Datalog: Bag Semantics via Set Semantics

Leopoldo Bertossi
RelationalAI Inc., USA
Carleton University, Ottawa, Canada
Member of the "Millenium Institute for Foundational Research on Data" (IMFD, Chile)
bertossi@scs.carleton.ca

Georg Gottlob
University of Oxford, UK
TU Wien, Austria
georg.gottlob@cs.ox.ac.uk

Reinhard Pichler
TU Wien, Austria
pichler@dbai.tuwien.ac.at

Abstract

Duplicates in data management are common and problematic. In this work, we present a translation of Datalog under bag semantics into a well-behaved extension of Datalog, the so-called *warded Datalog*$^\pm$, under set semantics. From a theoretical point of view, this allows us to reason on bag semantics by making use of the well-established theoretical foundations of set semantics. From a practical point of view, this allows us to handle the bag semantics of Datalog by powerful, existing query engines for the required extension of Datalog. This use of Datalog$^\pm$ is extended to give a set semantics to duplicates in Datalog$^\pm$ itself. We investigate the properties of the resulting Datalog$^\pm$ programs, the problem of deciding multiplicities, and expressibility of some bag operations. Moreover, the proposed translation has the potential for interesting applications such as to Multiset Relational Algebra and the semantic web query language SPARQL with bag semantics.

2012 ACM Subject Classification Information systems → Query languages; Theory of computation → Logic; Theory of computation → Semantics and reasoning

Keywords and phrases Datalog, duplicates, multisets, query answering, chase, Datalog$^\pm$

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.16

Related Version A full version of the paper is available at `http://arxiv.org/abs/1803.06445`.

Acknowledgements Many thanks to Renzo Angles and Claudio Gutierrez for information on their work on SPARQL with bag semantics; and to Wolfgang Fischl for his help testing some queries in SQL DBMSs. We appreciate the useful comments received from the reviewers. Part of this work was done while L. Bertossi was spending a sabbatical at the DBAI Group of TU Wien with support from the "Vienna Center for Logic and Algorithms" and the Wolfgang Pauli Society. This author is grateful for their support and hospitality, and specially to G. Gottlob for making the stay possible. He was also supported by NSERC Discovery Grant #06148. The work of G. Gottlob was supported by the Austrian Science Fund (FWF):P30930 and the EPSRC programme grant EP/M025268/1 VADA. The work of R. Pichler was supported by the Austrian Science Fund (FWF):P30930.

1 Introduction

Duplicates are a common feature in data management. They appear, for instance, in the result of SQL queries over relational databases or when a SPARQL query is posed over RDF data. However, the semantics of data operations and queries in the presence of duplicates is not always clear, because duplicates are handled by bags or multisets, but common logic-based semantics in data management are set-theoretical, making it difficult to tell apart duplicates through the use of sets alone. To address this problem, a bag semantics for Datalog programs was proposed in [21], what we refer to as the *derivation-tree bag semantics* (DTB semantics). Intuitively, two duplicates of the same tuple in an intentional predicate are accepted as such if they have syntactically different derivation trees. The DTB semantics was used in [2] to provide a bag semantics for SPARQL.

The DTB semantics follows a proof-theoretic approach, which requires external, meta-level reasoning over the set of all derivation trees rather than allowing for a query language that inherently collects those duplicates. The main goal of this paper is to identify a syntactic class of extended Datalog programs, such that: (a) it extends classical Datalog with stratified negation and has a classical model-based semantics, (b) for every program in the class with a bag semantics, another program in the same class can be built that has a set-semantics and fully captures the bag semantics of the initial program, (c) it can be used in particular to give a set-semantics for classical Datalog with stratified negation with bag semantics.

To this end, we show that the DTB semantics of a Datalog program can be represented by means of its transformation into a Datalog$^\pm$ program [8, 10], in such a way that the intended model of the former, including duplicates, can be characterized as the result of the duplicate-free chase instance for the latter. The crucial idea of our translation from bag semantics into set semantics (of Datalog$^\pm$) is the introduction of tuple ids (tids) via existentially quantified variables in the rule heads. Different tids of the same tuple will allow us to identify usual duplicates when falling back to a bag semantics for the original Datalog program. We establish the correspondence between the DTB semantics and ours. This correspondence is then extended to Datalog with stratified negation. We thus recover full relational algebra (including set difference) with bag semantics in terms of a well-behaved query language under set semantics.

