

Foundations of Declarative Data Analysis Using Limit Datalog Programs

Mark Kaminski, Bernardo Cuenca Grau, Egor V. Kostylev, Boris Motik and Ian Horrocks
Department of Computer Science, University of Oxford, UK

Abstract

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

1 Introduction

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

An essential ingredient of declarative data analysis is an efficient language that can capture the relevant tasks, and Datalog is a prime candidate since it supports recursion. Apart from recursion, however, data analysis usually also requires integer arithmetic to capture quantitative aspects of data (e.g., the length of a shortest path). Research on combining the two dates back to the ’90s [Mumick *et al.*, 1990;

Kemp and Stuckey, 1991; Beeri *et al.*, 1991; Van Gelder, 1992; Consens and Mendelzon, 1993; Ganguly *et al.*, 1995; Ross and Sagiv, 1997], and is currently experiencing a revival [Faber *et al.*, 2011; Mazuran *et al.*, 2013]. This extensive body of work, however, focuses primarily on integrating recursion and arithmetic with *aggregate functions* in a coherent semantic framework, where technical difficulties arise due to nonmonotonicity of aggregates. Surprisingly little is known about the computational properties of integrating recursion with arithmetic, apart from that a straightforward combination is undecidable [Dantsin *et al.*, 2001]. Undecidability also carries over to the above formalisms and practical Datalog-based systems such as BOOM [Alvaro *et al.*, 2010], DeALS [Shkapsky *et al.*, 2016], Myria [Wang *et al.*, 2015], Socialite [Seo *et al.*, 2015], Overlog [Loo *et al.*, 2009], Dyna [Eisner and Filardo, 2011], and Yedalog [Chin *et al.*, 2015].

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

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

$$\rightarrow sp(v_0, 0) \quad (1)$$

$$sp(x, m) \wedge edge(x, y, n) \rightarrow sp(y, m + n) \quad (2)$$

If these rules and a dataset entail a fact $sp(v, k)$, then the cost of a shortest path from v_0 to v is at most k ; hence, $sp(v, k')$ holds for each $k' \geq k$ since the cost of a shortest path is also at most k' . Rule (2) intuitively says that, if x is reachable from v_0 with cost at most m and $\langle x, y \rangle$ is an edge of cost n , then v' is reachable from v_0 with cost at most $m + n$. This is different from *Datalog_ℤ*, where there is no implicit semantic con-

nection between $sp(v, k)$ and $sp(v, k')$, and such semantic connections allow us to prove decidability of limit $Datalog_{\mathbb{Z}}$. We provide a direct semantics for limit predicates based on Herbrand interpretations, but we also show that this semantics can be axiomatised in standard $Datalog_{\mathbb{Z}}$. Our formalism can thus be seen as a fragment of $Datalog_{\mathbb{Z}}$, from which it inherits well-understood properties such as monotonicity and existence of a least fixpoint model [Dantsin *et al.*, 2001].

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

The proofs of all results are given in the appendix of this paper.

2 Preliminaries

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

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

Semantics A (*Herbrand*) *interpretation* I is a (not necessarily finite) set of facts. Such I *satisfies* a ground atom α , written $I \models \alpha$, if (i) α is a standard atom and evaluating the arithmetic functions in α produces a fact in I , or (ii) α is a

comparison atom and evaluating the arithmetic functions and comparisons produces *true*. The notion of satisfaction is extended to conjunctions of ground atoms, rules, and programs as in first-order logic, where each rule is universally quantified. If $I \models \mathcal{P}$, then I is a *model* of program \mathcal{P} ; and \mathcal{P} *entails* a fact α , written $\mathcal{P} \models \alpha$, if $I \models \alpha$ holds whenever $I \models \mathcal{P}$.

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

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

3 Limit Programs

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

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

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

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

In the rest of this paper we make three simplifying assumptions. First, numeric atoms occurring in a rule body are function-free (but comparison atoms and the head can contain arithmetic functions). Second, each numeric variable in

a rule occurs in at most one standard body atom. Third, distinct rules in a program use different variables. The third assumption is clearly w.l.o.g. because all variables are universally quantified so their names are immaterial. Moreover, the first two assumptions are w.l.o.g. as well since, for each rule, there exists a logically equivalent rule that satisfies these assumptions. In particular, we can replace an atom such as $A(\mathbf{t}, m_1 + m_2)$ with conjunction

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

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

$$\rightarrow I(0) \quad I(m) \rightarrow I(m+1) \quad I(m) \rightarrow I(m-1)$$

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

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

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

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

$$\begin{aligned} \rightarrow Z(0) \quad Z(m) \rightarrow Z(m+1) \quad Z(m) \rightarrow Z(m-1) \\ B(\mathbf{x}, m) \wedge Z(n) \wedge (m \leq n) \rightarrow B(\mathbf{x}, n) \\ \text{for each min predicate } B \text{ in } \mathcal{P} \\ B(\mathbf{x}, m) \wedge Z(n) \wedge (n \leq m) \rightarrow B(\mathbf{x}, n) \\ \text{for each max predicate } B \text{ in } \mathcal{P} \end{aligned}$$

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

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

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

In Section 1 we have shown that limit $Datalog_{\mathbb{Z}}$ can compute the cost of shortest paths in a graph. We next present further examples of data analysis tasks that our formalism

can handle. In all examples, we assume that all objects in the input are arranged in an arbitrary linear order using facts $first(a_1), next(a_1, a_2), \dots, next(a_{n-1}, a_n)$; we use this order to simulate aggregation by means of recursion.

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

$$\rightarrow tw(a_s) \quad (3)$$

$$follows(x, y') \wedge first(y) \rightarrow nt(x, y, 0) \quad (4)$$

$$follows(x, y) \wedge first(y) \wedge tw(y) \rightarrow nt(x, y, 1) \quad (5)$$

$$nt(x, y', m) \wedge next(y', y) \rightarrow nt(x, y, m) \quad (6)$$

$$nt(x, y', m) \wedge next(y', y) \wedge follows(x, y) \wedge tw(y) \rightarrow nt(x, y, m+1) \quad (7)$$

$$th(x, m) \wedge nt(x, y, n) \wedge (m \leq n) \rightarrow tw(x) \quad (8)$$

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

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

$$node(x) \rightarrow np(x, x, 1) \quad (9)$$

$$node(x) \wedge node(y) \wedge first(z) \rightarrow np'(x, y, z, 0) \quad (10)$$

$$edge(x, z) \wedge np(z, y, m) \wedge first(z) \rightarrow np'(x, y, z, m) \quad (11)$$

$$np'(x, y, z', m) \wedge next(z', z) \rightarrow np'(x, y, z, m) \quad (12)$$

$$np'(x, y, z', m) \wedge next(z', z) \wedge edge(x, z) \wedge np(z, y, n) \rightarrow np'(x, y, z, m+n) \quad (13)$$

$$np'(x, y, z, m) \rightarrow np(x, y, m) \quad (14)$$

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

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

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

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

4 Fixpoint Characterisation of Entailment

Programs are often grounded to eliminate variables and thus simplify the presentation. In limit *Datalog*_ℤ, however, numeric variables range over integers, so a grounding can be infinite. Thus, we first specialise the notion of a grounding.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5 Decidability of Entailment: Limit-Linearity

We now start our investigation of the computational properties of limit *Datalog*_ℤ. Theorem 15 bounds the cardinality of

the closure of a semi-ground program, but it does not bound the magnitude of the integers occurring in limit facts; in fact, integers can be arbitrarily large. Moreover, due to multiplication, checking rule applicability requires solving nonlinear inequalities over integers, which is undecidable.

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

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

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

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

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

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

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

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

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

Lemma 19. *Let $\varphi = \forall \mathbf{x} \exists \mathbf{y}. \bigvee_{i=1}^n \psi_i$ be a Presburger sentence where each ψ_i is a conjunction of possibly negated atoms of size at most k mentioning at most ℓ variables, a is the maximal magnitude of an integer in φ , and $m = |\mathbf{x}|$. Then, φ is valid if and only if φ is valid over models where each integer variable assumes a value whose magnitude is bounded by $(2^{O(\ell \log \ell)} \cdot a^{k\ell})^{n2^\ell \cdot O(m^4)}$.*

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

Theorem 20. *For \mathcal{P} a semi-ground limit-linear program, \mathcal{D} a dataset, and α a fact, $\mathcal{P} \cup \mathcal{D} \models \alpha$ if and only if a pseudo-model J of $\mathcal{P} \cup \mathcal{D}$ exists where $J \models \alpha$, $|J| \leq |\mathcal{P} \cup \mathcal{D}|$, and the magnitude of each integer in J is bounded polynomially by the largest magnitude of an integer in $\mathcal{P} \cup \mathcal{D}$, exponentially by $|\mathcal{P}|$, and double-exponentially by $\max_{r \in \mathcal{P}} \|r\|$.*

By Theorem 20, the following nondeterministic algorithm decides $\mathcal{P} \models \alpha$:

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

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

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

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

6 Tractability of Entailment: Stability

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

6.1 Cyclic Dependencies in Limit Programs

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

Example 22. *Let \mathcal{P}_c contain facts $A(0)$ and $B(0)$, and rules $A(m) \rightarrow B(m)$ and $B(m) \rightarrow A(m+1)$, where A and B are max predicates. Applying the first rule copies the value*

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

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

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

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

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

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

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

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

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

6.2 Stable Programs

As Example 22 shows, the presence of a positive-weight cycle in $G_{\mathcal{P}}^J$ does not imply the divergence of all atoms corresponding to the nodes in the cycle. This is because the weight of

such a cycle may decrease after certain rule applications and so it is no longer positive.

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

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

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

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

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

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

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

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

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

Lemmas 26 and 27 imply the following theorem.

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

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

Algorithm 1 Entailment for Semi-Ground Stable Programs

Input: semi-ground stable program \mathcal{P} , fact α **Output:** true if $\mathcal{P} \models \alpha$

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

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

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

6.3 Type-Consistent Programs

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

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

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

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

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

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

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

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

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

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

7 Conclusion and Future Work

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

Acknowledgments

We thank Christoph Haase for explaining to us his results on Presburger arithmetic and semi-linear sets, which were key to some of our technical results. Our work has also benefited from discussions with Michael Benedikt. This research was supported by the Royal Society and the EPSRC projects DBOnto, MaSI³, and ED³.

References

[Alvaro *et al.*, 2010] Peter Alvaro, Tyson Condie, Neil Conway, Khaled Elmeleegy, Joseph M. Hellerstein, and Rus-

- sell Sears. BOOM analytics: exploring data-centric, declarative programming for the cloud. In *EuroSys*. ACM, 2010.
- [Beeri *et al.*, 1991] Catriel Beeri, Shamim A. Naqvi, Oded Shmueli, and Shalom Tsur. Set constructors in a logic database language. *J. Log. Program.*, 10(3&4), 1991.
- [Berman, 1980] Leonard Berman. The complexity of logical theories. *Theor. Comput. Sci.*, 11, 1980.
- [Byrd *et al.*, 1987] Richard H. Byrd, Alan J. Goldman, and Miriam Heller. Recognizing unbounded integer programs. *Oper. Res.*, 35(1), 1987.
- [Chin *et al.*, 2015] Brian Chin, Daniel von Dincklage, Vuk Ercegovic, Peter Hawkins, Mark S. Miller, Franz Josef Och, Christopher Olston, and Fernando Pereira. Yedalog: Exploring knowledge at scale. In *SNAPL*, 2015.
- [Chistikov and Haase, 2016] Dmitry Chistikov and Christoph Haase. The taming of the semi-linear set. In *ICALP*, 2016.
- [Consens and Mendelzon, 1993] Mariano P. Consens and Alberto O. Mendelzon. Low complexity aggregation in GraphLog and Datalog. *Theor. Comput. Sci.*, 116(1), 1993.
- [Damaggio *et al.*, 2012] Elio Damaggio, Alin Deutsch, and Victor Vianu. Artifact systems with data dependencies and arithmetic. *ACM Trans. Database Syst.*, 37(3):22:1–22:36, 2012.
- [Dantsin *et al.*, 2001] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3), 2001.
- [Eisner and Filardo, 2011] Jason Eisner and Nathaniel Wesley Filardo. Dyna: Extending datalog for modern AI. In *Datalog*, 2011.
- [Faber *et al.*, 2011] Wolfgang Faber, Gerald Pfeifer, and Nicola Leone. Semantics and complexity of recursive aggregates in answer set programming. *Artif. Intell.*, 175(1), 2011.
- [Ganguly *et al.*, 1995] Sumit Ganguly, Sergio Greco, and Carlo Zaniolo. Extrema predicates in deductive databases. *J. Comput. Syst. Sci.*, 51(2), 1995.
- [Grädel, 1988] Erich Grädel. Subclasses of presburger arithmetic and the polynomial-time hierarchy. *Theor. Comput. Sci.*, 56, 1988.
- [Haase, 2014] Christoph Haase. Subclasses of Presburger arithmetic and the weak EXP hierarchy. In *CSL-LICS*, 2014.
- [Hougardy, 2010] Stefan Hougardy. The Floyd-Warshall algorithm on graphs with negative cycles. *Inf. Process. Lett.*, 110(8-9), 2010.
- [Kannan, 1987] Ravi Kannan. Minkowski’s convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987.
- [Kemp and Stuckey, 1991] David B. Kemp and Peter J. Stuckey. Semantics of logic programs with aggregates. In *ISLP*, 1991.
- [Koutsos and Vianu, 2017] Adrien Koutsos and Victor Vianu. Process-centric views of data-driven business artifacts. *J. Comput. System Sci.*, 86:82–107, 2017.
- [Loo *et al.*, 2009] Boon Thau Loo, Tyson Condie, Minos N. Garofalakis, David E. Gay, Joseph M. Hellerstein, Petros Maniatis, Raghu Ramakrishnan, Timothy Roscoe, and Ion Stoica. Declarative networking. *Commun. ACM*, 52(11), 2009.
- [Markl, 2014] Volker Markl. Breaking the chains: On declarative data analysis and data independence in the big data era. *PVLDB*, 7(13), 2014.
- [Mazuran *et al.*, 2013] Mirjana Mazuran, Edoardo Serra, and Carlo Zaniolo. Extending the power of datalog recursion. *VLDB J.*, 22(4), 2013.
- [Mumick *et al.*, 1990] Inderpal Singh Mumick, Hamid Pirahesh, and Raghu Ramakrishnan. The magic of duplicates and aggregates. In *VLDB*, pages 264–277, 1990.
- [Papadimitriou, 1981] Christos H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4), 1981.
- [Ross and Sagiv, 1997] Kenneth A. Ross and Yehoshua Sagiv. Monotonic aggregation in deductive databases. *J. Comput. System Sci.*, 54(1), 1997.
- [Schöning, 1997] Uwe Schöning. Complexity of presburger arithmetic with fixed quantifier dimension. *Theory Comput. Syst.*, 30(4), 1997.
- [Seo *et al.*, 2015] Jiwon Seo, Stephen Guo, and Monica S. Lam. Socialite: An efficient graph query language based on datalog. *IEEE Trans. Knowl. Data Eng.*, 27(7), 2015.
- [Shkapsky *et al.*, 2016] Alexander Shkapsky, Mohan Yang, Matteo Interlandi, Hsuan Chiu, Tyson Condie, and Carlo Zaniolo. Big data analytics with datalog queries on Spark. In *SIGMOD*. ACM, 2016.
- [Van Gelder, 1992] Allen Van Gelder. The well-founded semantics of aggregation. In *PODS*, 1992.
- [von zur Gathen and Sieveking, 1978] Joachim von zur Gathen and Malte Sieveking. A bound on solutions of linear integer equalities and inequalities. *Proc. AMS*, 72(1), 1978.
- [Wang *et al.*, 2015] Jingjing Wang, Magdalena Balazinska, and Daniel Halperin. Asynchronous and fault-tolerant recursive datalog evaluation in shared-nothing engines. *PVLDB*, 8(12), 2015.

