}), \max_C s)$ is unbounded. Again, in the former case, the proof proceeds analogously to the case of γ an object fact. In the latter case, an infinite sequence σ_i, $i \in \mathbb{N}$, of solutions to $\psi(\rho, \mathcal{J})$ exists such that $s\sigma_i \prec_C s\sigma_{i+1}$, for $i \in \mathbb{N}$. Lemma 4.1 ensures $\mathcal{J} \models \varphi\sigma_i$, for $i \in \mathbb{N}$, and so $\mathcal{I} \models \varphi\sigma_i$ as well. Thus, for each $k \in \mathbb{Z}$, some $i \in \mathbb{N}$ exists such that $k \prec_C s\sigma_i$. Then limit-closedness of $\mathcal{S}_{\mathcal{P}}(\mathcal{I})$ ensures that $\mathcal{K} \subseteq \mathcal{S}_{\mathcal{P}}(\mathcal{I})$, as required.

We now turn to claim 2. Again, it suffices to consider an arbitrary pseudofact γ and the cases specified next.

Let γ be an object or exact fact. We show that $\gamma \in \bigcup I$ if and only if $\gamma \in \sup J$. If $\gamma \in \bigcup I$, then $\gamma \in \mathcal{I}$ for some $\mathcal{I} \in I$, and hence $\gamma \in \mathcal{J}$ for $\mathcal{J} \in J$ corresponding to $\mathcal{I}$; thus, $\gamma \in \sup J$ as required. The converse direction is analogous.

Let γ be a limit fact $C(\mathbf{a}, k)$ where $k \in \mathbb{Z}$. We show that $\gamma \in \bigcup I$ if and only if $\gamma \sqsubseteq \sup J$. Assume first $\gamma \in \bigcup I$. Then $\gamma \in \mathcal{I}$ for some $\mathcal{I} \in I$, and $\gamma \sqsubseteq \mathcal{J}$ for $\mathcal{J} \in J$ corresponding to $\mathcal{I}$. Thus, $\gamma \sqsubseteq \sup J$, as required. The converse direction is analogous.

Let γ be a limit pseudofact of the form $C(\mathbf{a}, \infty)$. We show that, for $\mathcal{K} = \{C(\mathbf{a}, k) \mid k \in \mathbb{Z}\}$, $\mathcal{K} \subseteq \bigcup I$ if and only if $\gamma \in \sup J$. Assume $\mathcal{K} \subseteq \bigcup I$. Then, for each $k \in \mathbb{Z}$ there is $\mathcal{I}_k \in I$ such that $C(\mathbf{a}, k) \in \mathcal{I}_k$, and hence $C(\mathbf{a}, k) \sqsubseteq \mathcal{J}_k$ for $\mathcal{J}_k \in J$ corresponding to $\mathcal{I}_k$. Therefore, $\gamma \in \sup J$, as required. The converse direction is analogous. □

Let $\mathcal{P}$ be an OG-ground program, and let $\mathcal{J}$ and σ be a pseudointerpretation and a variable assignment, respectively, such that $\mathcal{J}$ and σ correspond to each other. Then, $\mathcal{J} \models \mathcal{P}$ if and only if σ is a solution to $\xi_{\mathcal{P}}$.

Proof. We first prove that $\mathcal{J} \models \alpha$ if and only if σ is a solution to ξ_α for every function-free ground atom α that is ∞-free whenever it is exact. Let us consider all possible forms of α.

Let $\alpha = A(\mathbf{a})$ be an object atom. Then, $\xi_\alpha = \mathit{def}_{A\mathbf{a}}$, and $\sigma(\mathit{def}_{A\mathbf{a}}) = \text{true}$ if and only if $\alpha \in \mathcal{J}$; so the claim follows.

Let $\alpha = B(\mathbf{a}, k)$ be an exact atom with $k \in \mathbb{Z}$. Then, $\xi_\alpha = \mathit{def}_{B\mathbf{a}k}$ and $\sigma(\mathit{def}_{B\mathbf{a}k}) = \text{true}$ if and only if $B(\mathbf{a}, k) \in \mathcal{J}$; so the claim follows.

Let $\alpha = C(\mathbf{a}, s)$ be a limit atom with $s \neq \infty$. If $\mathcal{J} \models \alpha$, then either $C(\mathbf{a}, \infty) \in \mathcal{J}$ or an integer ℓ exists such that $C(\mathbf{a}, \ell) \in \mathcal{J}$ and $s \preceq_C \ell$. Either way, $\sigma(\mathit{def}_{C\mathbf{a}}) = \text{true}$; moreover, $\sigma(\mathit{fin}_{C\mathbf{a}}) = \text{false}$ in the former and $\sigma(\mathit{val}_{C\mathbf{a}}) = \ell$ in the latter case. Thus, σ is a solution to ξ_α by definition. The converse direction is analogous.

Let $\alpha = C(\mathbf{a}, \infty)$ be a limit atom. The proof is analogous to the previous case.

Let α be a comparison atom. Then, $\xi_\alpha = \alpha$, and the truth of α is independent of $\mathcal{J}$ and σ; so the claim is immediate.

Consider now the limit-closed interpretation $\mathcal{I}$ corresponding to $\mathcal{J}$. By definition, for each rule $\rho \in \mathcal{P}$, $\mathcal{J} \models \rho$ if and only if $\mathcal{I} \models \rho$, and the latter is equivalent to $\mathcal{I} \models \rho'$ for each ground instance ρ' of ρ by the semantics of universal quantification in first-order logic. Hence, the claim can be restated as follows: for each $\rho \in \mathcal{P}$ and each grounding σ' of ρ, $\mathcal{J} \models \rho\sigma'$ if and only if σ' is a solution to ξ_ρ. Now note that, by construction, $\xi_{\rho\sigma'} = (\xi_\rho)\sigma'$ for each grounding σ' of ρ, and groundings of rules $\rho \in \mathcal{P}$ can be equivalently seen as assignments to the universally quantified numeric variables in ξ_ρ; so the claim of the lemma follows immediately from the fact that $\mathcal{J} \models \alpha$ if and only if σ is a solution to ξ_α for every ground atom α. □

For every semi-positive LL-program $\mathcal{P}$, $\mathcal{M}(\mathcal{P}) = \mathcal{M}(\mathcal{R}(\mathcal{P}))$.

Proof. Consider a semi-positive LL-program $\mathcal{P}$. First, note that $\mathcal{M}(\mathcal{P}) = \mathcal{M}(\mathcal{G}(\mathcal{P}))$ by the same argument as in the proof of Lemma 4.1. Thus, it remains to show $\mathcal{M}(\mathcal{G}(\mathcal{P})) = \mathcal{M}(\mathcal{R}(\mathcal{P}))$.

We first prove that $\mathcal{M}(\mathcal{G}(\mathcal{P})) \subseteq \mathcal{M}(\mathcal{R}(\mathcal{P}))$. For each number $i \geq 1$ and ordinal κ, let $\mathcal{M}_i^\kappa$ be the corresponding partial materialisation of $\mathcal{G}(\mathcal{P})$. Since $\mathcal{G}(\mathcal{P})$ is semi-positive, $\mathcal{M}(\mathcal{G}(\mathcal{P})) = \mathcal{M}_2^{\omega_1}$, and thus it is enough to show, by transfinite induction on κ, that $\mathcal{M}_2^\kappa \subseteq \mathcal{M}(\mathcal{R}(\mathcal{P}))$ for each ordinal κ.

Consider first the case when $\kappa = 0$. Then, by definition, $\mathcal{M}_2^0 = \mathcal{M}_1^{\omega_1} = \mathcal{M}(\mathcal{G}(\mathcal{P})[1])$ (it is worth reminding that $\mathcal{G}(\mathcal{P})[1]$ is the first stratum of $\mathcal{G}(\mathcal{P})$). However, $\mathcal{G}(\mathcal{P})[1]$ is a positive LL-program and hence $\mathcal{G}(\mathcal{P})[1] \subseteq \mathcal{R}(\mathcal{P})$ by construction, which implies $\mathcal{M}(\mathcal{G}(\mathcal{P})[1]) \subseteq \mathcal{M}(\mathcal{R}(\mathcal{P}))$.