The programs we use for this task belong to *warded* Datalog$^\pm$ [17]. This is a particularly well-behaved class of programs in that it properly extends Datalog, has a tractable conjunctive query answering (CQA) problem, and has recently been implemented in a powerful query engine, namely the VADALOG System [6, 7]. None of the other well-known classes of Datalog$^\pm$ share these properties: for instance, guarded [8], sticky and weakly-sticky [11] Datalog$^\pm$ only allow restricted forms of joins and, hence, do not cover Datalog. On the other hand, more expressive languages, such as weakly frontier guarded Datalog$^\pm$ [5], lose tractability of CQA. Warded Datalog$^\pm$ has been successfully applied to represent a core fragment of SPARQL under certain OWL 2 QL entailment regimes [16], with set semantics though [17] (see also [3, 4]), and it looks promising as a general language for specifying different data management tasks [6]. We then go one step further and also express the bag semantics of Datalog$^\pm$ by means of the set semantics of Datalog$^\pm$.

Structure and main results. In Section 2, we recall some basic notions. In Section 7, we conclude and discuss some future extensions. Our main results are detailed in Sections 3 – 6:

- Our translation of Datalog with bag semantics into warded Datalog$^\pm$ with set semantics, which will be referred to as program-based bag (PBB) semantics, is presented in Section 3. We also show how this translation can be extended to Datalog with stratified negation.
- In Section 4, we study the transformation from bag semantics into set semantics for Datalog$^\pm$ itself. We thus carry over both the DTB semantics and the PBB semantics to Datalog$^\pm$ with a form of stratified negation, and establish the equivalence of these two semantics also for this extended query language. Moreover, we verify that the Datalog$^\pm$ programs resulting from our transformation are warded whenever the programs to start

with belong to this class.

- In Section 5, we study crucial decision problems related to multiplicities. Above all, we are interested in the question if a given tuple has finite or infinite multiplicity. Moreover, in case of finiteness, we want to compute the precise multiplicity. We show that these tasks can be accomplished in polynomial time (data complexity).
- In Section 6, we apply our results on Datalog with bag semantics to Multiset Relational Algebra (MRA). We also discuss alternative semantics for multiset-intersection and multiset-difference, and the difficulties to capture them with our Datalog$^\pm$ approach.

2 Preliminaries

We assume familiarity with the relational data model, conjunctive queries (CQs), in particular Boolean conjunctive queries (BCQs); classical Datalog with minimal-model semantics, and Datalog with stratified negation with standard-model semantics, denoted Datalog$^{\neg s}$ (see [1] for an introduction). An n-ary relational predicate P has *positions*: $P[1], \ldots, P[n]$. With $\mathit{Pos}(P)$ we denote the set of positions of predicate P; and with $\mathit{Pos}(\Pi)$ the set of positions of (predicates in) a program Π.

2.1 Derivation-Tree Bag (DTB) Semantics for Datalog and Datalog$^{\neg s}$

We follow [21], where tuples are *colored* to tell apart duplicates of a same element in the extensional database (EDB), via an infinite, ordered list $\mathcal{C}$ of colors $c_1, c_2, \ldots$. For a multiset M, $e \in M$ if $\mathit{mult}(e, M) > 0$ holds, where $\mathit{mult}(e, M)$ denotes the multiplicity of e in M. In this case, the n copies of e are denoted by $e{:}1, \ldots, e{:}n$, indicating that they are colored with $c_1, \ldots, c_n$, respectively. So, $\mathit{col}(e) := \{e{:}1, \ldots, e{:}n\}$ becomes a set. A multiset M_1 is (multi)contained in multiset M_2, when $\mathit{mult}(e, M_1) \le \mathit{mult}(e, M_2)$ for every $e \in M_1$. For a multiset M, $\mathit{col}(M) := \bigcup_{e \in M} \mathit{col}(e)$, which is a set. For a "colored" set S, $\mathit{col}^{-1}(S)$ produces a multiset by stripping tuples from their colors.

▶ **Example 1.** For $M = \{a, a, a, b, b, c\}$, $\mathit{col}(a) = \{a{:}1, a{:}2, a{:}3\}$, and $\mathit{col}(M) = S = \{a{:}1, a{:}2, a{:}3, b{:}1, b{:}2, c{:}1\}$. The inverse operation, the decoloration, gives, for instance: $\mathit{col}^{-1}(a{:}2) := a$; and $\mathit{col}^{-1}(S) := \{a, a, a, b, b, c\}$, a multiset.