A Proofs for Section 3

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

Proof. We prove the claim by presenting a reduction of the halting problem for deterministic Turing machines on the empty tape. Let M be an arbitrary deterministic Turing machine with finite alphabet Γ containing the blank symbol \sqcup , the finite set of states S containing the initial state s and the halting state h , and transition function $\delta : S \times \Gamma \rightarrow S \times \Gamma \times \{L, R\}$. We assume that M works on a tape that is infinite to the right, that it starts with the empty tape and the head positioned on the leftmost cell, and that it never moves the head off the left edge of the tape.

We encode each time point $i \geq 0$ using an integer 2^i , and we index tape positions using zero-based integers; thus, at time i , each position $j \geq 2^i$ is necessarily empty, so we can encode a combination of a time point i and tape position j with $0 \leq j < 2^i$ using a single integer $2^i + j$. We use this idea to encode the state of the execution of M using the following facts:

- $Num(k)$ is true for each positive number k ;
- $Time(k)$ is true if $k = 2^i$ and so k encodes a time point i ;
- $Tape(a, 2^i + j)$ says that symbol a occupies position j of the tape at time i , and it will be defined for each $0 \leq j < 2^i$;
- $Pos(2^i + j)$ says that the head points to position j of the tape at time i ;
- $State(q, 2^i)$ says that the machine is in state q at time i ; and
- $Halts$ is a propositional variable saying that the machine has halted.

We next give a $Datalog_{\mathbb{Z}}$ program \mathcal{P}_M that simulates the behaviour of M on the empty tape. We represent each alphabet symbol $a \in \Gamma$ using an object constant a , and we represent each state $q \in S$ using an object constant q . Furthermore, we abbreviate $(s \leq t) \wedge (t < u)$ as $(s \leq t < u)$. Finally, we abbreviate conjunction $(s \leq t) \wedge (t \leq s)$ as $(s \doteq t)$, and disjunction $(s < t) \vee (t > s)$ as $(s \neq t)$. Strictly speaking, disjunctions are not allowed in rule bodies; however, each rule with a disjunction in the body of the form $\varphi \wedge (s \neq t) \rightarrow \alpha$ corresponds to rules $\varphi \wedge (s < t) \rightarrow \alpha$ and $\varphi \wedge (t < s) \rightarrow \alpha$, so we use the former form for the sake of clarity. With these considerations in mind, program \mathcal{P}_M contains rules (15)–(24).

$$\rightarrow Num(1) \quad (15)$$

$$Num(x) \rightarrow Num(x + 1) \quad (16)$$

$$\rightarrow Time(1) \quad (17)$$

$$Time(x) \rightarrow Time(x + x) \quad (18)$$

$$\rightarrow Tape(\sqcup, 1) \quad (19)$$

$$\rightarrow Pos(1) \quad (20)$$

$$\rightarrow State(s, 1) \quad (21)$$

$$State(h, x) \rightarrow Halts \quad (22)$$

$$Time(x) \wedge Tape(v, y) \wedge Pos(z) \wedge (x \leq y < x + x) \wedge (x \leq z < x + x) \wedge (y \neq z) \rightarrow Tape(v, x + y) \quad (23)$$

$$Time(x) \wedge Num(u) \wedge (x + x + x \leq u + u < x + x + x + x) \rightarrow Tape(\sqcup, u) \quad (24)$$

Moreover, for each alphabet symbol $a \in \Gamma$ and all states $q, q' \in S$ such that $\delta(q, a) = (q', a', D)$ where $D \in \{L, R\}$ is a direction, \mathcal{P}_M contains rules (25)–(28).

$$Time(x) \wedge State(q, x) \wedge Tape(a, y) \wedge Pos(y) \wedge (x \leq y < x + x) \rightarrow Tape(a', x + y) \quad (25)$$

$$Time(x) \wedge State(q, x) \wedge Tape(a, y) \wedge Pos(y) \wedge (x \leq y < x + x) \rightarrow State(q', x + x) \quad (26)$$

$$Time(x) \wedge State(q, x) \wedge Tape(a, y) \wedge Pos(y) \wedge Num(u) \wedge (x \leq y < x + x) \wedge (x + y \doteq u + 1) \rightarrow Pos(u) \quad \text{if } D = L \quad (27)$$

$$Time(x) \wedge State(q, x) \wedge Tape(a, y) \wedge Pos(y) \wedge Num(u) \wedge (x \leq y < x + x) \wedge (x + y + 1 \doteq u) \rightarrow Pos(u) \quad \text{if } D = R \quad (28)$$

Rules (15)–(16) initialise Num so that it holds of all positive integers, and rules (17)–(18) initialise $Time$ so that it holds for each integer $k = 2^i$. Rules (19)–(21) initialise the state of the M at time $i = 0$. Rule (22) derives $Halts$ if at any point the Turing machine enters the halting state h . The remaining rules encode the evolution of the state of M , and they are based on the following idea: if variable x encodes a time point i using value 2^i , then variable y encodes a position j for time point i if $x \leq y < x + x$ holds; moreover, for such y , position j at time point $i + 1$ is encoded as $2^{i+1} + j = 2^i + 2^i + j$ and can be obtained as $x + y$, and the encodings of positions $j - 1$ and $j + 1$ can be obtained as $x + y - 1$ and $x + y + 1$, respectively. Since our goal is to prove undecidability by just using $+$, we simulate subtraction by looking for a value u such that $x + y = u + 1$. With these observations in mind, one can see that rule (23) copies the unaffected part of the tape from time point i to time point

$i + 1$. Moreover, rule (24) pads the tape by filling each location j with $1.5 \times 2^i \leq j < 2 \times 2^i$ with the blank symbol; since division is not supported in our language, we express this condition as $3 \times 2^i \leq 2 \times j < 2 \times 2 \times 2^i$. Finally, rule (25) updates the tape at the position of the head; rule (26) updates the state; and rules (27) and (28) move the head left and right, respectively. Consequently, we have $\mathcal{P}_M \models \text{Halts}$ if and only if M halts on the empty tape. \square

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

Proof. Let \mathcal{P} be an arbitrary limit program. Without loss of generality, we construct a program \mathcal{P}' containing only max predicates. For each min predicate A , let A' be a fresh max predicate uniquely associated with A . We construct \mathcal{P}' from \mathcal{P} by modifying each rule $r \in \mathcal{P}$ as follows:

1. if $h(r) = A(\mathbf{t}, s)$ where A is a min predicate, replace the head of r with $A'(\mathbf{t}, -s)$;
2. for each body atom $A(\mathbf{t}, n) \in \text{sb}(r)$ where A is a min predicate and n is a variable, replace the atom with $A'(\mathbf{t}, m)$ where m is a fresh variable, and replace all other occurrences of n in the rule with $-m$;
3. for each body atom $A(\mathbf{t}, k) \in \text{sb}(r)$ where A is a min predicate and k is an integer, replace the atom with $A'(\mathbf{t}, -k)$.

Finally, if $\alpha = A(\mathbf{a}, k)$ is a min fact, let $\alpha' = A(\mathbf{a}, -k)$; otherwise, let $\alpha' = \alpha$. Now consider an arbitrary interpretation I , and let I' be the interpretation obtained from I by replacing each min fact $A(\mathbf{a}, k)$ with $A'(\mathbf{a}, -k)$; it is straightforward to see that $I \models \mathcal{P}$ if and only if $I' \models \mathcal{P}'$, and that $I \models \alpha$ if and only if $I' \models \alpha'$. Thus, $\mathcal{P} \models \alpha$ if and only if $\mathcal{P}' \models \alpha'$. \square

B Proofs for Section 4

We use the standard notion of substitutions—sort-compatible partial mappings of variables to constants. For φ a formula and σ a substitution, $\varphi\sigma$ is the formula obtained by replacing each free variable x in φ on which σ is defined with $\sigma(x)$.

Proposition B.1. *For each semi-ground rule $r = \varphi \rightarrow \alpha$, each pseudo-interpretation J , and each mapping σ of the variables of r to integers, σ is an integer solution to $\mathcal{C}(r, J)$ if and only if $J \models \varphi\sigma$.*

Proof. (\Rightarrow) Assume that σ is an integer solution to $\mathcal{C}(r, J)$. We consider each atom $\beta \in \varphi$ and show that $J \models \beta\sigma$ holds.

- If β is a comparison atom, the claim is straightforward due to $\beta \in \mathcal{C}(r, J)$.
- If β is an object atom or an ordinary numeric atom, β is ground and we have $\beta \in J$ and $J \models \beta$; otherwise, $(0 < 0) \in \mathcal{C}(r, J)$ would hold and so σ could not be a solution to $\mathcal{C}(r, J)$.
- If β is a max atom $B(\mathbf{b}, s)$, since $(0 < 0) \notin \mathcal{C}(r, J)$, either $B(\mathbf{b}, \ell) \in J$ for some integer $\ell \in \mathbb{Z}$ and $(s \leq \ell) \in \mathcal{C}(r, J)$, or $B(\mathbf{b}, \infty) \in J$. In the former case, since σ is a solution to $\mathcal{C}(r, J)$, we have $s\sigma \leq \ell$, and, since B is a max predicate, $J \models B(\mathbf{b}, s\sigma)$ holds. In the latter case, $J \models B(\mathbf{b}, s\sigma)$ holds due to $\{B(\mathbf{b}, s\sigma)\} \sqsubseteq \{B(\mathbf{b}, \infty)\}$.
- If β is a min atom, the proof is analogous to the previous case.

The proof of the (\Leftarrow) direction is analogous and we omit it for the sake of brevity. \square

Definition B.2. *Given a limit-closed interpretation I and a program \mathcal{P} , let*

$$\mathbf{I}_{\mathcal{P}}(I) = \{\gamma \mid \bigwedge_i \alpha_i \rightarrow \beta \text{ is a ground instance of a rule in } \mathcal{P} \text{ such that } I \models \bigwedge_i \alpha_i \text{ and } \gamma \text{ is a fact such that } \{\gamma\} \sqsubseteq \{\beta\}\}.$$

Let $\mathbf{I}_{\mathcal{P}}^0 = \emptyset$, let $\mathbf{I}_{\mathcal{P}}^n = \mathbf{I}_{\mathcal{P}}(\mathbf{I}_{\mathcal{P}}^{n-1})$ for $n > 0$, and let $\mathbf{I}_{\mathcal{P}}^\infty = \bigcup_{n \geq 0} \mathbf{I}_{\mathcal{P}}^n$.

Lemma B.3. *For each program \mathcal{P} , operator $\mathbf{I}_{\mathcal{P}}$ is monotonic w.r.t. \sqsubseteq ; moreover, for I an interpretation, $I \models \mathcal{P}$ if and only if $\mathbf{I}_{\mathcal{P}}(I) = I$, and $I \models \mathcal{P}$ implies $\mathbf{I}_{\mathcal{P}}^\infty \subseteq I$.*

Proof. Operator $\mathbf{I}_{\mathcal{P}}$ is the standard immediate consequence operator of Datalog, but applied to the program \mathcal{P}' obtained by extending \mathcal{P} with the rules from Section 3 encoding the semantics of limit predicates. Thus, all claims of this lemma hold in the usual way [Dantsin *et al.*, 2001]. \square

Lemma B.4. *For each limit-closed interpretation I and the corresponding pseudo-interpretation J , and for each semi-ground limit program \mathcal{P} , interpretation $\mathbf{I}_{\mathcal{P}}(I)$ corresponds to the pseudo-interpretation $\mathbf{T}_{\mathcal{P}}(J)$.*

Proof. It suffices to show that, for each fact α , the following claims hold:

1. if α is an object fact or ordinary numeric fact, then $\alpha \in \mathbf{I}_{\mathcal{P}}(I)$ if and only if $\alpha \in \mathbf{T}_{\mathcal{P}}(J)$;
2. if α is a limit fact of the form $A(\mathbf{a}, k)$ where k is an integer, then $\alpha \in \mathbf{I}_{\mathcal{P}}(I)$ if and only if $\{\alpha\} \sqsubseteq \mathbf{T}_{\mathcal{P}}(J)$; and
3. if α is a limit fact of the form $A(\mathbf{a}, \infty)$, then $\{A(\mathbf{a}, k) \mid k \in \mathbb{Z}\} \sqsubseteq \mathbf{I}_{\mathcal{P}}(I)$ if and only if $\alpha \in \mathbf{T}_{\mathcal{P}}(J)$.

(Claim 1) Consider an arbitrary object fact α of the form $A(\mathbf{a})$; the proof for ordinary numeric facts is analogous.

- Assume $\alpha \in \mathbf{I}_{\mathcal{P}}(I)$. Then, a rule $r = \varphi \rightarrow \alpha \in \mathcal{P}$ and a grounding σ of r exist such that $I \models \varphi\sigma$; the head of r must be α since \mathcal{P} is semi-ground. But then, $J \models \varphi\sigma$ holds as well, so Proposition B.1 ensures that σ is a solution to $\mathcal{C}(r, J)$; moreover, $\text{hd}(r, J) = \alpha$, and thus we have $\alpha \in \mathbf{T}_{\mathcal{P}}(J)$.
- Assume $\alpha \in \mathbf{T}_{\mathcal{P}}(J)$. Then, there exist a rule $r = \varphi \rightarrow \alpha \in \mathcal{P}$ and an integer solution σ to $\mathcal{C}(r, J)$. Proposition B.1 then ensures $J \models \varphi\sigma$, and so $I \models \varphi\sigma$ holds as well. Thus, we have $\alpha \in \mathbf{I}_{\mathcal{P}}(I)$.

(Claim 2) Consider an arbitrary max fact α of the form $A(\mathbf{a}, k)$; the proof for a min fact is analogous.

- Assume $\alpha \in \mathbf{I}_{\mathcal{P}}(I)$. Then, a rule $r = \varphi \rightarrow A(\mathbf{a}, s) \in \mathcal{P}$ and a grounding σ of r exist such that $I \models \varphi\sigma$ and $\alpha = A(\mathbf{a}, s\sigma)$. But then, $J \models \varphi\sigma$ holds as well, so Proposition B.1 ensures that σ is a solution to $\mathcal{C}(r, J)$; moreover, $s\sigma \leq \text{opt}(r, J)$, and therefore we have $\{\alpha\} \subseteq \{\text{hd}(r, J)\} = \{A(\mathbf{a}, \text{opt}(r, J))\} \subseteq \mathbf{T}_{\mathcal{P}}(J)$.
- Assume $\{\alpha\} \subseteq \mathbf{T}_{\mathcal{P}}(J)$. Then, there exist a rule $r = \varphi \rightarrow A(\mathbf{a}, s) \in \mathcal{P}$ and an integer solution σ to $\mathcal{C}(r, J)$ such that $\{\alpha\} \subseteq \{\text{hd}(r, J)\} = \{A(\mathbf{a}, s\sigma)\}$ where $s\sigma = \text{opt}(r, J)$. Proposition B.1 then ensures $J \models \varphi\sigma$, and so $I \models \varphi\sigma$ holds as well. Thus, $\gamma \in \mathbf{I}_{\mathcal{P}}(I)$ holds for each fact γ with $\{\gamma\} \subseteq \{A(\mathbf{a}, s\sigma)\}$, so we have $\alpha \in \mathbf{I}_{\mathcal{P}}(I)$.

(Claim 3) Consider an arbitrary max fact α of the form $A(\mathbf{a}, \infty)$; the proof for a min fact is analogous. In the following, let $S = \{A(\mathbf{a}, k) \mid k \in \mathbb{Z}\}$.