Let now $\kappa = \kappa' + 1$ and the claim hold for κ'. Consider a fact $\gamma \in \mathcal{M}_2^\kappa$; we need to show that $\gamma \in \mathcal{M}(\mathcal{R}(\mathcal{P}))$. By definition, $\mathcal{M}_2^\kappa = \mathcal{S}_{\mathcal{G}(\mathcal{P})[2]}(\mathcal{M}_2^{\kappa'}) \cup \mathcal{M}_1^{\omega_1}$. If $\gamma \in \mathcal{M}_1^{\omega_1}$, then $\gamma \in \mathcal{M}(\mathcal{R}(\mathcal{P}))$ follows as shown in the base case. Consider now the case $\gamma \in \mathcal{S}_{\mathcal{G}(\mathcal{P})[2]}(\mathcal{M}_2^{\kappa'})$. By definition, there is a rule $\rho = \varphi \to \alpha$ in $\mathcal{G}(\mathcal{P})[2]$, a grounding σ such that $\mathcal{M}_2^{\kappa'} \models \varphi\sigma$, and a fact γ' that can be obtained from $\alpha\sigma$ by replacing occurrences of ∞ with integers and evaluating numeric terms such that $\gamma \sqsubseteq \gamma'$. Since $\mathcal{M}(\mathcal{R}(\mathcal{P})) \models \mathcal{R}(\mathcal{P})$, it suffices to show that there is a rule $\rho' = \varphi' \to \alpha$ in $\mathcal{R}(\mathcal{P})$ obtained from ρ as in Definition 4.21 such that $\mathcal{M}(\mathcal{R}(\mathcal{P})) \models \varphi'\sigma$. To this end, let φ' be obtained from φ by

- deleting all negative object literals;
- deleting all negative exact literals $\mathsf{not}\, B(\mathbf{a}, s)$ such that $B(\mathbf{a}, k) \notin \mathcal{G}(\mathcal{P})$ for each $k \in \mathbb{Z}$ and replacing all other such literals with comparison atoms $(s < k_1)$ if $s\sigma < k_1$, with $(k_{i-1} < s < k_i)$ if $k_{i-1} < s\sigma < k_i$ for $i \in [2, h]$, and with $(k_h < s)$ if $k_h < s\sigma$, where $k_1 < \cdots < k_h$ are all integers such that $B(\mathbf{a}, k_i) \in \mathcal{G}(\mathcal{P})$, for $i \in [1, h]$;
- deleting all negative limit predicates of the form $\mathsf{not}\, C(\mathbf{a}, \infty)$; and
- deleting all negative limit literals $\mathsf{not}\, C(\mathbf{a}, s)$ with $s \neq \infty$ such that $C(\mathbf{a}, \ell) \notin \mathcal{G}(\mathcal{P})$ for each $\ell \in \mathbb{Z}$ and replacing all other such literals with a single comparison atom $(k \prec_C s)$ for $k = \max_C\{\ell \in \mathbb{Z} \mid C(\mathbf{a}, \ell) \in \mathcal{G}(\mathcal{P})\}$.

Note that, in the second case, we have $s\sigma \neq k_i$ for each $i \in [1, h]$ since $\mathcal{M}_2^{\kappa'} \models \mathsf{not}\, B(\mathbf{a}, s\sigma)$ by assumption, and all facts in $\mathcal{G}(\mathcal{P})$ are contained in $\mathcal{M}_2^{\kappa'}$; thus, φ' is always defined. We next show that the rule $\rho' = \varphi' \to \alpha$ is obtained from ρ according to Definition 4.21 and hence belongs to $\mathcal{R}(\mathcal{P})$. Indeed, by definition, the only possibility why this may not be the case is that ρ is deleted, which implies that φ contains a negative literal $\mathsf{not}\, \alpha'$ such that one of the following holds:

- α' is object and $\alpha' \in \mathcal{G}(\mathcal{P})$, or
- $\alpha' = C(\mathbf{a}, s)$ is limit and $C(\mathbf{a}, \infty) \in \mathcal{G}(\mathcal{P})$;

none of these cases, however, is possible since $\mathcal{M}_2^{\kappa'} \models \varphi\sigma$ by assumption, $\mathcal{G}(\mathcal{P})$ is semi-positive, and the predicate in α is EDB. We are left to show that $\mathcal{M}(\mathcal{R}(\mathcal{P})) \models \varphi'\sigma$. Since $\mathcal{M}_2^{\kappa'} \models \varphi\sigma$ and, by the inductive hypothesis, $\mathcal{M}_2^{\kappa'} \subseteq \mathcal{M}(\mathcal{R}(\mathcal{P}))$, $\mathcal{M}(\mathcal{R}(\mathcal{P}))$ satisfies all positive literals in $\varphi\sigma$, which are in

$\varphi'\sigma$ by construction. Hence, we only need to check that $\beta\sigma$ evaluates to true for each comparison atom β in φ' replacing a negative literal $\mathsf{not}\,\alpha'$ in φ. By construction, there are two cases:

- if β is one of $(s < k_1)$, $(k_{i-1} < s < k_i)$ for $i \in [2, h]$, and $(k_h < s)$, with $\alpha' = B(\mathbf{a}, s)$ exact and h as above, then $\beta\sigma$ evaluates to true by the choice of β in the construction of φ';
- if $\beta = (k \prec_C s)$, $\alpha' = C(\mathbf{a}, s)$ is limit, and $k = \max_C\{\ell \in \mathbb{Z} \mid C(\mathbf{a}, \ell) \in \mathcal{G}(\mathcal{P})\}$, then $\beta\sigma$ evaluates to true because $\mathcal{M}_2^{\kappa'} \models \mathsf{not}\,\alpha'\sigma$ (since $\mathcal{M}_2^{\kappa'} \models \varphi\sigma$) and C is EDB.

Finally, let κ be a limit ordinal and, by the inductive hypothesis, $\mathcal{M}_2^{\kappa'} \subseteq \mathcal{M}(\mathcal{R}(\mathcal{P}))$ for all $\kappa' < \kappa$. Then $\mathcal{M}_2^{\kappa} \subseteq \mathcal{M}(\mathcal{R}(\mathcal{P}))$ follows since $\mathcal{M}_2^{\kappa} = \bigcup_{\kappa' < \kappa} \mathcal{M}_2^{\kappa'}$ by definition.

We next prove that $\mathcal{M}(\mathcal{R}(\mathcal{P})) \subseteq \mathcal{M}(\mathcal{G}(\mathcal{P}))$. Let now, for every $i \geq 1$ and ordinal κ, $\mathcal{M}_i^{\kappa}$ be the partial materialisations of $\mathcal{R}(\mathcal{P})$. Since $\mathcal{R}(\mathcal{P})$ is positive, $\mathcal{M}(\mathcal{R}(\mathcal{P})) = \mathcal{M}_1^{\omega_1}$, and therefore it is enough to show that $\mathcal{M}_1^{\kappa} \subseteq \mathcal{M}(\mathcal{G}(\mathcal{P}))$ for every ordinal κ by transfinite induction on κ.

If $\kappa = 0$, then the claim is immediate, because $\mathcal{M}_1^0 = \emptyset$ by definition.

