Decidable Classes of Datalog Programs with Arithmetic

Mark Kaminski
Department of Computer Science, University of Oxford, UK

joint work with Bernardo Cuenca Grau, Egor V. Kostylev, Boris Motik, and Ian Horrocks

1 Introduction

Motivated by applications in declarative data analysis, we study $Datalog_{\mathbb{Z}}$—negation-free Datalog with integer arithmetic and comparisons. As this language is known to be undecidable, we propose a new *limit* $Datalog_{\mathbb{Z}}$ fragment that, like the existing data analysis languages studied in [2, 4, 6, 7, 10–14], is powerful and flexible enough to naturally capture many important analysis tasks. However, unlike $Datalog_{\mathbb{Z}}$ and the existing languages, reasoning with limit programs is decidable under mild restrictions on the use of multiplication, and it becomes tractable in data complexity with additional restrictions on the use of variables.

In limit $Datalog_{\mathbb{Z}}$, all intensional predicates with a numeric argument are *limit predicates* that keep the minimal (min) or the maximal (max) bounds of numeric values. 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.

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

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

If these rules and a dataset entail a fact $sp(v, k)$, then the cost of a shortest path from v_0 to v is at most k; hence, $sp(v, k')$ holds for each $k' \geq k$ since the cost of a shortest path is also at most k'. Rule (2) intuitively says that, if x is reachable from v_0 with cost at most m and $\langle x, y\rangle$ is an edge of cost n, then v' is reachable from v_0 with cost at most $m + n$. This is different from $Datalog_{\mathbb{Z}}$, where there is no implicit semantic connection between $sp(v, k)$ and $sp(v, k')$, and such semantic connections allow us to prove decidability of limit $Datalog_{\mathbb{Z}}$. We provide a direct semantics for limit predicates based on Herbrand interpretations, but we also show that this semantics can be axiomatised in standard $Datalog_{\mathbb{Z}}$. Our formalism can thus be seen as a fragment of $Datalog_{\mathbb{Z}}$, from which it inherits well-understood properties such as monotonicity and existence of a least fixpoint model [5]. We prove that fact entailment in limit $Datalog_{\mathbb{Z}}$ is undecidable, but, after restricting the use of multiplication, it becomes coNEXPTIME-complete in combined complexity and coNP-complete in data complexity. These results are presented in more detail in [9].

What if we want to check whether the cost of the shortest path from v_0 to v is *precisely* k? Our formalism so far does not allow to do this via a single query. Thus, we consider a non-monotonic *least upper bound* (LUB) operator that allows us to extract the least/greatest value that is entailed for a given predicate and objects. We provide a straightforward semantics for the operator by imposing a stratification requirement on its use in programs. We show that fact entailment in our decidable fragment of limit $Datalog_{\mathbb{Z}}$ extended by the LUB operator is Δ_2^{P}-complete in data complexity. Our efforts to determine the combined complexity of the fragment lead us to an ongoing analysis of its descriptive complexity and raise questions about its expressive power over metafinite structures, which we want to answer as part of future work.

2 Preliminaries

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

Syntax A vocabulary consists of *predicates*, *objects*, *object variables*, and *numeric variables*. Each predicate has an integer *arity* n, and each position $1 \leq i \leq n$ is of either *object* or *numeric sort*. An *object term* is an object or an object variable. A *numeric term* is an integer, a numeric variable, or of the form $s_1 + s_2$, $s_1 - s_2$, or $s_1 \times s_2$ where s_1 and s_2 are numeric terms, and $+$, $-$, and $\times$ are the standard *arithmetic functions*. A *constant* is an object or an integer. The *magnitude* of an integer is its absolute value. A *standard atom* is of the form $B(t_1, \ldots, t_n)$, where B is a predicate of arity n and each t_i is a term whose type matches the sort of position i of B. A *comparison atom* is of the form $(s_1 < s_2)$ or $(s_1 \leq s_2)$, where $<$ and $\leq$ are the standard *comparison predicates*, and s_1 and s_2 are numeric terms. A *rule* r is of the form $\bigwedge_i \alpha_i \wedge \bigwedge_j \beta_j \rightarrow \alpha$, where $\alpha_{(i)}$ are standard atoms, β_j are comparison atoms, and each variable in r occurs in some α_i. Atom α is the *head* of r, and the conjunction $\bigwedge_i \alpha_i \wedge \bigwedge_j \beta_j$ is the *body* of r. 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, for each $r \in \mathcal{P}$, the head of r is a fact and the body of r is empty. We sometimes say that $\mathcal{P}$ contains a fact α, which actually means that $\mathcal{P}$ contains the rule $\rightarrow \alpha$.