- Assume $S \subseteq \mathbf{I}_{\mathcal{P}}(I)$. Program \mathcal{P} contains only finitely many rules, so the infinitely many facts of S in $\mathbf{I}_{\mathcal{P}}(I)$ are produced by a rule $r = \varphi \rightarrow A(\mathbf{a}, s) \in \mathcal{P}$ and an infinite sequence $(\sigma_i)_{i \geq 0}$ of groundings of r such that, for each i , we have $I \models \varphi\sigma_i$ and $s\sigma_i < s\sigma_{i+1}$. But then, $J \models \varphi\sigma_i$, so Proposition B.1 ensures that σ_i satisfies $\mathcal{C}(r, J)$ for each $i \geq 0$; therefore, $\text{opt}(r, J) = \infty$ and $\alpha \in \mathbf{T}_{\mathcal{P}}(J)$ holds.
- Assume $A(\mathbf{a}, \infty) \in \mathbf{T}_{\mathcal{P}}(J)$. Then, a rule $r = \varphi \rightarrow A(\mathbf{a}, s) \in \mathcal{P}$ exists such that $\text{opt}(r, J) = \infty$, so an infinite sequence $(\sigma_i)_{i \geq 0}$ of solutions to $\mathcal{C}(r, J)$ exists such that $s\sigma_i < s\sigma_{i+1}$ for each $i \geq 0$. Proposition B.1 ensures $J \models \varphi\sigma_i$ for each $i \geq 0$, and so $I \models \varphi\sigma_i$ as well. Thus, for each $k \in \mathbb{Z}$, some $i \geq 0$ exists such that $k \leq s\sigma_i$, and therefore we have $\{A(\mathbf{a}, k)\} \subseteq \{A(\mathbf{a}, s\sigma_i)\}$; consequently, $S \subseteq \mathbf{I}_{\mathcal{P}}(I)$ holds. \square

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

Proof. Immediate from Lemmas B.3 and B.4. \square

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

Proof. By inductively applying Lemma B.4, for each $n \geq 0$, the limit-closed interpretation $\mathbf{I}_{\mathcal{P}}^n$ clearly corresponds to the pseudo-interpretation $\mathbf{T}_{\mathcal{P}}^n$. Thus, $\mathbf{I}_{\mathcal{P}}^{\infty}$ and $\mathbf{T}_{\mathcal{P}}^{\infty}$ also correspond on all object and ordinary numeric facts. Now consider an arbitrary n -ary max predicate A and a tuple of $n-1$ objects \mathbf{a} , and for $M = \{k \mid A(\mathbf{a}, k) \in \mathbf{I}_{\mathcal{P}}^{\infty}\}$ consider the following cases.

- $M = \emptyset$. Then, for each $n \geq 0$ and each $k \in \mathbb{Z}$, we have $A(\mathbf{a}, k) \notin \mathbf{I}_{\mathcal{P}}^n$, which implies $A(\mathbf{a}, k) \notin \mathbf{T}_{\mathcal{P}}^n$ and $A(\mathbf{a}, \infty) \notin \mathbf{T}_{\mathcal{P}}^n$. Finally, $\mathbf{T}_{\mathcal{P}}^{\infty}$ is the least (w.r.t. \sqsubseteq) fixpoint of $\mathbf{T}_{\mathcal{P}}$, so $A(\mathbf{a}, k) \notin \mathbf{T}_{\mathcal{P}}^{\infty}$ and $A(\mathbf{a}, \infty) \notin \mathbf{T}_{\mathcal{P}}^{\infty}$ holds as well.
- There exists $\ell = \max M$. Then, there exists $n \geq 0$ such that $A(\mathbf{a}, \ell) \in \mathbf{I}_{\mathcal{P}}^n$, and $A(\mathbf{a}, \ell') \notin \mathbf{I}_{\mathcal{P}}^n$ for each $\ell' > \ell$ and $m \geq 0$; but then, $A(\mathbf{a}, \ell) \in \mathbf{T}_{\mathcal{P}}^n$, and $A(\mathbf{a}, \ell') \notin \mathbf{T}_{\mathcal{P}}^m$ for each $\ell' > \ell$ and $m \geq 0$; finally, $\mathbf{T}_{\mathcal{P}}^{\infty}$ is the least (w.r.t. \sqsubseteq) fixpoint of operator $\mathbf{T}_{\mathcal{P}}$, so $A(\mathbf{a}, \ell) \in \mathbf{T}_{\mathcal{P}}^{\infty}$ holds.
- $M = \mathbb{Z}$. Then, for each $k \in \mathbb{Z}$, there exists $n \geq 0$ such that $A(\mathbf{a}, k) \in \mathbf{I}_{\mathcal{P}}^n$, and so $\mathbf{T}_{\mathcal{P}}^n \models A(\mathbf{a}, k)$ holds; but then, $A(\mathbf{a}, \infty) \in \mathbf{T}_{\mathcal{P}}^n$ holds as well.

Analogous reasoning holds for min predicates, so $\mathbf{I}_{\mathcal{P}}^{\infty}$ corresponds to $\mathbf{T}_{\mathcal{P}}^{\infty}$. But then, the first and the third claim of this theorem follow straightforwardly from Lemma B.3. Moreover, pseudo-interpretations contain at most one fact per combination of a limit predicate and a tuple of objects (of corresponding arity), and program \mathcal{P} is semi-ground so each rule in \mathcal{P} produces at most one fact in $\mathbf{T}_{\mathcal{P}}^{\infty}$, which implies the second claim of this theorem. \square

C Proofs for Section 5

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

Proof. We present a reduction from Hilbert's tenth problem, which is to determine whether, given a polynomial $P(x_1, \dots, x_n)$ over variables x_1, \dots, x_n , equation $P(x_1, \dots, x_n) = 0$ has integer solutions. It is well known that the problem remains undecidable even if the solutions must be nonnegative integers, so we use that variant in this proof. For each such polynomial P , let \mathcal{P}_P be the program containing rules (29)–(30) for A a unary min predicate and B a nullary object predicate; it is obvious that $\mathcal{P}_P \models B$ if and only if $P(x_1, \dots, x_n) = 0$ has a nonnegative integer solution.

$$\rightarrow A(0) \tag{29}$$

$$\bigwedge_{i=1}^n A(x_i) \wedge (P(x_1, \dots, x_n) \leq 0) \wedge (P(x_1, \dots, x_n) \geq 0) \rightarrow B \quad (30)$$

Moreover, rule (30) is applicable to $J = \{A(0)\}$ if and only if $P(x_1, \dots, x_n) = 0$ has a nonnegative integer solution. \square

Although Presburger arithmetic does not have propositional variables, these can clearly be axiomatised using numeric variables. Hence, in the rest of this section we use propositional variables in Presburger formulas for the sake of clarity.

Definition C.1. For each n -ary object predicate A , each $(n+1)$ -ary ordinary numeric predicate B , each $(n+1)$ -ary limit predicate C , each n -tuple of objects \mathbf{a} , and each integer k , let $\text{def}_{A\mathbf{a}}$, $\text{def}_{B\mathbf{a}k}$, $\text{def}_{C\mathbf{a}}$ and $\text{fin}_{C\mathbf{a}}$ be distinct propositional variables, and let $\text{val}_{C\mathbf{a}}$ a distinct integer variable. Moreover, let \preceq_C be \leq (resp. \geq) if C is a max (resp. min) predicate.

For \mathcal{P} a semi-ground program, $\text{Pres}(\mathcal{P}) = \bigwedge_{r \in \mathcal{P}} \text{Pres}(r)$ is the Presburger formula where $\text{Pres}(r) = \forall \mathbf{y}. r'$ for \mathbf{y} all numeric variables in r and r' is obtained by replacing each atom α in r with its encoding $\text{Pres}(\alpha)$ defined as follows:

- $\text{Pres}(\alpha) = \alpha$ if α is a comparison atom;
- $\text{Pres}(\alpha) = \text{def}_{A\mathbf{a}}$ if α is an object atom of the form $A(\mathbf{a})$;
- $\text{Pres}(\alpha) = \text{def}_{B\mathbf{a}k}$ if α is an ordinary numeric atom of the form $B(\mathbf{a}, k)$; and
- $\text{Pres}(\alpha) = \text{def}_{C\mathbf{a}} \wedge (\neg \text{fin}_{C\mathbf{a}} \vee s \preceq_C \text{val}_{C\mathbf{a}})$ if α is a limit atom of the form $C(\mathbf{a}, s)$.

Let J be a pseudo-interpretation, and let μ be an assignment of Boolean and integer variables. Then, J corresponds to μ if all of the following conditions hold for all A, B, C , and \mathbf{a} as specified above, for each integer $k \in \mathbb{Z}$:

- $\mu(\text{def}_{A\mathbf{a}}) = \text{true}$ if and only if $A(\mathbf{a}) \in J$;
- $\mu(\text{def}_{B\mathbf{a}k}) = \text{true}$ if and only if $B(\mathbf{a}, k) \in J$;
- $\mu(\text{def}_{C\mathbf{a}}) = \text{true}$ if and only if $C(\mathbf{a}, \infty) \in J$ or there exists $\ell \in \mathbb{Z}$ such that $C(\mathbf{a}, \ell) \in J$;
- $\mu(\text{fin}_{C\mathbf{a}}) = \text{true}$ and $\mu(\text{val}_{C\mathbf{a}}) = k$ if and only if $C(\mathbf{a}, k) \in J$.

Note that k in Definition C.1 ranges over all integers (which excludes ∞), $\mu(\text{val}_{C\mathbf{a}})$ is an equal to some integer k , and J is a pseudo-interpretation and thus cannot contain both $C(\mathbf{a}, \infty)$ and $C(\mathbf{a}, k)$; thus, $C(\mathbf{a}, \infty) \in J$ implies $\mu(\text{fin}_{C\mathbf{a}}) = \text{false}$.

Also note that each assignment μ corresponds to precisely one J ; however, each J corresponds to infinitely many assignments μ since Definition C.1 does not restrict the value of variables other than $\text{def}_{A\mathbf{a}}$, $\text{def}_{B\mathbf{a}k}$, $\text{def}_{C\mathbf{a}}$, $\text{fin}_{C\mathbf{a}}$, and $\text{val}_{C\mathbf{a}}$. Moreover, two assignments corresponding to the same pseudo-interpretation may differ on the value of $\text{val}_{C\mathbf{a}}$ if $\text{fin}_{C\mathbf{a}}$ is set to *false* in both assignments, and they can differ on the values of $\text{fin}_{C\mathbf{a}}$ and $\text{val}_{C\mathbf{a}}$ if $\text{def}_{C\mathbf{a}}$ is set to *false* in both assignments.

Lemma C.2. Let J be a pseudo-interpretation and let μ be a variable assignment such that J corresponds to μ . Then,

1. $J \models \alpha$ if and only if $\mu \models \text{Pres}(\alpha)$ for each ground atom α , and
2. $J \models r$ if and only if $\mu \models \text{Pres}(r)$ for each semi-ground rule r .

Proof. (Claim 1) We consider all possible forms of α .

- α is a comparison atom. Then, the truth of α is independent from J so the claim is immediate.
- $\alpha = A(\mathbf{a})$ is an object fact. Then, $\text{Pres}(\alpha) = \text{def}_{A\mathbf{a}}$, and $\mu(\text{def}_{A\mathbf{a}}) = \text{true}$ if and only if $\text{def}_{A\mathbf{a}} \in J$, so the claim holds.
- $\alpha = B(\mathbf{a}, k)$ is an ordinary numeric fact. The proof is analogous to the case of object facts.
- $\alpha = C(\mathbf{a}, k)$ is a limit fact. If $J \models \alpha$, then either $C(\mathbf{a}, \infty) \in J$ or an integer ℓ exists such that $C(\mathbf{a}, \ell) \in J$ and $k \preceq_C \ell$; either way, $\mu(\text{def}_{C\mathbf{a}}) = \text{true}$ holds; moreover, $\mu(\text{fin}_{C\mathbf{a}}) = \text{false}$ holds in the former and $\mu(\text{val}_{C\mathbf{a}}) = \ell$ holds in the latter case; thus, $\mu \models \text{Pres}(r)$ clearly holds. The converse direction is analogous so we omit it for the sake of brevity.

(Claim 2) Let r be an arbitrary semi-ground rule, and let I be the limit-closed interpretation corresponding to J . By definition, $J \models r$ if and only if $I \models r$, and the latter is equivalent to $I \models r'$ for each ground instance r' of r by the semantics of universal quantification in first-order logic; but then, the latter claim is equivalent to $J \models r'$ for each ground instance r' of r .

Now note that, by construction, we have $\text{Pres}(\beta\sigma) = \text{Pres}(\beta)\sigma$ for each semi-ground atom β and each grounding σ , and thus $\text{Pres}(r\sigma) = \text{Pres}(r)\sigma$. Finally, groundings of r can be equivalently seen as variable assignments to universally quantified numeric variables in $\text{Pres}(r)$, so Claim 2 follows immediately from Claim 1. \square

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

Proof. Lemma C.2 immediately implies that $\mathcal{P} \models \alpha$ if and only if the sentence $\varphi_0 = \forall \mathbf{x}. \text{Pres}(\alpha) \vee \neg \text{Pres}(\mathcal{P})$ is valid, where \mathbf{x} contains all variables $\text{def}_{A\mathbf{a}}, \text{def}_{B\mathbf{a}k}, \text{def}_{C\mathbf{a}}, \text{fin}_{C\mathbf{a}}$, and $\text{val}_{C\mathbf{a}}$ occurring in $\text{Pres}(\mathcal{P})$ or $\text{Pres}(\alpha)$. Clearly, $|\mathbf{x}|$ is polynomially bounded by $\|\mathcal{P}\| + \|\alpha\|$, and the magnitude of each integer in φ_0 is bounded by the maximum magnitude of an integer in \mathcal{P} and α . Let φ_1 be the sentence obtained from φ_0 by converting each top-level conjunct of $\text{Pres}(\mathcal{P})$ into form $\forall \mathbf{y}_i. \chi_i$ where χ_i is in CNF. Formulae φ_0 and φ_1 are equivalent, and φ_1 is of the form

$$\varphi_1 = \forall \mathbf{x}. \text{Pres}(\alpha) \vee \neg \bigwedge_{i=1}^n \forall \mathbf{y}_i. \bigwedge_{j=1}^{\ell_i} \chi_i^j,$$

where $n = |\mathcal{P}|$ and, for each rule $r_i \in \mathcal{P}$, integer ℓ_i is exponentially bounded by $\|r_i\|$, and $\|\chi_i^j\|$ and $|\mathbf{y}_i|$ are linearly bounded by $\|r_i\|$. By moving all quantifiers to the front of the formula and pushing negations inwards, we finally obtain formula

$$\varphi_2 = \forall \mathbf{x} \exists \mathbf{y}. \text{Pres}(\alpha) \vee \bigvee_{i=1}^n \bigvee_{j=1}^{\ell_i} \psi_i^j,$$

where $\mathbf{y} = \bigcup_{i=1}^n \mathbf{y}_i$ and each ψ_i^j is the negation-normal form of $\neg \chi_i^j$. Formula φ_2 is of the required form, $|\mathbf{y}|$ is bounded polynomially by $\|\mathcal{P}\|$, number $\sum_{i=1}^n \ell_i$ is bounded polynomially by $n = |\mathcal{P}|$ and exponentially by $\max_{r \in \mathcal{P}} \|r\|$, and $\|\psi_i^j\|$ is bounded linearly by $\|\mathcal{P}\|$. \square

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

Proof. Let $m' = |\mathbf{y}|$. Each ψ_i can be seen as a system of linear inequalities $S_i = (A_i \mathbf{x} \leq \mathbf{c}_i)$ such that $|\mathbf{x}| \leq \ell$, $|\mathbf{c}_i| \leq k$, and where the maximal magnitude of all numbers in A and \mathbf{c}_i is bounded by a^k . By Proposition 3 of Chistikov and Haase [2016] (adapted from the work by von zur Gathen and Sieveking [1978]), the set of solutions to S_i can be represented by a semi-linear set $\bigcup_{i' \in N_i} L(\mathbf{b}_{i'}, P_{i'})$ where $\mathbf{b}_{i'} \in \mathbb{Z}^\ell$, $P_{i'} \subseteq \mathbb{Z}^\ell$, $|N_i| \leq 2^\ell$, and the magnitude of all integers in $\mathbf{b}_{i'}$ and $P_{i'}$ is bounded by $2^{O(\ell \log \ell)} \cdot a^{k\ell}$. Consequently, disjunction $\bigvee_{i=1}^n \psi_i$ corresponds to a semi-linear set $\bigcup_{j \in M} L(\mathbf{b}_j, P_j)$ where $\mathbf{b}_j \in \mathbb{Z}^{m+m'}$, $P_j \subseteq \mathbb{Z}^{m+m'}$, $|M| \leq n2^\ell$, and the magnitude of each integer in \mathbf{b}_j and P_j is still bounded by $2^{O(\ell \log \ell)} \cdot a^{k\ell}$. Formula $\exists \mathbf{y}. \bigvee_{i=1}^n \psi_i$ then corresponds to the projection of $\bigcup_{j \in M} L(\mathbf{b}_j, P_j)$ on the variables in \mathbf{x} , which is a semi-linear set of the form $\bigcup_{j \in M} L(\mathbf{b}'_j, P'_j)$ where each $\mathbf{b}'_j \in \mathbb{Z}^m$ is a projection of \mathbf{b}_j on \mathbf{x} , and each $P'_j \subseteq \mathbb{Z}^m$ is a projection of P_j on \mathbf{x} . Now, Theorem 21 by Chistikov and Haase [2016] implies that the satisfying assignments to the formula $\varphi' = \neg \exists \mathbf{y}. \bigvee_{i=1}^n \psi_i$ can be represented as a semi-linear set $\bigcup_{j' \in M'} L(\mathbf{c}_{j'}, Q_{j'})$ where the magnitude of each integer in each $\mathbf{c}_{j'}$ and $Q_{j'}$ is bounded by $b = (2^{O(\ell \log \ell)} \cdot a^{k\ell})n^{2^\ell \cdot O(m^4)}$. Since φ' has a satisfying assignment if and only if it has a satisfying assignment involving only numbers from some $\mathbf{c}_{j'}$, it follows that φ' is satisfiable if and only if it is satisfiable over models where the absolute value of every integer variable is bounded by b . This implies the claim of this lemma since φ is valid if and only if φ' is unsatisfiable. \square

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

Proof. The \Leftarrow direction is trivial. For the \Rightarrow direction assume that $\mathcal{P} \cup \mathcal{D} \not\models \alpha$ holds, and let \mathcal{D}' be obtained from \mathcal{D} by removing each fact that does not unify with an atom in \mathcal{P} or α ; clearly, we have $\mathcal{P} \cup \mathcal{D}' \not\models \alpha$. Let $\varphi = \forall \mathbf{x} \exists \mathbf{y}. \bigvee_{i=1}^n \psi_i$ be the Presburger sentence from Lemma 18 for $\mathcal{P} \cup \mathcal{D}'$ and α . Sentence φ is not valid and it satisfies the following conditions.