We consider Datalog programs Π with multiset predicates and multiset EDBs D. A *derivation tree* (DT) for Π wrt. D is a tree with labeled nodes and edges, as follows:

1. For an EDB predicate P and $h \in \mathit{col}(P(D))$, a DT for $\mathit{col}^{-1}(h)$ contains a single node with label h.
2. For each rule of the form $\rho\colon\ H \leftarrow A_1, A_2, \ldots, A_k;$ with $k > 0$, and each tuple $\langle \mathcal{T}_1, \ldots, \mathcal{T}_k \rangle$ of DTs for the atoms $\langle d_1, \ldots, d_k \rangle$ that unify with $\langle A_1, A_2, \ldots, A_k \rangle$ with mgu θ, generate a DT for $H\theta$ with $H\theta$ as the root label, $\langle \mathcal{T}_1, \ldots, \mathcal{T}_k \rangle$ as the children, and ρ as the label for the edges from the root to the children. We assume that these children are arranged in the order of the corresponding body atoms in rule ρ.

For a DT $\mathcal{T}$, we define $\mathit{Atoms}(\mathcal{T})$ as col^{-1} of the root when $\mathcal{T}$ is a single-node tree, and the root-label of $\mathcal{T}$, otherwise. For a set of DTs $\mathfrak{T}$: $\mathit{Atoms}(\mathfrak{T}) := \biguplus\{\mathit{Atoms}(\mathcal{T}) \mid \mathcal{T} \in \mathfrak{T}\}$, which is a multiset containing D. Here, we write $\biguplus$ to denote the multi-union of (multi)sets that keeps all duplicates. If $\mathit{DT}(\Pi, \mathit{col}(D))$ is the set of (syntactically different) DTs, the *derivation-tree bag (DTB) semantics* for Π is the multiset $\mathit{DTBS}(\Pi, D) := \mathit{Atoms}(\mathit{DT}(\Pi, \mathit{col}(D)))$ (cf. [18] for an equivalent, provenance-based bag semantics for Datalog.)

▶ **Example 2.** Consider the program $\Pi = \{\rho_1, \rho_2, \rho_3\}$ and multiset EDB $D = \{Q(1, 2, 3), Q(1, 2, 5), Q(2, 3, 4), Q(2, 3, 4), T(4, 1, 2), T(4, 1, 2)\}$, where ρ_1, ρ_2, ρ_3 are defined as follows:

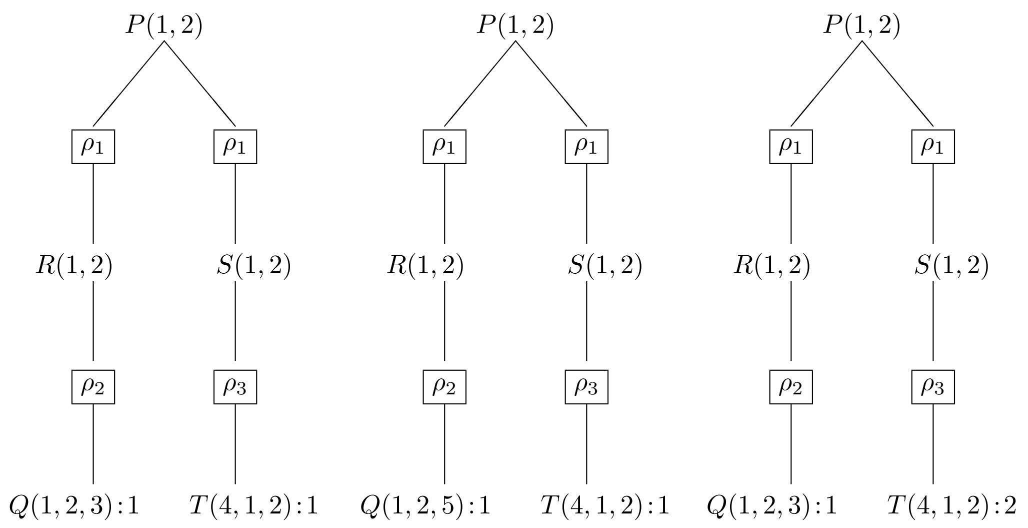

Figure 1 Three (out of four) derivation-trees for $P(1,2)$ in Example 2.