Let now $\kappa = \kappa' + 1$ and the claim hold for κ'. Consider a fact $\gamma \in \mathcal{M}_1^{\kappa}$; we need to show that $\gamma \in \mathcal{M}(\mathcal{G}(\mathcal{P}))$. Since $\mathcal{M}_1^{\kappa} = \mathcal{S}_{\mathcal{R}(\mathcal{P})}(\mathcal{M}_1^{\kappa'})$, there is a rule $\rho' = \varphi' \to \alpha$ in $\mathcal{R}(\mathcal{P})$, a grounding σ such that $\mathcal{M}_1^{\kappa'} \models \varphi'\sigma$, and a fact γ' that can be obtained from $\alpha\sigma$ by replacing occurrences of ∞ with integers and evaluating numeric terms such that $\gamma \sqsubseteq \gamma'$. Consider the rule $\rho = \varphi \to \alpha$ in $\mathcal{G}(\mathcal{P})$ that yields ρ' in $\mathcal{R}(\mathcal{P})$. Since $\mathcal{M}(\mathcal{G}(\mathcal{P})) \models \mathcal{G}(\mathcal{P})$, it suffices to show that $\mathcal{M}(\mathcal{G}(\mathcal{P})) \models \varphi\sigma$. Since $\mathcal{M}_1^{\kappa'} \models \varphi'\sigma$ and, by the inductive hypothesis, $\mathcal{M}_1^{\kappa'} \subseteq \mathcal{M}(\mathcal{G}(\mathcal{P}))$, $\mathcal{M}(\mathcal{G}(\mathcal{P}))$ satisfies all positive literals in $\varphi'\sigma$, which are copied from $\varphi\sigma$. Thus, we only need to check that $\mathcal{M}(\mathcal{G}(\mathcal{P})) \models \mathsf{not}\,\alpha'\sigma$ for each negative literal $\mathsf{not}\,\alpha'$ in φ. By construction, we have the following cases:

- if α' is an object atom, then, since ρ is not deleted, $\alpha' \notin \mathcal{G}(\mathcal{P})$ by the definition of the reduct; therefore, the claim follows because $\mathcal{P}$ is semi-positive and hence the predicate of α' is EDB;
- if $\alpha' = B(\mathbf{a}, s)$ is an exact atom with $B(\mathbf{a}, k) \in \mathcal{G}(\mathcal{P})$ for some $k \in \mathbb{Z}$, then $s\sigma \neq k$ for each such k since the comparisons replacing $\mathsf{not}\,\alpha'$ in φ' hold in $\mathcal{M}_1^{\kappa'}$ under σ, and the claim follows by the same reasons as in the preceding case;
- if $\alpha' = B(\mathbf{a}, s)$ is an exact atom with $B(\mathbf{a}, k) \notin \mathcal{G}(\mathcal{P})$ for all $k \in \mathbb{Z}$, then the claim follows as above;
- if $\alpha' = C(\mathbf{a}, s)$ is a limit atom with $s \neq \infty$ and $C(\mathbf{a}, \ell) \in \mathcal{G}(\mathcal{P})$ for some $\ell \in \mathbb{Z}$, then, since ρ is not deleted, $C(\mathbf{a}, \infty) \notin \mathcal{G}(\mathcal{P})$; moreover, $\max_C\{\ell \in \mathbb{Z} \mid C(\mathbf{a}, \ell) \in \mathcal{G}(\mathcal{P})\} \prec_C s\sigma$ since the comparison atom replacing $\mathsf{not}\,\alpha'$ in φ' holds in $\mathcal{M}_1^{\kappa'}$ under σ; hence, the claim then follows as above;
- if $\alpha' = C(\mathbf{a}, s)$ is a limit atom with either $s = \infty$ or $C(\mathbf{a}, \ell) \notin \mathcal{G}(\mathcal{P})$ for all $\ell \in \mathbb{Z} \cup \{\infty\}$, then the claim follows as above.

Finally, let κ be a limit ordinal and, by the inductive hypothesis, $\mathcal{M}_1^{\kappa'} \subseteq \mathcal{M}(\mathcal{G}(\mathcal{P}))$ for all $\kappa' < \kappa$. Then $\mathcal{M}_1^{\kappa} \subseteq \mathcal{M}(\mathcal{G}(\mathcal{P}))$ follows since $\mathcal{M}_1^{\kappa} = \bigcup_{\kappa' < \kappa} \mathcal{M}_1^{\kappa'}$. □

For each stable OG-ground program $\mathcal{P}$ and each pseudointerpretation $\mathcal{J}$, there is $i \in [1, |\mathcal{P}|]$ such that one of the following holds, where $\mathcal{J}_0 = \mathcal{J}$ and $\mathcal{J}_j = \mathcal{T}_{\mathcal{P}}(\mathcal{J}_{j-1})$ for each $j \geq 1$:

1. $\mathcal{J}_i = \mathcal{J}_{i-1}$,
2. there is a rule in $\mathcal{P}$ that is applicable to $\mathcal{J}_i$ but not to $\mathcal{J}_{i-1}$,
3. there is a node $\langle C\mathbf{a} \rangle$ on a positive-weight cycle in the value propagation graph of $\mathcal{P}$ over $\mathcal{J}_i$ such that $C(\mathbf{a}, \infty) \notin \mathcal{J}_i$.

Proof. As before, we focus on homogeneous programs all of whose limit predicates are max. We also use the following standard notions: a path in a graph is *simple* if it has no repeated nodes, and the *length* of a path is the number of edges on the path.

Let $\mathcal{P}$ be a homogeneous program as above, and let $\mathcal{J}$, $\mathcal{J}_j$, $j \geq 0$, be pseudointerpretations as above. Note that the length of the longest simple path in the value propagation graph of $\mathcal{P}$ over a pseudointerpretation is bounded by $|\mathcal{P}|$. Therefore, it is enough to show the following claim: if a number $j \in \mathbb{N}$ is such that there is no $i \in [1, j]$ for which properties 1–3 hold, then, for every node $\langle C\mathbf{a}\rangle$ in the value propagation graph $G = (V, E, \Omega)$ of $\mathcal{P}$ over $\mathcal{J}_j$ such that $\ell < \ell'$ for the pseudofacts $C(\mathbf{a}, \ell) \in \mathcal{J}_j$ and $C(\mathbf{a}, \ell') \in \mathcal{J}_{j+1}$, there is a simple path $\Pi = \langle C_1\mathbf{a}_1\rangle, \ldots, \langle C_j\mathbf{a}_j\rangle, \langle C\mathbf{a}\rangle$ in G (of length j) such that $\ell' \leq \ell^i + \Omega(\Pi^i)$ for each $i \in [1, j]$ with $C_i(\mathbf{a}_i, \ell^i) \in \mathcal{J}_j$ and $\Pi^i = \langle C_i\mathbf{a}_i\rangle, \ldots, \langle C_j\mathbf{a}_j\rangle, \langle C\mathbf{a}\rangle$. Indeed, the claim for $j = |\mathcal{P}|$ implies that there is no node $\langle C\mathbf{a}\rangle$ with $\ell < \ell'$ as there is no simple path of length more than $|\mathcal{P}|$; the lack of such a node, however, implies, together with property 2, that $\mathcal{J}_j = \mathcal{J}_{j-1}$—that is, property 1.