- Number $m = |\mathbf{x}|$ is polynomial in $\|\mathcal{P} \cup \mathcal{D}'\|$, which, in turn, is bounded by $|\mathcal{P} \cup \mathcal{D}'| \cdot \max_{r \in \mathcal{P} \cup \mathcal{D}'} \|r\|$. Moreover, \mathcal{D}' contains only facts that unify with atoms in \mathcal{P} and α , so m can be bounded further, namely linearly in the product cs , for $c = |\mathcal{P}|$ and $s = \max_{r \in \mathcal{P}} \|r\|$.
- Number n is linear in the product of c and 2^s .
- The size, and hence the number ℓ of variables in each ψ_i , are linear in s .

Let a be the maximal magnitude of an integer in $\mathcal{P} \cup \mathcal{D}'$ (and thus in φ as well). By Lemma 19, an assignment μ exists such that $\mu \models \varphi$ and the magnitude of each integer variable is bounded by $b = (2^{O(s \log s)} \cdot a^{O(s^2)})^{O(c \cdot 2^s)} \cdot 2^{O(s)} \cdot O((cs)^4)$. Clearly, b is polynomial in a , exponential in c , and doubly-exponential in s , as required. Moreover, clearly $\mu \models \text{Pres}(\mathcal{P} \cup \mathcal{D}')$ and $\mu \not\models \text{Pres}(\alpha)$. Now let J' be the pseudo-model corresponding to μ ; by Lemma C.2, we have $J' \models \mathcal{P} \cup \mathcal{D}'$ and $J' \not\models \alpha$. By construction, the magnitude of each integer in J' is bounded by b . Furthermore, let J be the restriction of J' to the facts that unify with the head of at least one rule in $\mathcal{P} \cup \mathcal{D}'$; clearly, we still have $J \models \mathcal{P} \cup \mathcal{D}'$ and $J \not\models \alpha$. Finally, $|J| \leq |\mathcal{P} \cup \mathcal{D}'|$ holds by our construction, which implies our claim. \square

Lemma C.3. *For each semi-ground, limit-linear program \mathcal{P} , pseudo-interpretation J , and dataset \mathcal{D} , there exists a polynomial p such that $\mathbf{T}_{\mathcal{P} \cup \mathcal{D}}(J)$ can be computed in nondeterministic polynomial time in $\|\mathcal{P}\| + \|\mathcal{D}\| + \|J\|$, and in deterministic polynomial time in $\|\mathcal{P}\| + \|\mathcal{D}\| + \|J\|^{p(\max_{r \in \mathcal{P}} \|r\|)}$.*

Proof. Let $S = \{\text{hd}(r, J) \mid \text{rule } r \in \mathcal{P} \cup \mathcal{D} \text{ is applicable to } J\}$. Program \mathcal{P} is semi-ground, and therefore $\mathcal{P} \cup \mathcal{D}$ is semi-ground as well; thus, each rule of $\mathcal{P} \cup \mathcal{D}$ can contribute at most one fact to S , so we have $|S| \leq |\mathcal{P}| + |\mathcal{D}|$. By Definition 12, $\mathbf{T}_{\mathcal{P} \cup \mathcal{D}}(J)$ is the smallest (w.r.t. \sqsubseteq) pseudo-interpretation such that $\mathbf{T}_{\mathcal{P} \cup \mathcal{D}}(J) \models S$, so we can compute $\mathbf{T}_{\mathcal{P} \cup \mathcal{D}}(J)$ as the set containing each object and ordinary numeric fact in S , each fact $A(\mathbf{a}, \infty) \in S$ for A a limit predicate, each fact $A(\mathbf{a}, \ell) \in S$ such that A is a min (resp. max) predicate and $A(\mathbf{a}, k) \in S$ implies $k \neq \infty$ and $k \geq \ell$ (resp. $k \leq \ell$). To complete the proof of this lemma, we next argue that set S can be computed within the required time bounds.

Consider an arbitrary rule $r \in \mathcal{P} \cup \mathcal{D}$, and let J' be the subset of J containing all facts that unify with a body atom in r ; note that $|J'| \leq \|r\|$. Rule r is applicable to J if and only if conjunction $\mathcal{C}(r, J')$ has an integer solution. By construction, $\|\mathcal{C}(r, J')\|$ is linear in $\|r\| + \|J'\|$, the number of variables in $\mathcal{C}(r, J')$ and r is the same, $|\mathcal{C}(r, J')|$ is linear in $\|r\|$, and the magnitude of each integer in $\mathcal{C}(r, J')$ is exponentially bounded in $\|r\| + \|J'\|$. But then, checking whether $\mathcal{C}(r, J)$ has an integer solution is in NP w.r.t. $\|r\| + \|J\|$, and in PTIME w.r.t. $\|J\|^{p(\|r\|)}$ for some polynomial p , as we argue next.

- We first consider the former claim. Let a be the maximal magnitude of an integer in $\mathcal{C}(r, J')$. Conjunction $\mathcal{C}(r, J')$ contains only the numbers from r and J' , whose magnitude is at most $2^{\|r\|}$ and $2^{\|J'\|}$, respectively; thus, we have $a \leq 2^{\|r\| + \|J'\|}$. Moreover, the results by Papadimitriou [1981] show that there exists a polynomial p_1 such that the magnitude of an integer in a solution to $\mathcal{C}(r, J')$ can be bounded by $b = a^{p_1(\|r\|)}$, and so there exists a polynomial p_2 such that $b \leq 2^{p_2(\|r\| + \|J'\|)}$. The binary representation of b thus requires at most $\|r\| + \|J\|$ bits, and so we can guess it in polynomial time.
- We next consider the latter claim. By Theorem 5.4 of Kannan [1987], checking satisfiability of $\mathcal{C}(r, J)$ over \mathbb{Z} is fixed-parameter tractable in the number n of variables in r —that is, there exists a polynomial p_3 such that a solution to $\mathcal{C}(r, J)$ can be computed in time $O((\|r\| + \|J\|) \cdot 2^{p_3(n)})$. Since $n \leq \|r\|$ clearly holds, there exists a polynomial p_4 such that the satisfiability of $\mathcal{C}(r, J)$ can be checked in time that is thus $\|J\|^{p_4(\|r\|)}$.

Now assume that r is applicable to J . Then $\text{hd}(r, J) = \text{h}(r)$ if $\text{h}(r)$ is an object atom, so we assume that $\text{h}(r) = A(\mathbf{a}, s)$ is a limit atom and argue that $\text{opt}(r, J)$ can be computed within the required time bounds using the following two steps.

1. Depending on whether A is a min or a max predicate, we check whether there is a smallest/largest value for s in all solutions to $\mathcal{C}(r, J)$ —that is, we check whether the integer linear program ‘minimise/maximise s subject to $\mathcal{C}(r, J)$ ’ is bounded. Byrd *et al.* [1987] showed that this amounts to checking boundedness of the corresponding linear relaxation, which in turn can be reduced to checking linear feasibility and can be solved in deterministic polynomial time in $\|r\| + \|J\|$.
2. If the above problem is bounded, we compute its optimal solution, which can be reduced to polynomially many (in $\|r\| + \|J\|$) feasibility checks, as shown by Papadimitriou [1981] (Corollary 2 with binary search). Each such feasibility check is in NP w.r.t. $\|r\| + \|J\|$, and in PTIME w.r.t. $\|J\|^{p(\|r\|)}$.

Thus, $\text{hd}(r, J)$ can be computed in nondeterministic polynomial time in $\|r\| + \|J\|$, and in deterministic polynomial time in $\|J\|^{p(\|r\|)}$, which implies our claim. \square

Lemma C.4. *Deciding $\mathcal{P} \models \alpha$ is CONP-hard in data complexity for \mathcal{P} a limit-linear program and α a fact.*

Proof. An instance \mathcal{T} of the *square tiling* problem is given by an integer N coded in unary, a set $T = \{t_0, \dots, t_{M-1}\}$ of M tiles, and two compatibility relations $H \subseteq T \times T$ and $V \subseteq T \times T$. The problem is to determine whether there exists a tiling $\tau : \{0, \dots, N-1\}^2 \rightarrow T$ of an $N \times N$ square such that $\langle \tau(i, j), \tau(i+1, j) \rangle \in H$ holds for all $0 \leq i < N-1$ and $0 \leq j < N$, and $\langle \tau(i, j), \tau(i, j+1) \rangle \in V$ holds for all $0 \leq i < N$ and $0 \leq j < N-1$, which is known to be NP-complete. Thus, to prove the claim of this lemma, we reduce the complement of the problem by presenting a fixed program $\mathcal{P}_{\text{tiling}}$ and a dataset $\mathcal{D}_{\mathcal{T}}$ (that depends on \mathcal{T}), and showing that \mathcal{T} has no solution if and only if $\mathcal{P}_{\text{tiling}} \cup \mathcal{D}_{\mathcal{T}} \models \text{noSolution}$.

Our encoding uses object EDB predicates *succ*, *incompatibleH*, and *incompatibleV*; ordinary numeric EDB predicates *shift*, *tileNo*, *numTiles*, and *maxTiling*; nullary object IDB predicate *noSolution*; unary min IDB predicate *I*; and unary max IDB predicate *tiling*. Program $\mathcal{P}_{\text{tiling}}$ contains rules (31)–(35), where $(s \doteq t)$ abbreviates $(s \leq t) \wedge (t \leq s)$.

$$\rightarrow I(0) \quad (31)$$

$$\rightarrow \text{tiling}(0) \quad (32)$$

$$\begin{aligned} & \text{tiling}(n) \wedge \text{numTiles}(nt) \wedge \\ & \text{shift}(x, y, s) \wedge \text{tileNo}(u, t) \wedge I(m_1) \wedge I(m_2) \wedge (n \doteq m_1 \times nt \times s + t \times s + m_2) \wedge (m_2 < s) \wedge \\ & \text{succ}(x, x') \wedge \\ & \text{shift}(x', y, s') \wedge \text{tileNo}(u', t') \wedge I(m'_1) \wedge I(m'_2) \wedge (n \doteq m'_1 \times nt \times s' + t' \times s' + m'_2) \wedge (m'_2 < s') \wedge \\ & \text{incompatibleH}(u, u') \rightarrow \text{tiling}(n+1) \end{aligned} \quad (33)$$

$$\begin{aligned} & \text{tiling}(n) \wedge \text{numTiles}(nt) \wedge \\ & \text{shift}(x, y, s) \wedge \text{tileNo}(u, t) \wedge I(m_1) \wedge I(m_2) \wedge (n \doteq m_1 \times nt \times s + t \times s + m_2) \wedge (m_2 < s) \wedge \\ & \text{succ}(y, y') \wedge \\ & \text{shift}(x, y', s') \wedge \text{tileNo}(u', t') \wedge I(m'_1) \wedge I(m'_2) \wedge (n \doteq m'_1 \times nt \times s' + t' \times s' + m'_2) \wedge (m'_2 < s') \wedge \\ & \text{incompatibleV}(u, u') \rightarrow \text{tiling}(n+1) \end{aligned} \quad (34)$$

$$tiling(n) \wedge maxTiling(m) \wedge (m < n) \rightarrow noSolution \quad (35)$$

Dataset $\mathcal{D}_{\mathcal{T}}$ contains facts (36)–(42), where g_0, \dots, g_{N-1} are fresh objects, and t_i for $0 \leq i < M$ are distinct objects corresponding to the tiles in T . Since N is coded in unary, although numbers $M^{N^2} - 1$ and M^{i+Nj} are exponential in N , they can be computed in polynomial time and represented using polynomially many bits.

$$\rightarrow numTiles(M) \quad (36)$$

$$\rightarrow tileNo(t_i, i) \quad \text{for each } 0 \leq i < M \quad (37)$$

$$\rightarrow incompatibleH(t_i, t_j) \quad \text{for each } 0 \leq i, j < M \text{ such that } (t_i, t_j) \notin H \quad (38)$$

$$\rightarrow incompatibleV(t_i, t_j) \quad \text{for each } 0 \leq i, j < M \text{ such that } (t_i, t_j) \notin V \quad (39)$$

$$\rightarrow maxTiling(M^{N^2} - 1) \quad (40)$$

$$\rightarrow succ(g_i, g_{i+1}) \quad \text{for each } 0 \leq i < N - 1 \quad (41)$$

$$\rightarrow shift(g_i, g_j, M^{i+Nj}) \quad \text{for each } 0 \leq i, j < N \quad (42)$$

Our reduction uses the following idea. Facts (37) associate each tile t_i with an integer i where $0 \leq i < M$; hence, in the rest of this discussion, we do not distinguish a tile from its number. This allows us to represent each tiling τ using a number $\sum_{0 \leq i, j < N} \tau(i, j) \times M^{i+Nj}$. Thus, given a number n that encodes a tiling, number t with $0 \leq t < M$ corresponds to the tile assigned to position (i, j) if $n = m_1 \times M \times M^{i+Nj} + t \times M^{i+Nj} + m_2$ for some integers m_1 and m_2 where $0 \leq m_2 < M^{i+Nj}$. Thus, if numeric variable n is assigned such an encoding of a tiling and numeric variable s is assigned the factor M^{i+Nj} corresponding to a position (i, j) , then conjunction

$$tileNo(u, t) \wedge I(m_1) \wedge I(m_2) \wedge (n \doteq m_1 \times nt \times s + t \times s + m_2) \wedge (m_2 < s)$$

is true if and only if u is assigned the tile object corresponding to position (i, j) in the tiling encoded by n . To complete the construction, we represent each position (i, j) by a pair of objects (g_i, g_j) , each of which is associated with the corresponding factor M^{i+Nj} using facts (42). Facts (41) provide an ordering on g_i , which allows us to identify adjacent positions. Finally, fact (40) records the maximal number that encodes a tiling as we outlined earlier. Program \mathcal{P}_{tiling} then simply checks through all tilings: rule (32) ensures that the tiling encoded as 0 is checked; moreover, for each n such that $tiling(n)$ holds, rules (33) and (34) derive $tiling(n + 1)$ if either the horizontal or the vertical compatibility requirement is violated for the tiling encoded by n . Finally, rule (35) detects that no solution exists if $tiling(M^{N^2})$ is derived. \square

Lemma C.5. *Deciding $\mathcal{P} \models \alpha$ is CONEXPTIME-hard for \mathcal{P} a limit-linear program and α a fact.*

Proof. We present a reduction from the *succinct square tiling* problem. An instance \mathcal{T} of the problem is given by an integer N coded in unary, a set T containing M tiles, and horizontal and vertical compatibility relations H and V , respectively, as in the proof of Lemma C.4; however, the objective is to tile a square of $2^N \times 2^N$ positions, which is known to be NEXPTIME-complete. Thus, to prove the claim of this lemma, we reduce the complement of the problem by presenting a program $\mathcal{P}_{\mathcal{T}}$ and showing that \mathcal{T} has no solution if and only if $\mathcal{P}_{\mathcal{T}} \models noSolution$.