Semantics A *(Herbrand) interpretation* I is a (not neces-

sarily 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 We consider 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 $+$ [5].

3 Limit Programs

Note that fact entailment in $Datalog_{\mathbb{Z}}$ is undecidable even if atoms contain at most one numeric term:

Proposition 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* limit ($Datalog_{\mathbb{Z}}$) rule *is either a rule whose body is empty, or a rule where all standard body atoms are object, ordinary numeric, or limit, and the head is an object or 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.*

W.l.o.g., 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.

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 $Datalog_{\mathbb{Z}}$.

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

For simplicity, in the following we assume all programs to be homogeneous, involving only max predicates. This assumption can be done without loss of generality as each limit program can be reduced to a homogeneous one:

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 $\mathcal{P}' \models \alpha'$.*

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}. We assume that all objects in $\mathcal{D}_{tw}$ are arranged in an arbitrary linear order using facts $first(a_1)$, $next(a_1, a_2)$, …, $next(a_{n-1}, a_n)$. Program $\mathcal{P}_{tw}$, containing rules* (3)–(8)*, encodes message propagation, where nt is a* max *predicate.*

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

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

4 Fixpoint Characterisation of Entailment

We next introduce the *immediate consequence* operator $\mathbf{I}_{\mathcal{P}}$ of a limit program $\mathcal{P}$ on limit-closed interpretations.

Definition 6. *For $\mathcal{P}$ a $Datalog_{\mathbb{Z}}$ program and I a limit-closed interpretation, we write $\mathbf{I}_{\mathcal{P}}(I)$ for the limit closure of the set*

$$\{\alpha \mid \varphi \to \alpha \text{ ground instance of a rule in } \mathcal{P} \text{ s.t. } I \models \varphi\}$$

We define $\mathbf{I}^0_{\mathcal{P}} = \emptyset$ and $\mathbf{I}^n_{\mathcal{P}} = \mathbf{I}_{\mathcal{P}}(\mathbf{I}^{n-1}_{\mathcal{P}})$ for $n > 0$.

It is easily seen that the operator $\mathbf{I}_{\mathcal{P}}$ is monotonic w.r.t. $\subseteq$. Moreover, $I \models \mathcal{P}$ iff $\mathbf{I}_{\mathcal{P}}(I) \subseteq I$. Monotonicity ensures existence of the *closure* $\mathbf{I}^\infty_{\mathcal{P}}$ of $\mathcal{P}$—the least limit-closed interpretation such that $\mathbf{I}^n_{\mathcal{P}} \subseteq \mathbf{I}^\infty_{\mathcal{P}}$ for each $n \geq 0$. Clearly, $\mathbf{I}^\infty_{\mathcal{P}}$ is the least model of $\mathcal{P}$.

Proposition 7. *For $\mathcal{P}$ a limit program and α a fact, we have $\mathcal{P} \models \alpha$ iff $\mathbf{I}^\infty_{\mathcal{P}} \models \alpha$.*

Importantly, for each $n \in \mathbb{N} \cup \{\infty\}$, $\mathbf{I}^n_{\mathcal{P}}$ can be finitely represented by a set J of facts containing each object and ordinary numeric fact in $\mathbf{I}^n_{\mathcal{P}}$, a max fact $A(\mathbf{a}, k)$ whenever $A(\mathbf{a}, k) \in \mathbf{I}^n_{\mathcal{P}}$ and $A(\mathbf{a}, k+1) \notin \mathbf{I}^n_{\mathcal{P}}$, and a max fact $A(\mathbf{a}, \dot\infty)$ whenever $A(\mathbf{a}, k) \in \mathbf{I}^n_{\mathcal{P}}$ for each $k \in \mathbb{Z}$, where $\dot\infty$ is a special symbol distinct from numbers in $\mathbb{Z}$. Clearly, each fact in J is finitely representable. Moreover, $|J|$ is exponentially bounded in $\|\mathcal{P}\|$ (and polynomially in data).

5 Decidability of Entailment: Limit-Linearity

Due to multiplication, checking rule applicability requires solving nonlinear inequalities over integers, which is undecidable.

Theorem 8. *For $\mathcal{P}$ a limit program and α a fact, checking $\mathcal{P} \models \alpha$ and checking applicability of a rule of $\mathcal{P}$ to a limit-closed interpretation are both undecidable.*

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 numeric terms contain no products between variables occurring in limit atoms. On the other hand, multiplication between variables occurring only in ordinary numeric atoms is harmless as such variables can be eliminated by grounding.