$$\rho_1\colon\ P(x,y) \leftarrow R(x,y), S(x,y); \qquad \rho_2\colon\ R(x,y) \leftarrow Q(x,y,z); \qquad \rho_3\colon\ S(x,y) \leftarrow T(z,x,y).$$

Here, $col(D) = \{Q(1,2,3){:}1, Q(1,2,5){:}1, Q(2,3,4){:}1, Q(2,3,4){:}2, T(4,1,2){:}1, T(4,1,2){:}2\}$. In total, we have 16 DTs: (a) 6 single-node trees with labels in $col(D)$ (b) 6 depth-two, linear trees (root to the left, i.e. rotated in $-90°$): $R(1,2) - \rho_2 - Q(1,2,3){:}1$. $R(1,2) - \rho_2 - Q(1,2,5){:}1$. $R(2,3) - \rho_2 - Q(2,3,4){:}1$. $R(2,3) - \rho_2 - Q(2,3,4){:}2$. $S(1,2) - \rho_3 - T(4,1,2){:}1$. $S(1,2) - \rho_3 - T(4,1,2){:}2$. (c) 4 depth-three trees for $P(1,2)$, three of which are displayed in Figure 1. The 16 different DTs in $DT(\Pi, col(D))$ give rise to $DTBS(\Pi, D) = D \cup \{R(1,2)$, $R(1,2)$, $R(2,3)$, $R(2,3)$, $S(1,2)$, $S(1,2)$, $P(1,2)$, $P(1,2)$, $P(1,2)$, $P(1,2)\}$.

In [22], a bag semantics for Datalog$^{\neg s}$ was introduced via derivation-trees (DTs), extending the DTB semantics in [21] for (positive) Datalog. This extension applies to Datalog programs with stratified negation that are range-restricted and safe, i.e. a variable in a rule head or in a negative literal must appear in a positive literal in the body of the same rule. If Π is a Datalog$^{\neg s}$ program with multiset predicates and multiset EDB D, a *derivation tree* for Π wrt. D is as for Datalog programs, but with condition **2.** modified as follows:

2'. Now let ρ be a rule of the form $\rho\colon\ H \leftarrow A_1, A_2, \dots, A_k, \neg B_1, \dots \neg B_\ell;$ with $k > 0$ and $\ell \geq 0$. Let the predicate of H be of some stratum i and let the predicates of $B_1, \dots, B_\ell$ be of some stratum $< i$. Assume that we have already computed all derivation trees for (atoms with) predicates up to stratum $i-1$. Then, for each tuple $\langle \mathcal{T}_1, \dots, \mathcal{T}_k \rangle$ of DTs for the atoms $\langle d_1, \dots, d_k \rangle$ that unify with $\langle A_1, A_2, \dots, A_k \rangle$ with mgu θ, such that there is no DT for any of the atoms $B_1\theta, \dots, B_\ell\theta$, generate a DT for $H\theta$ with $H\theta$ as the root label and $\langle \mathcal{T}_1, \dots, \mathcal{T}_k, \neg B_1\theta, \dots, \neg B_\ell\theta \rangle$ as the children, in this order. Furthermore, all edges from the root to its children are labelled with ρ.

Analogously to the positive case, now for a range-restricted and safe program Π in Datalog$^{\neg s}$ and multiset EDB D, we write $DTBS(\Pi, D)$ to denote the derivation-tree based bag semantics.

▶ **Example 3** (ex. 2 cont.). Consider now the EDB $D' = \{Q(1,2,3), Q(1,2,5), Q(2,3,4), Q(2,3,4), T(4,1,2)\}$ (with one duplicate of $T(4,1,2)$ removed from D), and modify Π to $\Pi' = \{\rho_1', \rho_2, \rho_3\}$ with $\rho_1'\colon\ \ P(x,y) \leftarrow R(x,y), \neg S(x,y)$ (i.e., ρ_1' now encodes multiset difference). Then, predicates Q, R, S, T are on stratum 0 and P is on stratum 1. The DTs for atoms with predicates from stratum 0 are as in Example 2 with two exceptions: there is now only one single-node DT for $T(4,1,2)$ and only one DT for $S(1,2)$.

For ground atoms with predicate P, we now only get two DTs producing $P(2,3)$ – derived via two copies of $R(2,3)$, which in turn stem from the two copies of $Q(2,3,4)$ in the EDB. In contrast, $P(1,2)$ cannot be derived from either of the copies of $R(1,2)$ obtained from either $Q(1,2,3)$ or $Q(1,2,5)$ in the EDB because $S(1,2)$ has a DT.