The main idea behind our reduction is similar to Lemma C.4. Program $\mathcal{P}_{\mathcal{T}}$ contains rules (43)–(45) that associate each tile with a number using an ordinary numeric predicate *tileNo*, and encode the horizontal and vertical incompatibility relations using the object predicates *incompatibleH* and *incompatibleV*.

$$\rightarrow tileNo(t_i, i) \quad \text{for each } 0 \leq i < M \quad (43)$$

$$\rightarrow incompatibleH(t_i, t_j) \quad \text{for each } 0 \leq i, j < M \text{ such that } (t_i, t_j) \notin H \quad (44)$$

$$\rightarrow incompatibleV(t_i, t_j) \quad \text{for each } 0 \leq i, j < M \text{ such that } (t_i, t_j) \notin V \quad (45)$$

The main difference to Lemma C.4 is that, in order to obtain a polynomial encoding, we cannot represent a position (i, j) in the grid explicitly using a pair of objects. Instead, we encode each position using a pair (\mathbf{i}, \mathbf{j}) where \mathbf{i} and \mathbf{j} are N -tuples of objects $\bar{0}$ and $\bar{1}$. If we read $\bar{0}$ and $\bar{1}$ as representing numbers 0 and 1, respectively, then each \mathbf{i} and \mathbf{j} can be seen as a binary number in $[0, 2^N - 1]$. By a slight abuse of notation, we often identify a tuple over $\bar{0}$ and $\bar{1}$ with the number it encodes and use tuples in arithmetic expressions. While positions can be encoded using N bits, we will also need to ensure distance between positions, which requires $N + 1$ bits. In the rest of this proof, $\bar{0}$ and $\bar{1}$ stand for tuples $\bar{0}, \dots, \bar{0}$ and $\bar{1}, \dots, \bar{1}$, respectively, whose length is often implicit from the context where these tuples occur. Similarly, \mathbf{x} , \mathbf{x}' , \mathbf{y} , and \mathbf{y}' are tuples of distinct variables whose length will also be clear from the context.

To axiomatise an ordering on numbers with N bits, program $\mathcal{P}_{\mathcal{T}}$ contains rules (46)–(49), where B is a unary object predicate, *succ* is a $2N$ -ary object predicate, and *succ'* is a $(2N + 2)$ -ary object predicate. Rules (46)–(48) ensure $\mathcal{P}_{\mathcal{T}} \models succ(\mathbf{i}, \mathbf{j})$ where \mathbf{i} and \mathbf{j} encode numbers with N bits such that $\mathbf{j} = 1 + \mathbf{i}$; in particular, rule (48) encodes binary incrementation, where $\mathbf{x}\bar{1}\bar{0} = 1 + \mathbf{x}\bar{1}\bar{0}$ holds for each position k and each k -tuple of zeros and ones \mathbf{x} . Rules (46)–(47) and (49) ensure an analogous property for *succ'*, but for numbers with $N + 1$ bits.

$$\rightarrow B(\bar{0}) \quad (46)$$

$$\rightarrow B(\bar{1}) \quad (47)$$

$$\bigwedge_{i=1}^k B(x_i) \rightarrow \text{succ}(\mathbf{x}, \bar{0}, \bar{1}, \mathbf{x}, \bar{1}, \bar{0}) \quad \text{for each } 0 \leq k < N \text{ where } |\mathbf{x}| = k \text{ and } |\bar{1}| = |\bar{0}| = N - k - 1 \quad (48)$$

$$\bigwedge_{i=1}^k B(x_i) \rightarrow \text{succ}'(\mathbf{x}, \bar{0}, \bar{1}, \mathbf{x}, \bar{1}, \bar{0}) \quad \text{for each } 0 \leq k < N + 1 \text{ where } |\mathbf{x}| = k \text{ and } |\bar{1}| = |\bar{0}| = N - k \quad (49)$$

Analogously to the proof of Lemma C.4, we encoded tilings using numbers in $[0, M^{2^{2N}} - 1]$. To compute the maximum number encoding a tiling, program $\mathcal{P}_{\mathcal{T}}$ contains rules (50)–(53), where maxTiling is a unary min predicate, and auxT is a $(2N + 1)$ -ary min predicate. Auxiliary rules (50)–(52) multiply M with itself as many times as there are grid positions, so we have $\mathcal{P}_{\mathcal{T}} \models \text{auxT}(\mathbf{i}, \mathbf{j}, M^{1+\mathbf{i}+2^N \cdot \mathbf{j}})$ for each position (\mathbf{i}, \mathbf{j}) . Consequently, rule (53) ensures that, for all s , we have $\mathcal{P}_{\mathcal{T}} \models \text{maxTiling}(s)$ if and only if $s \geq M^{2^{2N}} - 1$.

$$\rightarrow \text{auxT}(\bar{0}, \bar{0}, M) \quad (50)$$

$$\text{auxT}(\mathbf{x}, \mathbf{y}, n) \wedge \text{succ}(\mathbf{x}, \mathbf{x}') \rightarrow \text{auxT}(\mathbf{x}', \mathbf{y}, M \times n) \quad (51)$$

$$\text{auxT}(\bar{1}, \mathbf{y}, n) \wedge \text{succ}(\mathbf{y}, \mathbf{y}') \rightarrow \text{auxT}(\bar{0}, \mathbf{y}', M \times n) \quad (52)$$

$$\text{auxT}(\bar{1}, \bar{1}, n) \rightarrow \text{maxTiling}(n - 1) \quad (53)$$

Unlike in the proof of Lemma C.4, we cannot include shift factors explicitly into $\mathcal{P}_{\mathcal{T}}$ since this would make the encoding exponential; moreover, we could precompute shift factors using rules similar to (50)–(52), but then we would need to use values from limit predicates in multiplication, which would not produce a limit-linear program. Therefore, we check tilings using a different approach. As in the proof of Lemma C.4, our construction ensures that, for all s , we have $\mathcal{P}_{\mathcal{T}} \models \text{tiling}(s)$ if and only if each tiling n with $0 \leq n \leq s$ does not satisfy the compatibility relations. Given a tiling encoded by n and a position (\mathbf{i}, \mathbf{j}) , let

$$s_{n, \mathbf{i}, \mathbf{j}} = \left\lfloor \frac{n}{M^{\mathbf{i} + \mathbf{j} \cdot 2^N}} \right\rfloor.$$

Program $\mathcal{P}_{\mathcal{T}}$ contains rules (54)–(57) where shiftedTiling is a max predicate of arity $2N + 1$ and I is a unary min predicate. These rules ensure that, for each \mathbf{i}, \mathbf{j} , and tiling n such that $\mathcal{P}_{\mathcal{T}} \models \text{tiling}(n)$, we have $\mathcal{P}_{\mathcal{T}} \models \text{shiftedTiling}(\mathbf{i}, \mathbf{j}, s_{n, \mathbf{i}, \mathbf{j}})$. To understand how this is achieved, we order the grid positions as follows:

$$(0, 0), (1, 0), \dots, (2^N - 1, 0), (0, 1), (1, 1), \dots, (2^N - 1, 1), \dots, (0, 2^N - 1), (1, 2^N - 1), \dots, (2^N - 1, 2^N - 1)$$

Now consider an arbitrary position (\mathbf{i}, \mathbf{j}) and its successor $(\mathbf{i}', \mathbf{j}')$ in the ordering. The encoding of a tiling using an integer n ensures $s_{n, \mathbf{i}, \mathbf{j}} = M \times s_{n, \mathbf{i}', \mathbf{j}'} + t$ holds, where $0 \leq t < M$ is the number of the tile that n assigns to position (\mathbf{i}, \mathbf{j}) . Thus, rule (55) ensures that position $(\bar{0}, \bar{0})$ satisfies the mentioned property, rule (56) handles adjacent positions of the form (\mathbf{i}, \mathbf{j}) and $(\mathbf{i} + 1, \mathbf{j})$, and rule (57) handles adjacent positions of the form $(\bar{1}, \mathbf{j})$ and $(\bar{0}, \mathbf{j} + 1)$.

$$\rightarrow I(0) \quad (54)$$

$$\text{tiling}(n) \rightarrow \text{shiftedTiling}(\bar{0}, \bar{0}, n) \quad (55)$$

$$\text{shiftedTiling}(\mathbf{x}, \mathbf{y}, n) \wedge \text{succ}(\mathbf{x}, \mathbf{x}') \wedge I(\ell) \wedge I(m) \wedge (\ell < M) \wedge (n \doteq M \times m + \ell) \rightarrow \text{shiftedTiling}(\mathbf{x}', \mathbf{y}, m) \quad (56)$$

$$\text{shiftedTiling}(\bar{1}, \mathbf{y}, n) \wedge \text{succ}(\mathbf{y}, \mathbf{y}') \wedge I(\ell) \wedge I(m) \wedge (\ell < M) \wedge (n \doteq M \times m + \ell) \rightarrow \text{shiftedTiling}(\bar{0}, \mathbf{y}', m) \quad (57)$$

Note that, for all n and n' with $n < n'$ and each position (\mathbf{i}, \mathbf{j}) , we have $s_{n, \mathbf{i}, \mathbf{j}} < s_{n', \mathbf{i}, \mathbf{j}}$. Thus, since shiftedTiling is a max predicate, the limit value for s in $\text{shiftedTiling}(\mathbf{i}, \mathbf{j}, s)$ will always correspond to the limit value for n in $\text{tiling}(n)$.

Checking horizontal compatibility is now easy, but checking vertical compatibility requires dividing $s_{n, \mathbf{i}, \mathbf{j}}$ by M^{2^N} , which would make the reduction exponential. Hence, $\mathcal{P}_{\mathcal{T}}$ checks compatibility using rules (58)–(61), where conflict is a max predicate of arity $3N + 3$. These rules ensure that, for each $\mathbf{i}, \mathbf{j}, \mathbf{d}, u$, and tiling n such that $\mathcal{P}_{\mathcal{T}} \models \text{tiling}(n)$ and the position that precedes (\mathbf{i}, \mathbf{j}) by distance \mathbf{d} in the ordering cannot be labelled in n with tile u , we have $\mathcal{P}_{\mathcal{T}} \models \text{conflict}(\mathbf{i}, \mathbf{j}, \mathbf{d}, u, s_{n, \mathbf{i}, \mathbf{j}})$. To this end, assume that (\mathbf{x}, \mathbf{y}) is labelled with tile u' ; now if $(u, u') \notin H$ and $\mathbf{x} \neq \bar{0}$ (i.e., the predecessor \mathbf{x}' of \mathbf{x} exists), then rule (58) says that the position preceding (\mathbf{x}, \mathbf{y}) by $\bar{0}\bar{1}$ (i.e., the position to the left) cannot be labelled with u ; moreover, if $(u, u') \notin V$ and $\mathbf{y} \neq \bar{0}$ (i.e., the predecessor \mathbf{y}' of \mathbf{y} exists), then rule (59) says that the position preceding (\mathbf{x}, \mathbf{y}) by $\bar{1}\bar{0} = 2^N$ (i.e., the position above) cannot be labelled with u . Moreover, rule (60) propagates such constraints from position (\mathbf{i}, \mathbf{j}) to $(\mathbf{i} - 1, \mathbf{j})$ while reducing the distance by one, and rule (61) does so for positions $(\bar{0}, \mathbf{j})$ and $(\bar{1}, \mathbf{j} - 1)$.

$$\text{shiftedTiling}(\mathbf{x}, \mathbf{y}, m) \wedge \text{succ}(\mathbf{x}', \mathbf{x}) \wedge \text{incompatible}H(u, u') \wedge \text{tileNo}(u', t') \wedge I(\ell) \wedge (m \doteq M \times \ell + t') \rightarrow \text{conflict}(\mathbf{x}, \mathbf{y}, \bar{0}\bar{1}, u, m) \quad (58)$$

$$\text{shiftedTiling}(\mathbf{x}, \mathbf{y}, m) \wedge \text{succ}(\mathbf{y}', \mathbf{y}) \wedge \text{incompatible}V(u, u') \wedge \text{tileNo}(u', t') \wedge I(\ell) \wedge (m \doteq M \times \ell + t') \rightarrow \text{conflict}(\mathbf{x}, \mathbf{y}, \bar{1}\bar{0}, u, m) \quad (59)$$

$$\text{shiftedTiling}(\mathbf{x}, \mathbf{y}, m) \wedge \text{succ}(\mathbf{x}, \mathbf{x}') \wedge \text{conflict}(\mathbf{x}', \mathbf{y}, \mathbf{z}', u, m') \wedge \text{succ}'(\mathbf{z}, \mathbf{z}') \wedge I(\ell) \wedge (\ell < M) \wedge (m \doteq M \times m' + \ell) \rightarrow \text{conflict}(\mathbf{x}, \mathbf{y}, \mathbf{z}, u, m) \quad (60)$$

$$\text{shiftedTiling}(\bar{\mathbf{l}}, \mathbf{y}, m) \wedge \text{succ}(\mathbf{y}, \mathbf{y}') \wedge \text{conflict}(\bar{\mathbf{0}}, \mathbf{y}', \mathbf{z}', u, m') \wedge \text{succ}'(\mathbf{z}, \mathbf{z}') \wedge I(\ell) \wedge (\ell < M) \wedge (m \doteq M \times m' + \ell) \rightarrow \text{conflict}(\bar{\mathbf{l}}, \mathbf{y}, \mathbf{z}, u, m) \quad (61)$$

Program $\mathcal{P}_{\mathcal{T}}$ also contains rules (62)–(64) where *invalid* is a $(2N + 1)$ -ary max predicate. These rules ensure that, for each \mathbf{i}, \mathbf{j} , and each tiling n such that $\mathcal{P}_{\mathcal{T}} \models \text{tiling}(n)$ and there exists a position $(\mathbf{i}', \mathbf{j}')$ that comes after (\mathbf{i}, \mathbf{j}) in the position order such that n does not satisfy the compatibility relations between $(\mathbf{i}', \mathbf{j}')$ and its horizontal or vertical successor, we have $\mathcal{P}_{\mathcal{T}} \models \text{invalid}(\mathbf{i}, \mathbf{j}, s_{n, \mathbf{i}, \mathbf{j}})$. Rule (62) determines invalidity at position (\mathbf{x}, \mathbf{y}) for conflicts with zero distance, and rules (63) and (64) propagate this information to preceding positions analogously to rules (60) and (61).

$$\text{conflict}(\mathbf{x}, \mathbf{y}, \bar{\mathbf{0}}, u, m) \wedge \text{tileNo}(u, t) \wedge I(\ell) \wedge (m \doteq M \times \ell + t) \rightarrow \text{invalid}(\mathbf{x}, \mathbf{y}, m) \quad (62)$$

$$\text{shiftedTiling}(\mathbf{x}, \mathbf{y}, m) \wedge \text{succ}(\mathbf{x}, \mathbf{x}') \wedge \text{invalid}(\mathbf{x}', \mathbf{y}, m') \wedge I(\ell) \wedge (\ell < M) \wedge (m \doteq M \times m' + \ell) \rightarrow \text{invalid}(\mathbf{x}, \mathbf{y}, m) \quad (63)$$

$$\text{shiftedTiling}(\bar{\mathbf{l}}, \mathbf{y}, m) \wedge \text{succ}(\mathbf{y}, \mathbf{y}') \wedge \text{invalid}(\bar{\mathbf{0}}, \mathbf{y}', m') \wedge I(\ell) \wedge (\ell < M) \wedge (m \doteq M \times m' + \ell) \rightarrow \text{invalid}(\bar{\mathbf{l}}, \mathbf{y}, m) \quad (64)$$