Definition 9. *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 each m_i is a distinct numeric variable occurring in r in a limit atom, term s_0 contains no variable occurring in r in a limit atom, and each s_i with $i \geq 1$ is a term constructed from integers, variables not occurring in r in limit atoms, and multiplication $\times$. A limit-linear program contains only limit-linear rules.*[1]

Entailment for limit-linear programs is decidable. By Proposition 7 and the observations at the end of Section 4, it suffices to consider interpretations representable by sets of bounded cardinality. Additionally, limit-linearity allows us to bound the magnitude of numbers occurring in such sets.

It can be shown that for limit-linear programs, fact entailment can be reduced to validity of Presburger formulas of a certain shape. The magnitude of integers in models of such formulas can be bounded double-exponentially in the size of the original program and exponentially in data, which yields the following lemma:

Lemma 10. *For $\mathcal{P}$ a limit-linear program, $\mathcal{D}$ a dataset, and α a fact, $\mathcal{P} \cup \mathcal{D} \not\models \alpha$ if and only if a model I of $\mathcal{P} \cup \mathcal{D}$ exists such that $I \not\models \alpha$ and I can be finitely represented by a set J such that $|J| \leq 2 \cdot |\mathcal{P}| \cdot |\mathcal{D}|^{\|\mathcal{P}\|}$ and the magnitude of each integer in J is bounded polynomially in the largest magnitude of an integer in $\mathcal{P} \cup \mathcal{D}$, exponentially in $|\mathcal{P}| \cdot |\mathcal{D}|^{\|\mathcal{P}\|}$, and double-exponentially in $\max_{r \in \mathcal{P}} \|r\|$.*

By Lemma 10, the following nondeterministic algorithm decides $\mathcal{P} \not\models \alpha$:

1. guess the finite representation of a limit-closed interpretation I that satisfies the bounds given in Lemma 10;
2. if $\mathbf{I}_{\mathcal{P}}(I) \subseteq I$ (so $I \models \mathcal{P}'$) and $I \not\models \alpha$, return true.

Step 2 is nondeterministic exponential (polynomial in data), and Step 3, which can be performed on the finite representation of I, boils down to solving systems of linear inequalities, which also requires exponential (polynomial in data) time.

Theorem 11. *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.*

By restricting limit-linear programs further, we can obtain a practically useful fragment for which the complexity of fact entailment goes down to ExpTime in combined complexity and PTime in data complexity, as in plain Datalog. In particular, the fragment captures the program in Example 5.

6 Accessing Least Upper Bounds

We next introduce the least upper bound operator $\lceil . \rceil$. Intuitively, a fact $\lceil A(k) \rceil$, for A a max predicate and $k \in \mathbb{Z}$, is satisfied by a limit-closed interpretation I if k is the greatest number for which predicate A is true in I.

Definition 12. *A* least upper bound atom *(*LUB atom*) has the form $\lceil \alpha \rceil$ for α a* max *atom. A limit-closed interpretation I satisfies a ground atom $\lceil A(\mathbf{a}, k) \rceil$ if $k \in \mathbb{Z}$, $I \models A(\mathbf{a}, k)$, and $I \not\models A(\mathbf{a}, k+1)$.*

It is easily seen that entailment of LUB facts by (LUB-free) limit-linear programs is DP-complete in data complexity.

We now study what happens if we allow LUB atoms to occur in programs. For this, we extend our definition of programs, admitting LUB atoms in rule bodies. As with standard atoms, we assume, w.l.o.g., that each numeric variable in a rule occurs in at most one LUB body atom, and that LUB atoms share no variables with standard body atoms.

In the presence of LUB atoms, the immediate consequence operator becomes non-monotonic since such atoms can simulate negation as failure. Hence, we restrict ourselves to programs where applications of $\lceil . \rceil$ are stratified in the usual way: if $\lceil B(\mathbf{t}) \rceil$ occurs in the body of a rule and $A(\mathbf{s})$ in the head, then B must have a strictly lower stratum than A.

We can then define the stratified *materialisation* $\mathbf{S}^\infty_{\mathcal{P}}$ of $\mathcal{P}$ in an intuitive way.

Definition 13. *A stratified (limit) program $\mathcal{P}$* entails *a fact α, written $\mathcal{P} \models \alpha$, if $\alpha \in \mathbf{S}^\infty_{\mathcal{P}}$.*

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

The notion of limit-linearity can be relaxed for stratified programs by additionally allowing the terms s_i in Definition 9 to contain numeric variables occurring in LUB atoms. We call such programs *stratified-linear*.

Given that entailment of LUB facts by LUB-free limit-linear programs is DP-complete in data complexity, the following fact is unsurprising:

Theorem 14. *For $\mathcal{P}$ a stratified-linear program and α a fact, deciding $\mathcal{P} \models \alpha$ is Δ_2^{P}-complete in data complexity.*

What about combined complexity? It is not difficult to see that fact entailment for stratified-linear programs is feasible in Δ_2^{EXP} (a class in the weak EXP hierarchy). Hardness for Δ_2^{EXP} follows in a natural way provided stratified-linear programs capture Δ_2^{P}.