We next prove this claim by induction on j. If $j = 0$, then the claim is vacuous as $[1, j]$ is empty; so suppose $j > 0$ and the claim holds for $j - 1$. Consider a node $\langle C\mathbf{a}\rangle$ in the value propagation graph $G = (V, E, \Omega)$ of $\mathcal{P}$ over $\mathcal{J}_j$ such that $\ell < \ell'$ for the pseudofacts $C(\mathbf{a}, \ell) \in \mathcal{J}_j$ and $C(\mathbf{a}, \ell') \in \mathcal{J}_{j+1}$. Then, there is a rule $\rho = \varphi \to C(\mathbf{a}, s)$ in $\mathcal{P}$ applicable to $\mathcal{J}_j$ such that $\delta(\rho, \mathcal{J}_j) = C(\mathbf{a}, \ell')$—that is, such that ℓ' is ∞ if the IOP $(\psi(\rho, \mathcal{J}_j), \max s)$ is unbounded, and ℓ' is the optimal value of the IOP otherwise. Since property 2 does not hold, rule ρ is also applicable to $\mathcal{J}_{j-1}$ and so, by Proposition 6.3, there is an atom $C_j(\mathbf{a}_j, s_j)$ in φ with pseudofacts $C_j(\mathbf{a}_j, \ell_j) \in \mathcal{J}_{j-1}$ and $C_j(\mathbf{a}_j, \ell'_j) \in \mathcal{J}_j$ such that s depends on s_j and $\ell_j < \ell'_j$. Hence, ρ produces the edge $e = (\langle C_j\mathbf{a}_j\rangle, \langle C\mathbf{a}\rangle)$ in G. Moreover, by Definitions 4.7 and 6.4, $\Omega(e) = \Omega_\rho(e)$, while $\Omega_\rho(e) = \infty$ if the IOP if unbounded, $\Omega_\rho(e) = \bot$ if it is bounded and $\ell'_j = \infty$, and $\Omega_\rho(e) = \ell' - \ell'_j$ if it is bounded and $\ell'_j \in \mathbb{Z}$. In fact, $\Omega_\rho(e) = \bot$ is not possible, because stability of $\mathcal{P}$ would then imply $\Omega'(e) = \bot$ for $G' = (V, E, \Omega')$ the value propagation graph of $\mathcal{P}$ over $\mathcal{J}_{j-1}$ (which has the same set of edges as G since property 2 does not hold) and hence $\ell'_j = \ell_j = \infty$ by the definition of $\Omega'(e)$, contradicting the choice of $C_j(\mathbf{a}_j, s_j)$. In the two remaining cases (i.e., $\Omega_\rho(e) = \infty$ and $\Omega_\rho(e) = \ell' - \ell'_j$), we have $\ell'_j + \Omega(e) = \ell'_j + \Omega_\rho(e) = \ell'$.

By the inductive hypothesis for node $\langle C_j\mathbf{a}_j\rangle$, there is a simple path $\Pi_j = \langle C_1\mathbf{a}_1\rangle, \ldots, \langle C_j\mathbf{a}_j\rangle$ in G' of length $j - 1$ such that $\ell'_j \leq \ell^i_j + \Omega'(\Pi^i_j)$ for each $i \in [1, j-1]$ with $C_i(\mathbf{a}_i, \ell^i_j) \in \mathcal{J}_{j-1}$ and $\Pi^i_j = \langle C_i\mathbf{a}_i\rangle, \ldots, \langle C_j\mathbf{a}_j\rangle$. Next, we show that the path $\Pi = \Pi_j, \langle C\mathbf{a}\rangle$ in G satisfies the required conditions for $\langle C\mathbf{a}\rangle$. We first consider an arbitrary $i \in [1, j]$ and show that $\ell' \leq \ell^i + \Omega(\Pi^i)$ for the pseudofact $C_i(\mathbf{a}_i, \ell^i) \in \mathcal{J}_j$ and subpath $\Pi^i = \Pi^i_j, \langle C\mathbf{a}\rangle$. Indeed, on the one hand, if $i = j$, then $\ell' \leq \ell^i + \Omega(\Pi^i)$ because $\ell' = \ell'_j + \Omega(e)$, as shown above, $\ell'_j = \ell^i_j \leq \ell^i$ by monotonicity of $\mathcal{T}_\mathcal{P}$ (claim 2 of Corollary 4.9), and $\Omega(\Pi^i) = \Omega(e)$. On the other hand, if $i \in [1, j-1]$ then $\ell' \leq \ell^i + \Omega(\Pi^i)$ since, again, $\ell' = \ell'_j + \Omega(e)$, $\ell'_j \leq \ell^i_j + \Omega'(\Pi^i_j)$ by the inductive hypothesis, $\ell^i_j \leq \ell^i$ by monotonicity, and $\Omega'(\Pi^i_j) + \Omega(e) \leq \Omega(\Pi^i_j) + \Omega(e) = \Omega(\Pi^i)$ by stability of $\mathcal{P}$. To complete the proof, it remains to show that Π is simple. For the sake of contradiction, assume that this is not the case and Π^i is a cycle for some $i \in [1, j]$—that is, $\langle C\mathbf{a}\rangle = \langle C_i\mathbf{a}_i\rangle$. Then $\ell_i = \ell$ and, as we just proved, $\ell' \leq \ell + \Omega(\Pi^i)$. However, $\ell < \ell'$ by assumption, and so Π^i is a positive-weight cycle in G, which implies, since property 3 does not hold, that $\ell = \infty$, contradicting $\ell < \ell'$. We conclude that Π is indeed simple, as required. □

The following holds:

1. $\mathcal{I} \models \mathcal{P}$ if and only if $\mathcal{S}_\mathcal{P}(\mathcal{I}) \subseteq \mathcal{I}$, for each limit $\mathit{Datalog}^{\mathrm{agg}}_{\mathbb{Z}}$ program $\mathcal{P}$ and each object-finite limit-closed interpretation $\mathcal{I}$; and
2. $\mathcal{S}_\mathcal{P}(\mathcal{I}_1) \subseteq \mathcal{S}_\mathcal{P}(\mathcal{I}_2)$ for each semi-positive limit $\mathit{Datalog}^{\mathrm{agg}}_{\mathbb{Z}}$ program $\mathcal{P}$ and each object-finite limit-closed interpretations $\mathcal{I}_1$ and $\mathcal{I}_2$ that satisfy $\mathcal{I}_1 \subseteq \mathcal{I}_2$ and coincide on the EDB predicates.

Proof. Claim 1 follows in the same way as claim 1 of Proposition 3.1. For claim 2, note that the limit-closure operator is monotonic with respect to set inclusion, so we only need to show that so is the immediate consequence operator $\mathcal{S}$. In particular, it suffices to prove that if an object-finite limit-closed interpretation $\mathcal{I}_1$ satisfies a literal in the body of a ground rule in a semi-positive limit $\mathit{Datalog}^{\mathrm{agg}}_{\mathbb{Z}}$ program, then so does each object-finite limit-closed interpretation $\mathcal{I}_2$ containing $\mathcal{I}_1$

and coinciding with $\mathcal{I}_1$ on the EDB predicates. The statement is straightforward for nonaggregate literals, so we focus on aggregation. Consider $\mathcal{I}_1$ and $\mathcal{I}_2$ as above, and a ground aggregate atom $\beta = (s \lhd \mathsf{agg}\, C(\mathbf{t}, \text{-})\ \mathsf{group\ by}\ \mathbf{a})$ such that $\mathcal{I}_1 \models \beta$. If agg is count, then, by construction, the aggregation values in $\mathcal{I}_1$ and $\mathcal{I}_2$ do not depend on particular samples, and $\mathcal{I}_2 \models \beta$ follows since $\mathcal{I}_1 \subseteq \mathcal{I}_2$ and $\lhd \in \{\leq, <\}$. If agg is max, then, since $\mathcal{I}_1 \subseteq \mathcal{I}_2$, for each sample of C in $\mathcal{I}_1$ we can find a sample of C in $\mathcal{I}_2$ with the same or greater aggregation value, and $\mathcal{I}_2 \models \beta$ follows since $\lhd \in \{\leq, <\}$. The case when agg is sum^+ is analogous to the case of max, while the cases when agg is min or sum^- are symmetric to the cases of max and sum^+, respectively. □

Theorem 8.4. *The following holds for each predicate-nonrepeating strict rs-monotonic rs-program Q and each rs-fact δ:*

- *if δ is $A(\mathbf{a})$ with A a noncost rs-predicate, then $Q \models_{\mathsf{rs}} \delta$ if and only if $\mathcal{P}(Q) \models A'(\mathbf{a})$;*
- *if δ is $C(\mathbf{a}, \ell)$ with C a cost rs-predicate, then*
  - *if $\ell = -\infty$ then $Q \models_{\mathsf{rs}} \delta$ if and only if $\mathcal{P}(Q) \models C^{-\infty}(\mathbf{a}, 0)$ and $\mathcal{P}(Q) \not\models C'(\mathbf{a}, k)$ for all $k \in \mathbb{Z}$,*
  - *if $\ell \in \mathbb{Z}$ then $Q \models_{\mathsf{rs}} \delta$ if and only if $\mathcal{P}(Q) \models \lceil C'(\mathbf{a}, \ell)\rceil$, and*
  - *if $\ell = +\infty$ then $Q \models_{\mathsf{rs}} \delta$ if and only if $\mathcal{P}(Q) \models C'(\mathbf{a}, \infty)$.*