Finally, program $\mathcal{P}_{\mathcal{T}}$ contains rules (65)–(67) where *tiling* is a unary max predicate and *noSolution* is a nullary predicate. Rule (65) ensures that tiling encoded by 0 is checked. Based on our discussion from the previous paragraph, for each invalid tiling n such that $\mathcal{P}_{\mathcal{T}} \models \text{tiling}(n)$, we have $\mathcal{P}_{\mathcal{T}} \models \text{invalid}(\bar{\mathbf{0}}, \bar{\mathbf{0}}, s_{\bar{\mathbf{0}}, \bar{\mathbf{0}}, n})$; moreover, $s_{\bar{\mathbf{0}}, \bar{\mathbf{0}}, n} = n$ so, if $\mathcal{P}_{\mathcal{T}} \models \text{invalid}(\bar{\mathbf{0}}, \bar{\mathbf{0}}, n)$ holds, then rule (66) ensures that tiling encoded by $n + 1$ is considered; atom $(m \doteq n)$ is needed in the rule since no numeric variable is allowed to occur in more than one standard body atom. If we exhaust all available tilings, rule (67) determines that no solution exists, just as in the proof of Lemma C.4.

$$\rightarrow \text{tiling}(0) \quad (65)$$

$$\text{invalid}(\bar{\mathbf{0}}, \bar{\mathbf{0}}, m) \wedge \text{tiling}(n) \wedge (m \doteq n) \rightarrow \text{tiling}(n + 1) \quad (66)$$

$$\text{tiling}(n) \wedge \text{maxTiling}(m) \wedge (m < n) \rightarrow \text{noSolution} \quad (67)$$

Based on our discussion of the consequences of $\mathcal{P}_{\mathcal{T}}$, we conclude that instance \mathcal{T} of the succinct tiling problem does not have a solution if and only if $\mathcal{P}_{\mathcal{T}} \models \text{noSolution}$. \square

Proposition C.6. *For J a pseudo-interpretation and α a fact, $J \models \alpha$ if and only if $\{\alpha\} \sqsubseteq J$.*

Proof. Consider an arbitrary pseudo-interpretation J and the corresponding limit-closed interpretation I . If $J \models \alpha$, then $\alpha \in I$, so there exists a fact $\alpha' \in J$ such that $\{\alpha\} \sqsubseteq \{\alpha'\}$, which implies $\{\alpha\} \sqsubseteq J$. Moreover, if $\{\alpha\} \sqsubseteq J$, then there exists a fact $\alpha' \in J$ such that $\{\alpha\} \sqsubseteq \{\alpha'\}$ and, since I is limit-closed, we have $\alpha \in I$, which implies $J \models \alpha$. \square

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

Proof. Lemmas C.4 and C.5 prove hardness. Moreover, the following nondeterministic algorithm decides $\mathcal{P} \cup \mathcal{D} \models \alpha$ in time polynomial in $\|\mathcal{D}\| + \|\alpha\|$, and exponential in $\|\mathcal{P}\| + \|\mathcal{D}\| + \|\alpha\|$.

1. Compute the semi-grounding \mathcal{P}' of \mathcal{P} .
2. Guess a pseudo-interpretation J over the signature of $\mathcal{P}' \cup \mathcal{D}$ such that the number of facts in J and the absolute values of all integers in J are bounded as in Theorem 20.
3. Check that J is a pseudo-model of $\mathcal{P} \cup \mathcal{D}$; if not, return false.
4. Return false if $J \models \alpha$ and true otherwise.

Correctness of the algorithm follows from Theorem 20, so we next argue about its complexity. The mentioned data complexity holds by the following observations.

- In step 1, $\|\mathcal{P}'\|$, $|\mathcal{P}'|$, and the time required to compute \mathcal{P}' are all polynomial in $\|\mathcal{D}\|$ and constant in $\|\alpha\|$.
- Since $|\mathcal{P}'|$ is polynomial in $\|\mathcal{D}\|$ and constant in $\|\alpha\|$, and $\max_{r \in \mathcal{P}'} \|r\|$ is constant in $\|\mathcal{D}\|$ and $\|\alpha\|$, the magnitude of the integers in J is exponentially bounded in $\|\alpha\| + \|\mathcal{D}\|$ by Theorem 20; thus, the number of bits needed to represent each integer in J is polynomial in $\|\alpha\| + \|\mathcal{D}\|$. Furthermore, we have $|J| \leq |\mathcal{D}| + |\mathcal{P}'|$, and $|\mathcal{D}| + |\mathcal{P}'|$ is polynomial in $\|\mathcal{D}\|$ and constant in $\|\alpha\|$; thus, J can be guessed in step 2 in nondeterministic polynomial time in $\|\alpha\| + \|\mathcal{D}\|$.
- By Lemma 14, checking that J is a pseudo-model of $\mathcal{P} \cup \mathcal{D}$ amounts to checking $\mathbf{T}_{\mathcal{P}' \cup \mathcal{D}}(J) = J$. By Lemma C.3, $\mathbf{T}_{\mathcal{P}' \cup \mathcal{D}}(J)$ can be computed in deterministic polynomial time in $\|J\| + \|\mathcal{D}\|$ and hence in $\|\mathcal{D}\|$ (as $\|J\|$ is polynomial in $\|\mathcal{D}\|$). Hence, step 3 requires deterministic polynomial time in $\|\alpha\| + \|\mathcal{D}\|$.
- By Proposition C.6, step 4 amounts to checking $\{\alpha\} \sqsubseteq J$, which can be done in time polynomial in $\|J\|$ and $\|\alpha\|$, and hence polynomial in $\|\mathcal{D}\| + \|\alpha\|$ as well.

Finally, the mentioned combined complexity holds by the following observations.

- In step 1, $\|\mathcal{P}'\|$, $|\mathcal{P}'|$, and time required to compute \mathcal{P}' are all exponential in $\|\mathcal{P}\| + \|\mathcal{D}\|$ and constant in $\|\alpha\|$.
- Since $|\mathcal{P}'|$ is exponential in $\|\mathcal{P}\| + \|\mathcal{D}\|$ and constant in $\|\alpha\|$, and $\max_{r \in \mathcal{P}'} \|r\|$ is linear in $\|\mathcal{P}\|$ and constant in $\|\mathcal{D}\| + \|\alpha\|$, the magnitude of the integers in J is doubly exponentially bounded in $\|\mathcal{P}\| + \|\mathcal{D}\| + \|\alpha\|$ by Theorem 20; thus, the number of bits needed to represent each integer in J is exponential in $\|\mathcal{P}\| + \|\mathcal{D}\| + \|\alpha\|$. Furthermore, we have $|J| < |\mathcal{D}| + |\mathcal{P}'|$, and $|\mathcal{D}| + |\mathcal{P}'|$ is exponential in $\|\mathcal{P}\| + \|\mathcal{D}\|$ and constant in $\|\alpha\|$; thus, J can be guessed in step 2 in nondeterministic exponential time in $\|\mathcal{P}\| + \|\mathcal{D}\| + \|\alpha\|$.
- By Lemma 14, checking that J is a pseudo-model of $\mathcal{P} \cup \mathcal{D}$ amounts to checking $\mathbf{T}_{\mathcal{P} \cup \mathcal{D}}(J) = J$. By Lemma C.3, polynomial p exists such that $\mathbf{T}_{\mathcal{P} \cup \mathcal{D}}(J)$ can be computed in deterministic polynomial time in $\|\mathcal{P}'\| + \|\mathcal{D}\| + \|J\|^{p(\max_{r \in \mathcal{P}'} \|r\|)}$, which, in turn, is bounded by $O(2^{\|\mathcal{P}\| + \|\mathcal{D}\|}) + \|\mathcal{D}\| + O(2^{(\|\mathcal{P}\| + \|\mathcal{D}\|)p(\max_{r \in \mathcal{P}} \|r\|)})$. Hence, step 3 requires deterministic exponential time in $\|\mathcal{P}\| + \|\mathcal{D}\| + \|\alpha\|$.
- By Proposition C.6, step 4 amounts to checking $\{\alpha\} \sqsubseteq J$, which can be done in time polynomial in $\|J\|$ and $\|\alpha\|$, and hence in time exponential in $\|\mathcal{P}\| + \|\mathcal{D}\| + \|\alpha\|$. \square

D Proofs for Section 6

For arbitrary value propagation graph $G_{\mathcal{P}}^J = (\mathcal{V}, \mathcal{E}, \mu)$, a *path* in $G_{\mathcal{P}}^J$ is a nonempty sequence $\pi = v_1, \dots, v_n$ of nodes from \mathcal{V} such that $\langle v_i, v_{i+1} \rangle \in \mathcal{E}$ holds for each $0 \leq i < n$; such π *starts* in v_1 and *ends* in v_n . We define $|\pi| = n$; moreover, by a slight abuse of notation, we sometimes write $\pi \cap X = \emptyset$ or $v_i \in \pi$, where we identify π with the set of its nodes. A path π is *simple* if all of its nodes are pair-wise distinct. A path π is a *cycle* if $v_n = v_1$.

Definition D.1. Given a semi-ground, limit linear program \mathcal{P} , a pseudo-interpretation J , value propagation graph $G_{\mathcal{P}}^J = (\mathcal{V}, \mathcal{E}, \mu)$, and a path $\pi = v_1, \dots, v_n$ in $G_{\mathcal{P}}^J$, the weight $\mu(\pi)$ of π is defined as

$$\mu(\pi) = \sum_{i=1}^{n-1} \mu(\langle v_i, v_{i+1} \rangle).$$

Lemma D.2. Let \mathcal{P} be a semi-ground and stable limit-linear program, let J be a pseudo-model of \mathcal{P} , let $G_{\mathcal{P}}^J = (\mathcal{V}, \mathcal{E}, \mu)$, and let $v_{Aa}, v_{Bb} \in \mathcal{V}$ be nodes such that v_{Ab} is reachable from v_{Ba} by a path π . Then, for each $k \in \mathbb{Z}$ such that $J \models B(b, k)$,

- $J \models A(a, k + \mu(\pi))$ if A and B are both max predicates;
- $J \models A(a, -k - \mu(\pi))$ if A is a min predicate and B is a max predicate;
- $J \models A(a, k - \mu(\pi))$ if A and B are both min predicates; and
- $J \models A(a, -k + \mu(\pi))$ if A is a max predicate and B is a min predicate.

Proof. We consider the case when A and B are both max predicates; the remaining cases are analogous. We proceed by induction on the length of π . The base case (π is empty) is immediate. For the inductive step, assume that $\pi = \pi', v_{Aa}$ where π' is a path starting at v_{Bb} and ending in node v_{Cc} . Then, there exists an edge $e = \langle v_{Cc}, v_{Aa} \rangle \in \mathcal{E}$, and e is produced by a rule $r = C(c, n) \wedge \varphi \rightarrow A(a, s) \in \mathcal{P}$ such that n is a variable occurring in s , and σ is a grounding of r such that

1. $J \models (C(c, n) \wedge \varphi)\sigma$, and
2. $\delta_r^e(J) = \mu(e) = \mu(\langle v_{Cc}, v_{Aa} \rangle)$.

We next consider the case when C is a max predicate; the case when C is a min predicate is analogous. Let ℓ be such that $C(c, \ell) \in J$. We have the following possibilities.

- If $\text{opt}(r, J)$ and ℓ are both integers (i.e., they are not ∞), we have $\mu(e) = \text{opt}(r, J) - \ell$.
- If $\text{opt}(r, J) = \infty$, then $\mu(e) = \infty$ by Definition 23.
- If $\ell = \infty$, then $\mu(e) = \infty$ by Definition 24 and the fact that \mathcal{P} is stable, and moreover $\text{opt}(r, J) = \infty$ by Definition 23.

Now for an arbitrary $k \in \mathbb{Z}$ such that $J \models B(b, k)$, we consider the following two cases.

- $\mu(e) \neq \infty$. The inductive hypothesis holds for π' , so $J \models C(c, k + \mu(\pi'))$ and thus $k + \mu(\pi') \leq \ell$. Consequently, we have $\mu(\pi) = \mu(\pi') + \mu(e) = \mu(\pi') + \text{opt}(r, J) - \ell \leq \text{opt}(r, J) - k$, and so $k + \mu(\pi) \leq \text{opt}(r, J)$ holds. Moreover, $J \models \mathcal{P}$ implies $\mathbf{T}_{\mathcal{P}}(J) = J$ by Lemma 14; thus, Proposition C.6 and the definition of $\mathbf{T}_{\mathcal{P}}$ imply $J \models A(a, k + \mu(\pi))$.
- $\mu(e) = \infty$. Clearly, $\mu(\pi) = \infty$. Moreover, $J \models \mathcal{P}$ implies $\mathbf{T}_{\mathcal{P}}(J) = J$ by Lemma 14; thus, $\text{opt}(r, J) = \infty$, Proposition C.6, and the definition of $\mathbf{T}_{\mathcal{P}}$ imply $J \models A(a, \infty)$. \square

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

Proof. Let $G_{\mathcal{P}}^J = (\mathcal{V}, \mathcal{E}, \mu)$, let $J' = \mathbf{T}_{\mathcal{P}}^\infty$, and let $G_{\mathcal{P}}^{J'} = (\mathcal{V}', \mathcal{E}', \mu')$. Now assume for the sake of a contradiction that there exist a cycle π in $G_{\mathcal{P}}^J$ and a node $v_{A\mathbf{a}} \in \pi$ such that $\mu(\pi) > 0$ and $J' \not\models A(\mathbf{a}, \infty)$. Rule applicability is monotonic w.r.t. \sqsubseteq , so π is still a cycle in $G_{\mathcal{P}}^{J'}$, and, since \mathcal{P} is stable, we have $\mu'(\pi) \geq \mu(\pi) > 0$. We consider the case when A is a max predicate; the remaining case is analogous. Now $v_{A\mathbf{a}} \in \mathcal{V} \subseteq \mathcal{V}'$ implies that $A(\mathbf{a}, k) \in J'$ for some k ; moreover, $J' \not\models A(\mathbf{a}, \infty)$ implies $k \neq \infty$. But then, Lemma D.2 implies $J' \models A(\mathbf{a}, k + \mu'(\pi))$; moreover, $\mu'(\pi) > 0$ implies that $k + \mu'(\pi)$ is either ∞ or it is an integer larger than k ; either way, this contradicts our assumption that $A(\mathbf{a}, k) \in J'$. \square

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

Proof. For J a pseudo-interpretation, A an $(n+1)$ -ary limit predicate, and \mathbf{a} an n -tuple of objects such that $A(\mathbf{a}, \ell) \in J$, let $\text{val}(J, A\mathbf{a}) = \ell$ if $\ell = \infty$ or A is a max predicate and $\ell \in \mathbb{Z}$, and $\text{val}(J, A\mathbf{a}) = -\ell$ if A is a min predicate and $\ell \in \mathbb{Z}$; moreover, let $\mathcal{R}(J, A\mathbf{a})$ be the set containing each rule $r \in \mathcal{P}$ that is applicable to J and where $\text{h}(r)$ is of the form $A(\mathbf{a}, s)$. By monotonicity of $\text{Datalog}_{\mathbb{Z}}$, we have $\mathcal{R}(J, A\mathbf{a}) \subseteq \mathcal{R}(J', A\mathbf{a})$ for each J and J' such that $J \sqsubseteq J'$. Moreover, for each edge $e = \langle v_{B\mathbf{b}}, v_{A\mathbf{a}} \rangle \in \mathcal{E}$ generated by a rule $r \in \mathcal{R}(J, A\mathbf{a})$, Definition 23 ensures that the following property holds:

$$\text{val}(J, B\mathbf{b}) + \mu(e) \geq \text{val}(\mathbf{T}_{\{r\}}(J), A\mathbf{a}) \quad (*)$$

To prove this lemma, we first show the following auxiliary claim.