Definition 15. *Let γ be a fixed nullary fact and let $\mathbf{Dat}$ be a class of datasets not mentioning γ. We define the subclass of datasets* expressed *by a program $\mathcal{P}$ (possibly mentioning γ) over $\mathbf{Dat}$ as $\mathbf{Dat}(\mathcal{P}) = \{\mathcal{D}$ dataset in $\mathbf{Dat} \mid \mathcal{P} \cup \mathcal{D} \models \gamma\}$. A class of programs $\mathbf{Prog}$* captures *a complexity class C over $\mathbf{Dat}$ if for every subclass $\mathbf{Dat}'$ of $\mathbf{Dat}$ it holds that $\mathbf{Dat}'$ is recognisable in C iff $\mathbf{Dat}' = \mathbf{Dat}(\mathcal{P})$ for some $\mathcal{P}$ in $\mathbf{Prog}$.*

By the results in [8], we then obtain the following lemma.

Lemma 16. *If stratified-linear programs capture Δ_2^{P} over ordered datasets containing no numeric facts, then deciding $\mathcal{P} \models \alpha$, for $\mathcal{P}$ stratified-linear and α a fact, is Δ_2^{EXP}-hard.*

Thus, we are currently working on the proof of the following conjecture.

Conjecture 17. *Stratified-linear programs capture Δ_2^{P} on ordered datasets containing no numeric facts.*

7 Connections to Metafinite Model Theory

While Conjecture 17 is restricted to datasets free of numeric facts, we see no reason why it should be false for datasets that have numeric facts, which brings up the following questions:

- Do stratified-linear programs capture Δ_2^{P} (on ordered datasets) in the metafinite case?
- Can we extend limit-linear programs (e.g., with negation on EDB atoms) to capture coNP over datasets with numbers?
- Can we extend our tractable subclass of limit-linear programs to capture PTIME over datasets with numbers?
- Are there other natural classes of metafinite structures that can be captured by (variants of) our formalism?
- What happens if we consider reals instead of integers?

Furthermore, we are interested in exploring the computational properties and expressive power of variants of our language that give up limit-linearity, i.e., allow for multiplication between variables in limit atoms. Decidability of such formalisms can be re-gained by changing the definition of rule applicability in a way consistent with the formalism in [13]. The computational properties of such logics seem to be related to the problem PosSLP studied in [1], and thus to the Blum-Shub-Smale model of computation [3].

Acknowledgments

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

References

[1] E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P. B. Miltersen. On the complexity of numerical analysis. *SIAM J. Comput.*, 38(5):1987–2006, 2009.

[2] C. Beeri, S. A. Naqvi, O. Shmueli, and S. Tsur. Set constructors in a logic database language. *J. Log. Program.*, 10(3&4):181–232, 1991.

[3] L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bull. Am. Math. Soc.*, 21(1):1–46, 1989.

[4] M. P. Consens and A. O. Mendelzon. Low complexity aggregation in GraphLog and Datalog. *Theor. Comput. Sci.*, 116(1):95–116, 1993.

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

[6] W. Faber, G. Pfeifer, and N. Leone. Semantics and complexity of recursive aggregates in answer set programming. *Artif. Intell.*, 175(1):278–298, 2011.

[7] S. Ganguly, S. Greco, and C. Zaniolo. Extrema predicates in deductive databases. *J. Comput. Syst. Sci.*, 51(2):244–259, 1995.

[8] G. Gottlob, N. Leone, and H. Veith. Succinctness as a source of complexity in logical formalisms. *Ann. Pure Appl. Log.*, 97(1-3):231–260, 1999.

[9] M. Kaminski, B. Cuenca Grau, E. V. Kostylev, B. Motik, and I. Horrocks. Foundations of declarative data analysis using limit datalog programs. In *IJCAI 2017*. To appear.

[10] D. B. Kemp and P. J. Stuckey. Semantics of logic programs with aggregates. In V. A. Saraswat and K. Ueda, editors, *ISLP*, pages 387–401. MIT Press, 1991.

[11] M. Mazuran, E. Serra, and C. Zaniolo. Extending the power of datalog recursion. *VLDB J.*, 22(4):471–493, 2013.

[12] I. S. Mumick, H. Pirahesh, and R. Ramakrishnan. The magic of duplicates and aggregates. In *VLDB*, pages 264–277, 1990.

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

[14] A. Van Gelder. The well-founded semantics of aggregation. In M. Y. Vardi and P. C. Kanellakis, editors, *PODS*, pages 127–138. ACM Press, 1992.