Proof. An rs-interpretation $\mathcal{H}$ and a limit-closed interpretation $\mathcal{I}$ *correspond* to each other if the following holds:

- $A(\mathbf{a}) \in \mathcal{H}$ with A noncost if and only if $A'(\mathbf{a}) \in \mathcal{I}$;
- $C(\mathbf{a}, -\infty) \in \mathcal{H}$ with C cost if and only if $C^{-\infty}(\mathbf{a}, 0) \in \mathcal{I}$ and $C'(\mathbf{a}, k) \notin \mathcal{I}$ for all $k \in \mathbb{Z}$;
- $C(\mathbf{a}, k) \in \mathcal{H}$ with C cost and $k \in \mathbb{Z}$ if and only if $C'(\mathbf{a}, k) \in \mathcal{I}$ and $C'(\mathbf{a}, k+1) \notin \mathcal{I}$;
- $C(\mathbf{a}, +\infty) \in \mathcal{H}$ with C cost if and only if $C'(\mathbf{a}, k) \in \mathcal{I}$ for all $k \in \mathbb{Z}$.

It is straightforward to check that each rs-interpretation has a unique corresponding limit-closed interpretation, and that $\mathcal{H}_1 \sqsubseteq_{\mathsf{rs}} \mathcal{H}_2$ if and only if $\mathcal{I}_1 \subseteq \mathcal{I}_2$ for the limit-closed interpretations $\mathcal{I}_1$ and $\mathcal{I}_2$ corresponding to rs-interpretations $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively; moreover, due to rs-atoms $(0 \leq m)$ introduced in the step from Q to Q^{sum} of the translation of a strict rs-program Q in Definition 8.3, each partial materialisation of translation $\mathcal{P}(Q)$ has a unique corresponding rs-interpretation.

Consider an rs-program Q and an rs-fact δ as in the statement of the theorem. Recall that rs-programs Q and Q^{sum} are equivalent, and hence Q^{sum} is rs-monotonic; moreover, same as Q, Q^{sum} is predicate-nonrepeating. Therefore, by construction, the materialisation $\mathcal{M}(\mathcal{P}(Q))$ is a model of the set $\mathcal{P}^{-\infty}$ of all rules (83), and it suffices to prove that $\mathcal{L}(Q^{\mathsf{sum}})$ and $\mathcal{M}(\mathcal{P}(Q))$ correspond to each other. To this end, we first show that the limit-closed interpretation $\mathcal{I}$ corresponding to $\mathcal{L}(Q^{\mathsf{sum}})$ satisfies $\mathcal{I} \subseteq \mathcal{M}(\mathcal{P}(Q))$ and then that the rs-interpretation $\mathcal{H}$ corresponding to $\mathcal{M}(\mathcal{P}(Q))$ satisfies $\mathcal{H} \sqsubseteq_{\mathsf{rs}} \mathcal{L}(Q^{\mathsf{sum}})$.

To show $\mathcal{I} \subseteq \mathcal{M}(\mathcal{P}(Q))$, consider the rs-interpretation $\mathcal{L}^\kappa$ defined as follows for each ordinal κ, where sup is the supremum in the complete lattice of rs-interpretations:

$$\mathcal{L}^0 = \emptyset, \qquad \mathcal{L}^\kappa = \mathcal{R}_{Q^{\mathsf{sum}}}(\mathcal{L}^{\kappa'}) \quad \text{if } \kappa = \kappa' + 1, \qquad \mathcal{L}^\kappa = \sup_{\kappa' < \kappa} \mathcal{L}^{\kappa'} \quad \text{if } \kappa \text{ is a limit ordinal.}$$

Consider also, for each κ, the limit-closed interpretation $\mathcal{I}^\kappa$ corresponding to $\mathcal{L}^\kappa$. It is enough to show, by transfinite induction on κ, that $\mathcal{I}^\kappa \subseteq \mathcal{M}(\mathcal{P}(Q))$ for each κ.

If $\kappa = 0$, the claim is immediate because $\mathcal{I}^0 = \mathcal{L}^0 = \emptyset$.

Let $\kappa = \kappa' + 1$ and $\mathcal{I}^{\kappa'} \subseteq \mathcal{M}(\mathcal{P}(Q))$. It suffices to show that, for each fact $\gamma \in \mathcal{I}^\kappa$, we have $\gamma \in \mathcal{M}(\mathcal{P}(Q))$. So, let $\gamma \in \mathcal{I}^\kappa$ be arbitrary. Then, there is a grounding σ of an rs-rule $\pi = \psi \rightarrow \beta$ in Q^{sum} such that $\mathcal{L}^{\kappa'} \models \psi\sigma$ and

- if $\beta\sigma = A(\mathbf{a})$ with A noncost, then $\gamma = A'(\mathbf{a})$;
- if $\beta\sigma = C(\mathbf{a}, -\infty)$ with C cost, then $\gamma = C^{-\infty}(\mathbf{a}, 0)$;

– if $\beta\sigma = C(\mathbf{a}, k)$ with C cost and $k \in \mathbb{Z}$, then $\gamma = C^{-\infty}(\mathbf{a}, 0)$ or $\gamma = C'(\mathbf{a}, k')$ for some $k' \leq k$;
– if $\beta\sigma = C(\mathbf{a}, +\infty)$ with C cost, then $\gamma = C^{-\infty}(\mathbf{a}, 0)$ or $\gamma = C'(\mathbf{a}, k)$ for some $k \in \mathbb{Z}$.

Consider the rule $\rho_\pi^{\mathbf{m},\mathbf{n}} = \varphi \to \alpha$ in $\mathcal{P}(Q)$, where $\mathbf{m}$ and $\mathbf{n}$ are the variables of π that are sent by σ to $-\infty$ and $+\infty$, respectively. Then, by construction (in particular, since $\mathcal{M}(\mathcal{P}(Q))$ is limit-closed and satisfies $\mathcal{P}^{-\infty}$—that is, rules (83)), it suffices to show that $\mathcal{M}(\mathcal{P}(Q))$ satisfies $\alpha\sigma$ (i.e., that $\alpha\sigma \in \mathcal{M}(\mathcal{P}(Q))$ if $\alpha\sigma$ is a fact or $C'(\mathbf{a}, k) \in \mathcal{M}(\mathcal{P}(Q))$ for each $k \in \mathbb{Z}$ if $\alpha\sigma = C'(\mathbf{a}, \infty)$). To this end, we claim that there exists an extension σ' of σ to all the nonlocal variables in φ that are local in ψ (all such variables are in the max and sum aggregate rs-atoms in ψ) such that $\mathcal{I}^{\kappa'} \models \varphi\sigma'$. Indeed, consider any atom α' in φ, which is the translation of an rs-atom β' in ψ. Depending on the shape of α', we have the following cases:

– if $\alpha' = A'(\mathbf{t})$ with A' object and $\beta' = A(\mathbf{t})$, then $\alpha'\sigma \in \mathcal{I}^{\kappa'}$ since $\beta'\sigma \in \mathcal{L}^{\kappa'}$;
– if α' is one of $C^{-\infty}(\mathbf{t}, 0)$, $C'(\mathbf{t}, k)$ for $k \in \mathbb{Z}$, or $C'(\mathbf{t}, \infty)$ with C cost, and $\beta' = C(\mathbf{t}, s)$ with $s\sigma = -\infty$, $s\sigma = k$ or $s\sigma = +\infty$, respectively, then $\alpha'\sigma \in \mathcal{I}^{\kappa'}$ since $\beta'\sigma \in \mathcal{L}^{\kappa'}$;
– if α' is a comparison atom and β' is built-in, then $\alpha'\sigma$ evaluates to true since so does $\beta'\sigma$;
– if $\alpha' = (s \leq \mathtt{count}\, C^{-\infty}(\mathbf{t}, \text{-})\ \mathtt{group\ by}\ \mathbf{t}')$ and $\beta' = (s \overset{\text{rs}}{=} \mathtt{count}\, C(\mathbf{t}, \text{-})\ \mathtt{group\ by}\ \mathbf{t}')$ with $s\sigma \in \mathbb{Z}$, then $\mathcal{I}^{\kappa'} \models \alpha'\sigma$ since $\mathcal{L}^{\kappa'} \models_{\text{rs}} \beta'\sigma$ and $\mathcal{I}^{\kappa'}$ satisfies $\mathcal{P}^{-\infty}$;
– if $\alpha' = (0 \leq 0)$ and $\beta' = (s \overset{\text{rs}}{=} \mathtt{max}\, C(\mathbf{t}, \text{-})\ \mathtt{group\ by}\ \mathbf{t}')$ with $s\sigma = -\infty$, then $\alpha'\sigma$ evaluates to true;
– if $\alpha' = C'(\mathbf{t}, s)$ and $\beta' = (s \overset{\text{rs}}{=} \mathtt{max}\, C(\mathbf{t}, \text{-})\ \mathtt{group\ by}\ \mathbf{t}')$ with $s\sigma \in \mathbb{Z}$, then $\alpha'\sigma' \in \mathcal{I}^{\kappa'}$, for σ' the extension of σ to the variables in $\mathbf{t}$ such that $C(\mathbf{t}\sigma', s\sigma) \in \mathcal{L}^{\kappa'}$, which exists since $\mathcal{L}^{\kappa'} \models_{\text{rs}} \beta'\sigma$;
– if $\alpha' = C'(\mathbf{t}, \infty)$ and $\beta' = (s \overset{\text{rs}}{=} \mathtt{max}\, C(\mathbf{t}, \text{-})\ \mathtt{group\ by}\ \mathbf{t}')$ with $s\sigma = +\infty$, then $\mathcal{I}^{\kappa'} \models \alpha'\sigma'$ (i.e., $C'(\mathbf{t}\sigma', k) \in \mathcal{I}^{\kappa'}$ for all $k \in \mathbb{Z}$), for σ' the extension of σ to the variables in $\mathbf{t}$ such that $C(\mathbf{t}\sigma', +\infty) \in \mathcal{L}^{\kappa'}$, which exists since $\mathcal{L}^{\kappa'} \models_{\text{rs}} \beta'\sigma$;
– if $\alpha' = (s \leq \mathtt{sum}^{+}\, C'(\mathbf{t}, \text{-})\ \mathtt{group\ by}\ \mathbf{t}')$ and $\beta' = (s \overset{\text{rs}}{=} \mathtt{sum}\, C(\mathbf{t}, \text{-})\ \mathtt{group\ by}\ \mathbf{t}')$ with $s\sigma \in \mathbb{Z}$, then $\mathcal{I}^{\kappa'} \models \alpha'\sigma$ since $\mathcal{L}^{\kappa'} \models_{\text{rs}} \beta'\sigma$ and $\mathcal{I}^{\kappa'}$ corresponds to $\mathcal{L}^{\kappa'}$;
– if $\alpha' = C'(\mathbf{t}, \infty)$ and $\beta' = (s \overset{\text{rs}}{=} \mathtt{sum}\, C(\mathbf{t}, \text{-})\ \mathtt{group\ by}\ \mathbf{t}')$ with $s\sigma = +\infty$, then $\mathcal{I}^{\kappa'} \models \alpha'\sigma'$ (i.e., $C'(\mathbf{t}\sigma', k) \in \mathcal{I}^{\kappa'}$ for all $k \in \mathbb{Z}$), for σ' the extension of σ to the local variables of β' such that $C(\mathbf{t}\sigma', +\infty) \in \mathcal{L}^{\kappa'}$, which exists since $\mathcal{L}^{\kappa'} \models_{\text{rs}} \beta'\sigma$ and $\mathbf{t}$ has only finitely many ground instances.

Note that the case $\alpha' = (0 < 0)$ and $\beta' = (s \overset{\text{rs}}{=} \mathtt{count}\, C(\mathbf{t}, \text{-})\ \mathtt{group\ by}\ \mathbf{t}')$ with $s\sigma \in \{-\infty, +\infty\}$ is not possible, because $\mathcal{L}^{\kappa'} \models_{\text{rs}} \beta'\sigma$ and count ranges over natural numbers (count cannot output $+\infty$ because $\mathbf{t}$ has only finitely many possible ground instances over constants in $\mathcal{L}^{\kappa'}$). Also, the case $\alpha' = (0 < 0)$ and $\beta' = (s \overset{\text{rs}}{=} \mathtt{sum}\, C(\mathbf{t}, \text{-})\ \mathtt{group\ by}\ \mathbf{t}')$ with $s\sigma = -\infty$ is not possible, because $\mathcal{L}^{\kappa'} \models_{\text{rs}} \beta'\sigma$ while the cost domain of sum is over $\mathbb{N}$.

We conclude that there is an extension σ' of σ such that $\mathcal{I}^{\kappa'} \models \alpha'\sigma'$ for every atom α' in φ, and hence $\mathcal{I}^{\kappa'} \models \varphi\sigma'$. Therefore, $\mathcal{S}_{\mathcal{P}(Q)}(\mathcal{I}^{\kappa'})$ satisfies $\alpha\sigma' = \alpha\sigma$. Furthermore, $\mathcal{I}^{\kappa'} \subseteq \mathcal{M}(\mathcal{P}(Q))$ by the inductive hypothesis, and $\mathcal{M}(\mathcal{P}(Q))$ is a model of $\mathcal{P}(Q)$; therefore, $\mathcal{S}_{\mathcal{P}(Q)}(\mathcal{I}^{\kappa'}) \subseteq \mathcal{M}(\mathcal{P}(Q))$. Hence $\mathcal{M}(\mathcal{P}(Q))$ satisfies $\alpha\sigma$, which implies $\mathcal{I}^{\kappa} \subseteq \mathcal{M}(\mathcal{P}(Q))$, as required.

Finally, let κ be a limit ordinal and $\mathcal{I}^{\kappa'} \subseteq \mathcal{M}(\mathcal{P}(Q))$ for all $\kappa' < \kappa$. Then, $\mathcal{I}^{\kappa} \subseteq \mathcal{M}(\mathcal{P}(Q))$ since $\mathcal{I}^{\kappa}$ is the union of all $\mathcal{I}^{\kappa'}$ with $\kappa' < \kappa$ by the correspondence of $\sqsubseteq_{\text{rs}}$ and $\subseteq$, and the definition of the correspondence relation between rs-interpretations and limit-closed interpretations.

To show $\mathcal{H} \sqsubseteq_{\text{rs}} \mathcal{L}(Q^{\text{sum}})$ for the rs-interpretation $\mathcal{H}$ corresponding to $\mathcal{M}(\mathcal{P}(Q))$, consider, for each ordinal κ, the limit-closed interpretation $\mathcal{K}^{\kappa}$ defined as follows:

$$\mathcal{K}^0 = \emptyset, \qquad \mathcal{K}^{\kappa} = \mathcal{S}_{\mathcal{P}^{-\infty}}(\mathcal{S}_{\mathcal{P}(Q)}(\mathcal{K}^{\kappa'})) \quad \text{if } \kappa = \kappa' + 1, \qquad \mathcal{K}^{\kappa} = \bigcup_{\kappa' < \kappa} \mathcal{K}^{\kappa'} \quad \text{if } \kappa \text{ is a limit ordinal.}$$

In other words, each $\mathcal{K}^{\kappa}$ is the closure of the partial materialisation $\mathcal{M}_1^{\kappa}$ of $\mathcal{P}(Q)$ with the rules in $\mathcal{P}^{-\infty}$. Since $\mathcal{P}(Q)$ is positive and includes $\mathcal{P}^{-\infty}$, $\mathcal{M}(\mathcal{P}(Q)) = \mathcal{K}^{\omega_1}$. Hence, it suffices to show, by transfinite induction on κ, that $\mathcal{H}^\kappa \sqsubseteq_{\mathrm{rs}} \mathcal{L}(\mathcal{Q}^{\mathsf{sum}})$ for each κ, where $\mathcal{H}^\kappa$ is the rs-interpretation corresponding to $\mathcal{K}^\kappa$.