Claim (\diamond). *For each $n \geq 0$ determining the pseudo-interpretation $J = \mathbf{T}_{\mathcal{P}}^n$ and the value propagation graph $G_{\mathcal{P}}^J = (\mathcal{V}, \mathcal{E}, \mu)$, each $n' \geq 1$ determining the pseudo-interpretation $J' = \mathbf{T}_{\mathcal{P}}^{n+n'}$ and the value propagation graph $G_{\mathcal{P}}^{J'} = (\mathcal{V}', \mathcal{E}', \mu')$, each set of nodes $X \subseteq \mathcal{V}$, and each node $v_{A\mathbf{a}} \in \mathcal{V}$ of such that*

1. $\mathcal{E} = \mathcal{E}'$;
2. $\text{val}(J', B\mathbf{b}) = \infty$ holds for each node $v_{B\mathbf{b}} \in \mathcal{V}'$ that occurs in $G_{\mathcal{P}}^{J'}$ in a positive-weight cycle;
3. $v_{A\mathbf{a}} \notin X$;
4. $\text{val}(J', A\mathbf{a}) < \text{val}(\mathbf{T}_{\mathcal{P}}(J'), A\mathbf{a})$;
5. $|\pi| \leq n'$ holds for each simple path π in $G_{\mathcal{P}}^J$ that ends in $v_{A\mathbf{a}}$ and satisfies $\pi \cap X = \emptyset$;
6. for each node $v_{B\mathbf{b}} \in X$, there exists a path π in $G_{\mathcal{P}}^J$ that starts in $v_{A\mathbf{a}}$ and ends in $v_{B\mathbf{b}}$;

one of the following holds:

- (i) $\text{val}(J', C\mathbf{c}) + \mu'(\pi) \geq \text{val}(\mathbf{T}_{\mathcal{P}}(J'), A\mathbf{a})$ for some node $v_{C\mathbf{c}} \in X$ and path π in $G_{\mathcal{P}}^J$ starting in $v_{C\mathbf{c}}$ and ending in $v_{A\mathbf{a}}$;
- (ii) $\mathcal{R}(J, C\mathbf{c}) \subsetneq \mathcal{R}(J', C\mathbf{c})$ for some node $v_{C\mathbf{c}} \in \mathcal{V}$.

Proof. For arbitrary n , we prove the claim by induction on n' . For the base case $n' = 1$, consider an arbitrary set $X \subseteq \mathcal{V}$ and vertex $v_{A\mathbf{a}} \in \mathcal{V}$ that satisfy properties 1–6 of (\diamond). We distinguish two cases.

- There exists an edge $e = \langle v_{B\mathbf{b}}, v_{A\mathbf{a}} \rangle \in \mathcal{E}$ such that $\text{val}(J', B\mathbf{b}) + \mu'(e) = \text{val}(\mathbf{T}_{\mathcal{P}}(J'), A\mathbf{a})$. Now either $v_{B\mathbf{b}} \in X$ or $v_{B\mathbf{b}} = v_{A\mathbf{a}}$ holds: if that were not the case, path $\pi = v_{B\mathbf{b}}, v_{A\mathbf{a}}$ would be a simple path in $G_{\mathcal{P}}^J$ such that $|\pi| = 2$, which would contradict property 5.

We next show that $v_{B\mathbf{b}} = v_{A\mathbf{a}}$ is impossible. For the sake of a contradiction, assume that $v_{B\mathbf{b}} = v_{A\mathbf{a}}$ holds, and thus we have $\text{val}(J', A\mathbf{a}) + \mu'(e) = \text{val}(\mathbf{T}_{\mathcal{P}}(J'), A\mathbf{a})$. By property 4, this implies $\text{val}(J', A\mathbf{a}) + \mu'(e) > \text{val}(J', A\mathbf{a})$, and hence $\mu'(e) > 0$. Consequently, path π is a positive-weight cycle in $G_{\mathcal{P}}^{J'}$, and so, by property 2, we have $\text{val}(J, A\mathbf{a}) = \infty$, which, in turn, contradicts property 4.

Consequently, we have $v_{B\mathbf{b}} \in X$. But then, since, by assumption, $\text{val}(J', B\mathbf{b}) + \mu'(e) = \text{val}(\mathbf{T}_{\mathcal{P}}(J'), A\mathbf{a})$, part (i) of the claim holds for $v_{C\mathbf{c}} = v_{B\mathbf{b}}$.

- For each edge $\langle v_{B\mathbf{b}}, v_{A\mathbf{a}} \rangle \in \mathcal{E}$ we have $\text{val}(J', B\mathbf{b}) + \mu'(e) < \text{val}(\mathbf{T}_{\mathcal{P}}(J'), A\mathbf{a})$. Then, for each rule $r \in \mathcal{P}$ that generates an edge $e = \langle v_{B\mathbf{b}}, v_{A\mathbf{a}} \rangle$, property (*) ensures $\text{val}(\mathbf{T}_{\{r\}}(J'), A\mathbf{a}) < \text{val}(\mathbf{T}_{\mathcal{P}}(J'), A\mathbf{a})$. Since

$$\text{val}(\mathbf{T}_{\mathcal{P}}(J'), A\mathbf{a}) = \max_{r \in \mathcal{P}, \text{h}(r) = A(\mathbf{a}, s)} \text{val}(\mathbf{T}_{\{r\}}(J'), A\mathbf{a}),$$

a rule $r \in \mathcal{P}$ exists that satisfies $\text{val}(\mathbf{T}_{\mathcal{P}}(J'), A\mathbf{a}) = \text{val}(\mathbf{T}_{\{r\}}(J'), A\mathbf{a})$ but does not generate an edge in \mathcal{E} ending in $v_{A\mathbf{a}}$. Clearly, $\text{h}(r)$ is of the form $A(\mathbf{a}, s)$ and r is applicable to J' , so $r \in \mathcal{R}(J', A\mathbf{a})$ holds. Moreover, r is semi-ground; hence, if s were to contain a variable, this variable would occur in a limit body atom of r , and so r would generate an edge in \mathcal{E} ; consequently, s is ground. Finally, if r were applicable to J , then $\{A(\mathbf{a}, s)\} \sqsubseteq J'$ and so $\text{val}(J', A\mathbf{a}) \geq \text{val}(\{A(\mathbf{a}, s)\}, A\mathbf{a}) = \text{val}(\mathbf{T}_{\mathcal{P}}(J'), A\mathbf{a})$, which contradicts property 4. Consequently, we have $r \notin \mathcal{R}(J, A\mathbf{a})$, and so part (ii) of the claim holds for $v_{C\mathbf{c}} = v_{A\mathbf{a}}$.

For the inductive step, we assume that (\diamond) holds for $n' - 1 \geq 1$, each set $X \subseteq \mathcal{V}$, and each node $v_{Aa} \in \mathcal{V}$; and we consider an arbitrary set $X \subseteq \mathcal{V}$ and vertex $v_{Aa} \in \mathcal{V}$ that satisfy properties 1–6 of (\diamond) . By property 4, there exists a rule $r \in \mathcal{P}$ such that $\text{val}(J', Aa) < \text{val}(\mathbf{T}_{\mathcal{P}}(J'), Aa) = \text{val}(\mathbf{T}_{\{r\}}(J'), Aa)$. Now if r does not generate an edge in \mathcal{E} , then in exactly the same way as in the base case we conclude that part (ii) of claim (\diamond) holds for $v_{Cc} = v_{Aa}$; consequently, in the rest of this proof we assume that r generates at least one edge in \mathcal{E} . Let $J'' = \mathbf{T}_{\mathcal{P}}^{n'+n'-1}(J')$ and let $G_{\mathcal{P}}^{J''} = (\mathcal{V}'', \mathcal{E}'', \mu'')$. Then, $\mathcal{E} = \mathcal{E}'' = \mathcal{E}'$ by property 1, and $\text{val}(\mathbf{T}_{\{r\}}(J''), Aa) \leq \text{val}(J', Aa) < \text{val}(\mathbf{T}_{\{r\}}(J'), Aa)$, so there exists an edge $e = \langle v_{Bb}, v_{Aa} \rangle \in \mathcal{E}$ such that $\text{val}(J'', Bb) < \text{val}(J', Bb)$. Furthermore, since $\text{val}(J', Aa) < \text{val}(\mathbf{T}_{\{r\}}(J'), Aa)$, if v_{Ba} were equal to v_{Aa} , then path v_{Aa}, v_{Aa} would be a positive-weight cycle containing v_{Aa} , which contradicts property 2; hence, we have $v_{Bb} \neq v_{Aa}$ and so path v_{Bb}, v_{Aa} is simple. Now if $v_{Bb} \in X$ holds, then, since r generates e and $\text{val}(\mathbf{T}_{\{r\}}(J'), Aa) = \text{val}(\mathbf{T}_{\mathcal{P}}(J'), Aa)$, by property $(*)$, we have $\text{val}(J', Bb) + \mu'(e) \geq \text{val}(\mathbf{T}_{\mathcal{P}}(J'), Aa)$ —that is, part (i) of the claim holds for $v_{Cc} = v_{Bb}$. Therefore, in the rest of this proof we assume $v_{Bb} \notin X$. We now distinguish two cases.

- v_{Bb} is reachable from v_{Aa} in $G_{\mathcal{P}}^J$. We next show that the set $X \cup \{v_{Aa}\}$ and node v_{Bb} satisfy properties 5 and 6 of the inductive hypothesis for $n' - 1$.

For property 5, note that, since v_{Bb} is the direct predecessor of v_{Aa} in $G_{\mathcal{P}}^J$, each simple path π in $G_{\mathcal{P}}^J$ that ends in v_{Bb} and does not involve v_{Aa} can be extended to the simple path π, v_{Aa} that ends in v_{Aa} . Thus, we have

$$\max\{|\pi| \mid \pi \text{ is a simple path in } G_{\mathcal{P}}^J \text{ ending in } v_{Bb} \text{ and } \pi \cap (X \cup \{v_{Aa}\}) = \emptyset\} < \max\{|\pi| \mid \pi \text{ is a simple path in } G_{\mathcal{P}}^J \text{ ending in } v_{Aa} \text{ and } \pi \cap X = \emptyset\}.$$

Property 5 for X, v_{Aa} , and n' ensures

$$\max\{|\pi| \mid \pi \text{ is a simple path in } G_{\mathcal{P}}^J \text{ ending in } v_{Aa} \text{ and } \pi \cap X = \emptyset\} \leq n',$$

which in turn implies

$$\max\{|\pi| \mid \pi \text{ is a simple path in } G_{\mathcal{P}}^J \text{ ending in } v_{Bb} \text{ and } \pi \cap (X \cup \{v_{Aa}\}) = \emptyset\} \leq n' - 1.$$

Property 6 holds for X and v_{Aa} ; moreover, there exists a path from v_{Bb} to v_{Aa} via the edge e , so the property also holds for the set $X \cup \{v_{Aa}\}$ and node v_{Bb} .

Property 3 ($v_{Bb} \notin X \cup \{v_{Aa}\}$) and property 4 ($\text{val}(J'', Bb) < \text{val}(J', Bb)$) have already been established for $X \cup \{v_{Aa}\}$, v_{Bb} , and $n' - 1$; moreover, properties 1 and 2 do not depend on X, v_{Aa} , and n' . Thus, we can apply the inductive hypothesis and conclude that one of the following holds:

- (i) $\text{val}(J'', Cc) + \mu''(\pi) \geq \text{val}(J', Bb)$ holds for some node $v_{Cc} \in X \cup \{v_{Aa}\}$ and path π in $G_{\mathcal{P}}^J$ that starts in v_{Cc} and ends in v_{Bb} ;
- (ii) $\mathcal{R}(J, Cc) \subsetneq \mathcal{R}(J'', Cc)$ holds for some node $v_{Cc} \in \mathcal{V}$.

If (ii) is true, then case (ii) of claim (\diamond) holds since $\mathcal{R}(J'', Cc) \subseteq \mathcal{R}(J', Cc)$. Thus, we next assume that case (i) holds, and we show that then part (i) of claim (\diamond) holds for v_{Cc}, X , and v_{Aa} .

We first show that $v_{Cc} \neq v_{Aa}$. For contradiction, assume $v_{Cc} = v_{Aa}$. Then $\text{val}(J'', Aa) + \mu''(\pi) \geq \text{val}(J', Bb)$. Moreover, since r generates e , by property $(*)$ and property 4, we have

$$\text{val}(J', Bb) + \mu'(e) \geq \text{val}(\mathbf{T}_{\{r\}}(J'), Aa) = \text{val}(\mathbf{T}_{\mathcal{P}}(J'), Aa) > \text{val}(J', Aa).$$

Consequently, $\text{val}(J'', Aa) + \mu''(\pi) + \mu'(e) > \text{val}(J', Aa)$. Moreover, $\text{val}(J', Aa) \geq \text{val}(J'', Aa)$ holds since $\mathbf{T}_{\mathcal{P}}$ is monotonic, and $\mu'(\pi) > \mu''(\pi)$ holds since \mathcal{P} is stable. By these observations, we have

$$\text{val}(J', Aa) + \mu'(\pi) + \mu'(e) > \text{val}(J', Aa);$$

that is, $\mu'(\pi) + \mu'(e) > 0$. But then π, v_{Aa} is a positive-weight cycle in $G_{\mathcal{P}}^J$, and so we have $\text{val}(J', Aa) = \infty$, which contradicts property 4.

Thus, we have $v_{Cc} \in X$. Then, from

$$\begin{aligned} \text{val}(J'', Cc) + \mu''(\pi) &\geq \text{val}(J', Bb) \text{ and} \\ \text{val}(J', Bb) + \mu'(e) &\geq \text{val}(\mathbf{T}_{\{r\}}(J'), Aa) = \text{val}(\mathbf{T}_{\mathcal{P}}(J'), Aa) \text{ we conclude} \\ \text{val}(J', Cc) + \mu'(\pi) + \mu'(e) &\geq \text{val}(\mathbf{T}_{\mathcal{P}}(J'), Aa) \end{aligned}$$

as in the case for $v_{Cc} = v_{Aa}$. Since $\mu'(\pi) + \mu'(e) = \mu'(\pi, v_{Aa})$, part (i) of claim (\diamond) holds for v_{Cc}, X , and v_{Aa} .

- v_{Bb} is not reachable from v_{Aa} in $G_{\mathcal{P}}^J$. Then, by property 6, v_{Bb} is not reachable in $G_{\mathcal{P}}^J$ from any node in X ; otherwise, v_{Bb} would also be reachable in $G_{\mathcal{P}}^J$ from v_{Aa} via some node in X . Thus, no simple path in $G_{\mathcal{P}}^J$ ending in v_{Bb} involves v_{Aa} or a node in X —that is, each such path can be extended to a simple path ending in v_{Aa} . Now property 5 ensures

$$\max\{|\pi| \mid \pi \text{ is a path in } G_{\mathcal{P}}^J \text{ ending in } v_{Aa} \text{ and } \pi \cap X = \emptyset\} \leq n', \text{ which implies}$$

$$\max\{|\pi| \mid \pi \text{ is a path in } G_{\mathcal{P}}^J \text{ ending in } v_{Bb}\} \leq n' - 1.$$

Thus, property 5 of the inductive hypothesis for $n' - 1$ holds for the set \emptyset and node v_{Bb} . Moreover, property 6 holds vacuously for \emptyset , properties 3 and 4 have already been established for v_{Bb} , and properties 1 and 2 hold by assumption. Thus, we can apply the inductive hypothesis for $n' - 1$ to \emptyset and v_{Bb} , and so one of the following holds:

- (i) $\text{val}(J'', Cc) + \mu''(\pi) \geq \text{val}(J', Bb)$ for some node $v_{Cc} \in \emptyset$ and path π in $G_{\mathcal{P}}^J$ that starts in v_{Cc} and ends in v_{Bb} ;
- (ii) $\mathcal{R}(J, Cc) \subsetneq \mathcal{R}(J'', Cc)$ for some node v_{Cc} in $G_{\mathcal{P}}^J$.