If $\kappa = 0$, the claim is immediate because $\mathcal{H}^0 = \mathcal{K}^0 = \emptyset$.

Let $\kappa = \kappa' + 1$ and $\mathcal{H}^{\kappa'} \sqsubseteq_{\mathrm{rs}} \mathcal{L}(\mathcal{Q}^{\mathsf{sum}})$. It suffices to show that, for each rs-fact $\delta \in \mathcal{H}^\kappa$, there is some $\delta' \in \mathcal{L}(\mathcal{Q}^{\mathsf{sum}})$ such that $\delta \sqsubseteq_{\mathrm{rs}} \delta'$. So, let $\delta \in \mathcal{H}^\kappa$ be arbitrary. Then, there is a grounding σ of a rule $\rho = \varphi \to \alpha$ in $\mathcal{P}(\mathcal{Q})$ such that $\mathcal{K}^{\kappa'} \models \varphi\sigma$, and δ is $A(\mathbf{a})$, $C(\mathbf{a}, -\infty)$, $C(\mathbf{a}, k)$ for $k \in \mathbb{Z}$, or $C(\mathbf{a}, +\infty)$ whenever $\alpha\sigma$ is $A'(\mathbf{a})$, $C^{-\infty}(\mathbf{a}, 0)$, $C'(\mathbf{a}, k)$, or $C'(\mathbf{a}, \infty)$, respectively (note that ρ cannot come from $\mathcal{P}^{-\infty}$ since rules (83) of $\mathcal{P}^{-\infty}$ derive only facts corresponding to rs-facts that are strictly subsumed by rs-facts in $\mathcal{H}^\kappa$ with respect to $\sqsubseteq_{\mathrm{rs}}$). Consider a minimal limit-closed sub-interpretation $\mathcal{I}'$ of $\mathcal{K}^{\kappa'}$ such that $\mathcal{I}' \models \mathcal{P}^{-\infty}$ and $\mathcal{I}' \models \varphi\sigma$, the rs-interpretation $\mathcal{H}'$ corresponding to $\mathcal{I}'$, and the rs-rule $\pi = \psi \to \beta$ in $\mathcal{Q}^{\mathsf{sum}}$ such that $\rho = \rho_\pi^{\mathbf{m},\mathbf{n}}$ for cost variables $\mathbf{m}$ and $\mathbf{n}$ of π. Note that $\delta = \beta\sigma'$, where σ' is the substitution obtained from σ by assigning $-\infty$ to all $\mathbf{m}$ and $+\infty$ to all $\mathbf{n}$. We claim that $\mathcal{H}' \models_{\mathrm{rs}} \psi\sigma'$. Indeed, consider any rs-atom β' in ψ and its translation $\alpha_{\beta'}$ in φ. Depending of the shape of β', we have the following cases:

- if $\beta' = A(\mathbf{t})$ with A noncost, then $\alpha_{\beta'} = A'(\mathbf{t})$ and so $\beta'\sigma' \in \mathcal{H}'$ since $\alpha_{\beta'}\sigma \in \mathcal{I}'$ and $\beta'\sigma' = \beta'\sigma$;
- if $\beta' = C(\mathbf{t}, s)$ with C cost and $s\sigma' = -\infty$, then $\alpha_{\beta'} = C^{-\infty}(\mathbf{t}, 0)$ and so $\beta'\sigma' = C(\mathbf{t}\sigma, -\infty) \in \mathcal{H}'$ since $\alpha_{\beta'}\sigma \in \mathcal{I}'$;
- if $\beta' = C(\mathbf{t}, s)$ with C cost and $s\sigma' \in \mathbb{Z}$, then $\alpha_{\beta'} = C'(\mathbf{t}, s)$ and so $\beta'\sigma' \in \mathcal{H}'$ since $\alpha_{\beta'}\sigma \in \mathcal{I}'$, interpretation $\mathcal{I}'$ is a minimal limit-closed subset of $\mathcal{K}^\kappa$ such that $\mathcal{I}' \models \varphi\sigma$, $\mathcal{Q}$ is predicate-nonrepeating and hence C' does not occur in any other atom in φ, and $\beta'\sigma' = \beta'\sigma$;
- if $\beta' = C(\mathbf{t}, s)$ with C cost and $s\sigma' = +\infty$, then $\alpha_{\beta'} = C'(\mathbf{t}, \infty)$ and so $\beta'\sigma' = C(\mathbf{t}\sigma, +\infty) \in \mathcal{H}'$ since $\mathcal{I}' \models \alpha_{\beta'}\sigma$ and hence $C'(\mathbf{t}\sigma, k) \in \mathcal{I}'$ for all $k \in \mathbb{Z}$;
- if β' is a built-in rs-atom, then $\beta'\sigma'$ evaluates to true since $\alpha_{\beta'}\sigma$ evaluates to true by construction and Proposition 8.2;
- if $\beta' = (s \overset{\mathrm{rs}}{=} \mathsf{count}\, C(\mathbf{t}, \text{-})\ \mathsf{group\ by}\ \mathbf{t}')$ with $s\sigma' \in \mathbb{Z}$, then $\alpha_{\beta'} = (s \le \mathsf{count}\, C^{-\infty}(\mathbf{t}, \text{-})\ \mathsf{group\ by}\ \mathbf{t}')$ and so $\mathcal{H}' \models_{\mathrm{rs}} \beta'\sigma'$ since $\mathcal{I}' \models \alpha_{\beta'}\sigma$, $\mathcal{I}' \models \mathcal{P}^{-\infty}$, interpretation $\mathcal{I}'$ is a minimal limit-closed subset of $\mathcal{K}^\kappa$ such that $\mathcal{I}' \models \varphi\sigma$, and C' does not occur in any other atom in φ;
- if $\beta' = (s \overset{\mathrm{rs}}{=} \mathsf{max}\, C(\mathbf{t}, \text{-})\ \mathsf{group\ by}\ \mathbf{t}')$ with $s\sigma' = -\infty$ then $\alpha_{\beta'} = (0 \le 0)$ and so $\mathcal{H}' \models_{\mathrm{rs}} \beta'\sigma'$ since $C^{-\infty}$ and C' do not occur in any atom in φ—as interpretation $\mathcal{I}'$ is a minimal limit-closed subset of $\mathcal{K}^\kappa$ such that $\mathcal{I}' \models \varphi\sigma$, it thus contains no facts over $C^{-\infty}$ or C', which, in turn, implies that $\mathcal{H}'$ contains no facts over C;
- if $\beta' = (s \overset{\mathrm{rs}}{=} \mathsf{max}\, C(\mathbf{t}, \text{-})\ \mathsf{group\ by}\ \mathbf{t}')$ with $s\sigma' \in \mathbb{Z}$, then $\alpha_{\beta'} = C'(\mathbf{t}, s)$ and so $\mathcal{H}' \models_{\mathrm{rs}} \beta'\sigma'$ since $\alpha_{\beta'}\sigma \in \mathcal{I}'$, interpretation $\mathcal{I}'$ is a minimal limit-closed subset of $\mathcal{K}^\kappa$ such that $\mathcal{I}' \models \varphi\sigma$, and C' does not occur in any other atom in φ;
- if $\beta' = (s \overset{\mathrm{rs}}{=} \mathsf{max}\, C(\mathbf{t}, \text{-})\ \mathsf{group\ by}\ \mathbf{t}')$ with $s\sigma' = +\infty$, then $\alpha_{\beta'} = C'(\mathbf{t}, \infty)$ and so $\mathcal{H}' \models_{\mathrm{rs}} \beta'\sigma'$ since $\mathcal{I}' \models \alpha_{\beta'}\sigma$ and hence $C'(\mathbf{t}\sigma, k) \in \mathcal{I}'$ for all $k \in \mathbb{Z}$, which implies that $C(\mathbf{t}\sigma', +\infty) \in \mathcal{H}'$;
- if $\beta' = (s \overset{\mathrm{rs}}{=} \mathsf{sum}\, C(\mathbf{t}, \text{-})\ \mathsf{group\ by}\ \mathbf{t}')$ with $s\sigma' \in \mathbb{Z}$, then $\alpha_{\beta'} = (s \le \mathsf{sum}^+\, C'(\mathbf{t}, \text{-})\ \mathsf{group\ by}\ \mathbf{t}')$ and so $\mathcal{H}' \models_{\mathrm{rs}} \beta'\sigma'$ since $\mathcal{I}' \models \alpha_{\beta'}\sigma$, interpretation $\mathcal{I}'$ is a minimal limit-closed subset of $\mathcal{K}^\kappa$ such that $\mathcal{I}' \models \varphi\sigma$, C' does not occur in any other atom in φ, and since $\mathcal{H}'$ has no negative numbers or $-\infty$ in rs-facts over C (which, once again, is a consequence of the minimality of $\mathcal{I}'$);
- if $\beta' = (s \overset{\mathrm{rs}}{=} \mathsf{sum}\, C(\mathbf{t}, \text{-})\ \mathsf{group\ by}\ \mathbf{t}')$ with $s\sigma' = +\infty$, then $\alpha_{\beta'} = C'(\mathbf{t}, \infty)$ and so $\mathcal{H}' \models_{\mathrm{rs}} \beta'\sigma'$ since $\mathcal{I}' \models \alpha_{\beta'}\sigma$ and hence $C'(\mathbf{t}\sigma, k) \in \mathcal{I}'$ for all $k \in \mathbb{Z}$, which implies that $C(\mathbf{t}\sigma', +\infty) \in \mathcal{H}'$.

Note also that the cases of $\beta' = (s \overset{\mathrm{rs}}{=} \mathsf{count}\, C(\mathbf{t}, \text{-})\ \mathsf{group\ by}\ \mathbf{t}')$ with $s\sigma' \in \{-\infty, +\infty\}$ and $\beta' = (s \overset{\mathrm{rs}}{=} \mathsf{sum}\, C(\mathbf{t}, \text{-})\ \mathsf{group\ by}\ \mathbf{t}')$ with $s\sigma' = -\infty$ are not possible, because then $\alpha_{\beta'} = (0 < 0)$ and hence $\mathcal{I}' \not\models \alpha_{\beta'}\sigma$.

So, $\mathcal{H}' \models_{\mathrm{rs}} \beta'\sigma'$ for each rs-atom β' in ψ and hence $\mathcal{H}' \models_{\mathrm{rs}} \psi\sigma'$. Therefore, $\delta \in \mathcal{R}_{\mathcal{Q}^{\mathsf{sum}}}(\mathcal{H}')$.

Since $\mathcal{I}' \subseteq \mathcal{K}^{\kappa'}$ by construction, $\mathcal{H}'$ corresponds to $\mathcal{I}'$, and $\mathcal{H}^{\kappa'}$ to $\mathcal{K}^{\kappa'}$, we also have $\mathcal{H}' \sqsubseteq_{\mathsf{rs}} \mathcal{H}^{\kappa'}$, and hence $\mathcal{R}_{Q^{\mathsf{sum}}}(\mathcal{H}') \sqsubseteq_{\mathsf{rs}} \mathcal{R}_{Q^{\mathsf{sum}}}(\mathcal{H}^{\kappa'})$ by rs-monotonicity of Q^{sum}. Also, $\mathcal{R}_{Q^{\mathsf{sum}}}(\mathcal{H}^{\kappa'}) \sqsubseteq_{\mathsf{rs}} \mathcal{L}(Q^{\mathsf{sum}})$ since $\mathcal{H}^{\kappa'} \sqsubseteq_{\mathsf{rs}} \mathcal{L}(Q^{\mathsf{sum}})$ by the inductive hypothesis, and $\mathcal{L}(Q^{\mathsf{sum}})$ is the least fixpoint if $\mathcal{R}_Q$. Finally, since $\delta \in \mathcal{R}_{Q^{\mathsf{sum}}}(\mathcal{H}')$ and $\mathcal{R}_{Q^{\mathsf{sum}}}(\mathcal{H}') \sqsubseteq_{\mathsf{rs}} \mathcal{R}_{Q^{\mathsf{sum}}}(\mathcal{H}^{\kappa'}) \sqsubseteq_{\mathsf{rs}} \mathcal{L}(Q^{\mathsf{sum}})$, there is some $\delta' \in \mathcal{L}(Q^{\mathsf{sum}})$ such that $\delta \sqsubseteq_{\mathsf{rs}} \delta'$, as required.

Finally, let κ be a limit ordinal and $\mathcal{H}^{\kappa'} \sqsubseteq_{\mathsf{rs}} \mathcal{L}(Q^{\mathsf{sum}})$ for all $\kappa' < \kappa$. It suffices to show that, for each rs-fact $\delta \in \mathcal{H}^{\kappa}$, there is an rs-fact $\delta' \in \mathcal{L}(Q^{\mathsf{sum}})$ such that $\delta \sqsubseteq_{\mathsf{rs}} \delta'$. Since $\mathcal{H}^{\kappa}$ corresponds to $\mathcal{K}^{\kappa}$ and $\mathcal{K}^{\kappa}$ is the union of all $\mathcal{K}^{\kappa'}$ for $\kappa' < \kappa$, we have the following cases for δ:

- if $\delta = A(\mathbf{a})$ with A noncost, then $A'(\mathbf{a}) \in \mathcal{K}^{\kappa}$ and hence $A'(\mathbf{a}) \in \mathcal{K}^{\kappa'}$ for some $\kappa' < \kappa$; therefore, $\delta \in \mathcal{H}^{\kappa'}$, and hence $\delta \in \mathcal{L}(Q^{\mathsf{sum}})$ by the inductive hypothesis;
- if $\delta = C(\mathbf{a}, -\infty)$ with C cost, then $C^{-\infty}(\mathbf{a}, 0) \in \mathcal{K}^{\kappa}$, and the inductive hypothesis implies the existence of $\delta' \in \mathcal{L}(Q^{\mathsf{sum}})$ with $\delta \sqsubseteq_{\mathsf{rs}} \delta'$ analogously to the previous case;
- if $\delta = C(\mathbf{a}, k)$ with C cost and $k \in \mathbb{Z}$, then $C'(\mathbf{a}, k) \in \mathcal{K}^{\kappa}$ and the argument proceeds as before;
- if $\delta = C(\mathbf{a}, +\infty)$ with C cost, then $C'(\mathbf{a}, k) \in \mathcal{K}^{\kappa}$ for all $k \in \mathbb{Z}$, and we distinguish two cases:
  - if there is $\kappa' < \kappa$ such that $C'(\mathbf{a}, k) \in \mathcal{K}^{\kappa'}$ for all $k \in \mathbb{Z}$, then $\delta \in \mathcal{H}^{\kappa'}$ and hence $\delta \in \mathcal{L}(Q^{\mathsf{sum}})$;
  - otherwise, for every $k \in \mathbb{Z}$ there are $\kappa'_k < \kappa$ and $k' \geq k$ such that $C(\mathbf{a}, k') \in \mathcal{H}^{\kappa'_k}$ and $\delta \in \mathcal{L}(Q^{\mathsf{sum}})$ follows since, by the inductive hypothesis, $\mathcal{H}^{\kappa'_k} \sqsubseteq_{\mathsf{rs}} \mathcal{L}(Q^{\mathsf{sum}})$ for all k. $\square$