Clearly, (i) is trivially false, so (ii) holds. But then, case (ii) of claim (\diamond) holds since $\mathcal{R}(J'', Cc) \subseteq \mathcal{R}(J', Cc)$. \square

Note that, for each $n \geq 0$ and each simple path π in $G_{\mathcal{P}}^{\mathbf{T}_{\mathcal{P}}^n}$, $|\pi|$ is bounded by the number of nodes in $G_{\mathcal{P}}^{\mathbf{T}_{\mathcal{P}}^n}$, which is in turn bounded by $m = |\mathcal{P}|$. Therefore, claim (\diamond) for $n' = m$ and $X = \emptyset$ ensures that, for each $n \geq 0$ such that $\mathbf{T}_{\mathcal{P}}^{n+m} \neq \mathbf{T}_{\mathcal{P}}^{n+m+1}$, one of the following holds:

- 1. $\text{val}(\mathbf{T}_{\mathcal{P}}^{n+m}, Cc) \neq \infty$ for some node v_{Cc} that occurs in $G_{\mathcal{P}}^{\mathbf{T}_{\mathcal{P}}^{n+m}}$ in a positive weight cycle (so the value of the fact corresponding to v_{Cc} is set to ∞ in the next iteration of the main loop of the algorithm),
- 2. $G_{\mathcal{P}}^{\mathbf{T}_{\mathcal{P}}^{n+m}}$ contains at least one edge that does not occur in $G_{\mathcal{P}}^{\mathbf{T}_{\mathcal{P}}^n}$, or
- 3. $\mathcal{R}(\mathbf{T}_{\mathcal{P}}^n, Cc) \subsetneq \mathcal{R}(\mathbf{T}_{\mathcal{P}}^{n+m}, Cc)$ for some node v_{Cc} in $G_{\mathcal{P}}^{\mathbf{T}_{\mathcal{P}}^n}$.

For each $n \geq 0$, the size of the set $\mathcal{R}(\mathbf{T}_{\mathcal{P}}^n, Cc)$ for each node v_{Cc} and the number of nodes in $G_{\mathcal{P}}^{\mathbf{T}_{\mathcal{P}}^n}$ are both bounded by m , and the number of edges in $G_{\mathcal{P}}^{\mathbf{T}_{\mathcal{P}}^n}$ is bounded by m^2 . Thus, the number of iterations of the main loop is bounded by $m \cdot (m+1) \cdot (m^2+1) \cdot (m^2+1) \leq 8m^6$, where the first factor is given by Claim (\diamond), the second factor comes from the first case above, the third factor comes from second case, and the fourth factor comes from the third case. Hence, Algorithm 1 reaches a fixpoint after at most $8m^6$ iterations of the main loop. \square

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

Proof. Partial correctness follows by Lemma 26, while termination follows by Lemma 27. Moreover, the number of iterations of the main loop of Algorithm 1 is polynomially bounded in $|\mathcal{P} \cup \mathcal{D}|$, and hence $\|J\|$ in each such iteration is bounded by $\|\mathcal{P} \cup \mathcal{D}\|$. Consequently, lines 7 and 4 of Algorithm 1 require time that is worst-case exponential in $\max_{r \in \mathcal{P}} \|r\|$ and polynomial in $\|\mathcal{P} \cup \mathcal{D}\|$ by Lemma C.3. Moreover, lines 3, 6, and the check in line 8 require time polynomial in $\|J\|$ and hence in $\|\mathcal{P} \cup \mathcal{D}\|$. Finally, we argue that the check for positive-weight cycles in line 5 is feasible in time polynomial in $\|\mathcal{P} \cup \mathcal{D}\|$. Let G' be the graph obtained from $G_{\mathcal{P}}^J$ by negating all weights. Then, a maximal-weight path from v_{v_1} to v_{v_2} in $G_{\mathcal{P}}^J$ corresponds to the least-weight path from v_{v_1} to v_{v_2} in G' . Thus, detecting whether a node occurs in $G_{\mathcal{P}}^J$ in at least one positive-weight cycle reduces to detecting whether the node occurs in G' on a negative cycle (i.e., on a cycle with a negative sum of weights), which can be solved in polynomial time using, for example, a variant of the Floyd-Warshall algorithm [Hougardy, 2010]. \square

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

Proof. The EXPTIME lower bound in combined complexity and the PTIME lower bound in data complexity are inherited from plain Datalog [Dantsin *et al.*, 2001]. The PTIME upper bound in data is immediate by Theorem 28. For the EXPTIME upper bound in combined complexity, note that, for $\mathcal{P}' = \mathcal{P}'_0 \cup \mathcal{D}$ the semi-grounding of $\mathcal{P} = \mathcal{P}_0 \cup \mathcal{D}$ over constants in \mathcal{P} , we have that $\|\mathcal{P}'\|$ is exponentially bounded in $\|\mathcal{P}\|$, whereas $\max_{r \in \mathcal{P}'_0} \|r\| = \max_{r \in \mathcal{P}_0} \|r\|$. Hence, by Theorem 28, running Algorithm 1 on \mathcal{P}' gives us an exponential-time decision procedure for $\mathcal{P} \models \alpha$. \square

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

Proof. We present a reduction from Hilbert's tenth problem, which is to determine whether, given a polynomial $P(x_1, \dots, x_n)$ over variables x_1, \dots, x_n , equation $P(x_1, \dots, x_n) = 0$ has integer solutions. For each such polynomial P , we can assume without loss of generality that P is of the form $\sum_j (c_j \times \prod_{i=1}^n x_i^{k_{j,i}})$ for $c_j \in \mathbb{Z}$ and $k_{j,i} \geq 0$. Now let \mathcal{P}_P be the program containing the following rule, where B is a unary max predicate, and A_1, \dots, A_n are distinct unary ordinary numeric predicates:

$$A_1(x_1) \wedge \dots \wedge A_n(x_n) \wedge (P(x_1, \dots, x_n) \leq 0) \wedge (P(x_1, \dots, x_n) \geq 0) \wedge B(m) \wedge (m \leq 0) \rightarrow B(m+1) \quad (68)$$

Note that rule (68) is limit-linear since variables x_1, \dots, x_n do not occur in a limit atom in the rule. We show that \mathcal{P}_P is stable if and only if $P(x_1, \dots, x_n) = 0$ has no integer solutions.

Assume that $P(x_1, \dots, x_n) = 0$ has no integer solutions. Then, for each grounding σ of (68), at least one of the first two comparison atoms in the rule is not satisfied, and so \mathcal{P}_P is trivially stable since, for each pseudo-interpretation J , the value propagation graph $G_{\mathcal{P}}^J$ does not contain any edges.

Assume that substitution σ exists such that $P(\sigma(x_1), \dots, \sigma(x_n)) = 0$ holds, and let J_1 and J_2 be the following pseudo-interpretations with the corresponding value propagation graphs:

$$J_1 = \{A_1(\sigma(x_1)), \dots, A_1(\sigma(x_n))\} \cup \{B(0)\} \quad G_{\mathcal{P}_P}^{J_1} = (\mathcal{V}_1, \mathcal{E}_1, \mu_1) \quad (69)$$

$$J_2 = \{A_1(\sigma(x_1)), \dots, A_1(\sigma(x_n))\} \cup \{B(1)\} \quad G_{\mathcal{P}_P}^{J_2} = (\mathcal{V}_2, \mathcal{E}_2, \mu_2) \quad (70)$$

Then, we clearly have $J_1 \sqsubseteq J_2$ and $e = \langle v_B, v_B \rangle \in \mathcal{E}_1 = \mathcal{E}_2$; however, $\mu_1(e) = 1$ and $\mu_2(e) = 0$. Consequently, program \mathcal{P}_P is not stable. \square

Lemma D.3. *For each pseudo-interpretation J and each semi-ground type-consistent rule r , if r is applicable to J and it contains a limit body atom $B(\mathbf{b}, n) \in \text{sb}(r)$ such that $B(\mathbf{b}, \infty) \in J$ and variable n occurs in $\text{h}(r)$, then $\text{opt}(r, J) = \infty$.*

Proof. Consider an arbitrary pseudo-interpretation J and rule r applicable to J that contains a limit body atom $B(\mathbf{b}, n) \in \text{sb}(r)$ with $B(\mathbf{b}, \infty) \in J$ and n occurring in $\text{h}(r)$. We consider the case when $\text{h}(r) = A(\mathbf{a}, s)$ for A a max predicate and variable n occurs in s with a negative coefficient; the cases when A is a min predicate and/or n occurs in s with a positive coefficient are analogous. Then, term s has the form $\ell \times n + t$ for some negative integer ℓ and term t not containing n . Moreover, r is type-consistent, so B is a min predicate. Since r is applicable to J , conjunction $\mathcal{C}(r, J)$ has a solution σ . We next show that $\text{opt}(r, J) = \infty$ holds, for which it suffices to argue that, for each $k \in \mathbb{Z}$, conjunction $\mathcal{C}(r, J)$ has a solution σ' such that $s\sigma' \geq k$. Let $k_0 = s\sigma$ and let σ' be the grounding of $\mathcal{C}(r, J)$ such that $\sigma'(n) = \sigma(n) - |k - k_0|$ and $\sigma'(m) = \sigma(m)$ for each variable $m \neq n$. Since $B(\mathbf{b}, \infty) \in J$, we have $J \models B(\mathbf{b}, \sigma'(n))$. Moreover, r is type-consistent, σ satisfies all comparison atoms in the body of r , B is a min predicate, and $\sigma'(n) \leq \sigma(n)$, so σ' also satisfies all comparison atoms in the body of r . Hence, σ' is a solution to $\mathcal{C}(r, J)$. Then, the following calculation implies the claim of this lemma:

$$s\sigma' = \ell \times (\sigma(n) - |k - k_0|) + t\sigma = (\ell \times \sigma(n) + t\sigma) - \ell \times |k - k_0| = k_0 - \ell \times |k - k_0| \geq k_0 + |k - k_0| \geq k. \quad \square$$

Lemma D.4. *For each type-consistent, limit-linear rule r with $\text{h}(r) = A(\mathbf{t}, s)$, each limit body atom $B(\mathbf{t}', n) \in \text{sb}(r)$ such that n occurs in s , each semi-grounding σ of r such that $n \notin \text{dom}(\sigma)$, and all pseudo-interpretations J_1 and J_2 such that $J_1 \sqsubseteq J_2$ and $r\sigma$ is applicable to J_1 , we have $\delta_{r\sigma}^e(J_1) \leq \delta_{r\sigma}^e(J_2)$ where $e = \langle v_{B\mathbf{t}'\sigma}, v_{A\mathbf{t}\sigma} \rangle$.*

Proof. Consider arbitrary r , $B(\mathbf{t}', n)$, σ , J_1 , J_2 , and e as stated in the lemma. We consider the case when A is a max and B is a min predicate; the remaining cases are analogous. Let φ be the body of r and let σ be a semi-grounding of r . Moreover, the claim is trivial if $\delta_{r\sigma}^e(J_2) = \infty$, so we next assume $\delta_{r\sigma}^e(J_2) \neq \infty$. Due to $J_1 \sqsubseteq J_2$, each solution to $\mathcal{C}(r\sigma, J_1)$ is a solution to $\mathcal{C}(r\sigma, J_2)$ as well, so therefore $\delta_{r\sigma}^e(J_1) \neq \infty$ holds. By Definition 23, we then have $\text{opt}(r\sigma, J_1) \neq \infty$ and $\text{opt}(r\sigma, J_2) \neq \infty$. But then, by Lemma D.3, there exist $k_1, k_2 \in \mathbb{Z}$ such that $B(\mathbf{t}'\sigma, k_1) \in J_1$ and $B(\mathbf{t}'\sigma, k_2) \in J_2$; since B is a min predicate, we have $k_2 \leq k_1$; moreover, by Definition 23 we have $\delta_{r\sigma}^e(J_1) = \text{opt}(r, J_1) + k_1$ and $\delta_{r\sigma}^e(J_2) = \text{opt}(r, J_2) + k_2$. Rule r is type-consistent, so variable n occurs negatively in s ; thus, $s\sigma$ is of the form $s' \times n + s''$ where s' is a ground product evaluating to a negative integer and s'' does not mention n . Moreover, $\text{opt}(r\sigma, J_1) \neq \infty$, so there exists a grounding σ_1 of $r\sigma$ such that $J_1 \models \varphi\sigma\sigma_1$ and $\text{opt}(r\sigma, J_1) = s\sigma\sigma_1 = s' \times k_1 + s''\sigma_1 = \delta_{r\sigma}^e(J_1) - k_1$. Let σ_2 be the substitution such that $\sigma_2(n) = k_2$ and $\sigma_2(m) = \sigma_1(m)$ for $m \neq n$. Clearly, J_2 satisfies all object and numeric atoms in $\varphi\sigma\sigma_2$. Then, we have the following:

$$\delta_{r\sigma}^e(J_2) = \text{opt}(r\sigma, J_2) + k_2 \geq s\sigma\sigma_2 + k_2 = s' \times n\sigma_2 + s''\sigma_2 + k_2 = s' \times k_2 + s''\sigma_1 + k_2$$

Furthermore, we have already established $s' \times k_1 + s''\sigma_1 = \delta_{r\sigma}^e(J_1) - k_1$, which implies the following:

$$\delta_{r\sigma}^e(J_2) \geq s' \times k_2 - s' \times k_1 + \delta_{r\sigma}^e(J_1) - k_1 + k_2 = (s' + 1) \times (k_2 - k_1) + \delta_{r\sigma}^e(J_1)$$

But then, $s' < 0$ and $k_2 \leq k_1$ clearly imply $\delta_{r\sigma}^e(J_2) \geq \delta_{r\sigma}^e(J_1)$, as required. \square

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

Proof. For \mathcal{P} a type-consistent program and σ a semi-grounding of \mathcal{P} , condition 1 of Definition 24 follows by Lemma D.4, and condition 2 of Definition 24 follows by Lemma D.3. \square

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

Proof. Let \mathcal{P} be a limit-linear program. We can check whether \mathcal{P} is type-consistent by considering each rule $r \in \mathcal{P}$ independently. Note that the first type consistency condition is satisfied for every semi-ground limit-linear rule where all numeric terms are simplified as much as possible; thus, no semi-grounding of r (with constants from \mathcal{P}) where all numeric terms are simplified as much as possible can violate the first condition of Definition 31. Thus, it suffices to check whether a semi-grounding of r (with constants from \mathcal{P}) can violate the second or the third condition. In both cases, it suffices to consider at most one atom α at a time (a limit head atom $A(\mathbf{a}, s)$ for the second condition or a comparison atom $s_1 < s_2$ or $s_1 \leq s_2$ for the third condition). In α , we consider at most one numeric term s at a time ($s \in \{s_1, s_2\}$ for the third condition), where s is of the form $t_0 + \sum_{i=1}^n t_i \times m_i$ and t_i , for $i \geq 1$, are terms constructed from integers, variables not occurring in limit atoms, and multiplication. Moreover, for each such s , we consider each variable m occurring in s .

By assumption, m occurs in s , so we have $m_i = m$ for some i . For the second condition of Definition 31, we need to check that, if the limit body atom $B(s, m_i)$ introducing m_i has the same (different) type as the head atom, then term t_i can only be grounded to positive (negative) integers or zero. For the third condition, we need to check that, if $s = s_1$ and the limit body atom $B(s, m_i)$ introducing m_i is min (max), then term t_i can only be grounded to positive (negative) integers or 0, and dually for the case $s = s_2$. Hence, in either case, it suffices to check whether term t_i can be semi-grounded so that it evaluates to a positive integer, a negative integer, or zero. We next discuss how this can be checked in logarithmic space. Let $t_i = t_i^1 \times \cdots \times t_i^k$, where each t_i^j is an integer or a variable not occurring in a limit atom, and assume without loss of generality that we want to check whether t_i can be grounded to a positive integer; this is the case if and only if one of the following holds:

- all t_i^j are integers whose product is positive;
- the product of all integers in t_i is positive and \mathcal{P} contains a positive integer;
- the product of all integers in t_i is positive, \mathcal{P} contains a negative integer, and the total number of variable occurrences in t_i is even;
- the product of all integers in t_i is negative, \mathcal{P} contains a negative integer, and the total number of variable occurrences in t_i is odd; or
- the product of all integers in t_i is negative, \mathcal{P} contains both positive and negative integers, and some variable t_i^j has an odd number of occurrences in t_i .

Each of these conditions can be verified using a constant number of pointers into \mathcal{P} and binary variables. This clearly requires logarithmic space, and it implies our claim. \square