In total, we have 12 different DTs in $DT(\Pi', \mathit{col}(D'))$ with $DTBS(\Pi', D') = D' \cup \{R(1,2)$, $R(1,2)$, $R(2,3)$, $R(2,3)$, $S(1,2)$, $P(2,3)$, $P(2,3)\}$. Notice that the DT semantics interprets the difference in rule ρ_1' as "all or nothing": when computing $P(x,y)$, a single DT for $S(x,y)$ "kills" all the DTs for $R(x,y)$ (cf. Section 6). For example, $P(1,2)$ is not derived despite the fact that we have two copies of $R(1,2)$ and only one of $S(1,2)$.

2.2 Warded Datalog$^{\pm}$

Datalog$^{\pm}$ was introduced in [9] as an extension of Datalog, where the "+" stands for the new ingredients, namely: tuple-generating dependencies (*tgds*), written as *existential rules* of the form $\exists \bar{z} P(\bar{x}', \bar{z}) \;\leftarrow\; P_1(\bar{x}_1), \ldots, P_n(\bar{x}_n)$, with $\bar{x}' \subseteq \bigcup_i \bar{x}_i$, and $\bar{z} \cap \bigcup_i \bar{x}_i = \emptyset$; as well as equality-generating dependencies (*egds*) and negative constraints. In this work we ignore egds and constraints. The "$-$" in Datalog$^{\pm}$ stands for syntactic restrictions on rules to ensure decidability of CQ answering.

We consider three sets of term symbols: $\mathbf{C}$, $\mathbf{B}$, and $\mathbf{V}$ containing constants, labelled nulls (also known as blank nodes in the semantic web context), and variables, respectively. Let T denote an atom or a set of atoms. We write $\mathit{var}(T)$ and $\mathit{nulls}(T)$ to denote the set of variables and nulls, respectively, occurring in T. In a DB, typically an EDB D, all terms are from $\mathbf{C}$. In an *instance*, we also allow terms to be from $\mathbf{B}$. For a rule ρ, $\mathit{body}(\rho)$ denotes the set of atoms in its body, and $\mathit{head}(\rho)$, the atom in its head. A homomorphism h from a set X of atoms to a set X' of atoms is a partial function $h\colon \mathbf{C} \cup \mathbf{B} \cup \mathbf{V} \to \mathbf{C} \cup \mathbf{B} \cup \mathbf{V}$ such that $h(c) = c$ for all $c \in \mathbf{C}$ and $P(h(t_1), \ldots, h(t_k)) \in X'$ for every atom $P(t_1, \ldots, t_k) \in X$.

We say that a rule $\rho \in \Pi$ is *applicable* to an instance I if there exists a homomorphism h from $\mathit{body}(\rho)$ to I. In this case, the result of applying ρ to I is an instance $I' = I \cup \{h'(\mathit{head}(\rho))\}$, where h' coincides with h on $\mathit{var}(\mathit{body}(\rho))$ and h' maps each existential variable in the head of ρ to a fresh labelled null not occurring in I. For such an application of ρ to I, we write $I\langle \rho, h\rangle I'$. Such an application of ρ to I is called a *chase step*. The chase is an important tool in the presence of existential rules. A *chase sequence* for a DB D and a Datalog$^{\pm}$ program Π is a sequence of chase steps $I_i\langle \rho_i, h_i\rangle I_{i+1}$ with $i \geq 0$, such that $I_0 = D$ and $\rho_i \in \Pi$ for every i (also denoted $I_i \rightsquigarrow_{\rho_i, h_i} I_{i+1}$). For the sake of readability, we sometimes only write the newly generated atoms of each chase step without repeating the old atoms. Also the subscript ρ_i, h_i is omitted if it is clear from the context. A chase sequence then reads $I_0 \rightsquigarrow A_1 \rightsquigarrow A_2 \rightsquigarrow A_3 \rightsquigarrow \ldots$ with $I_{i+1} \setminus I_i = \{A_{i+1}\}$.

The final atoms of all possible chase sequences for DB D and Π form an instance referred to as $\mathit{Chase}(D, \Pi)$, which can be infinite. We denote the result of all chase sequences up to length d for some $d \geq 0$ as $\mathit{Chase}^d(D, \Pi)$. The chase variant assumed here is the so-called *oblivious chase* [8, 19], i.e., if a rule $\rho \in \Pi$ ever becomes applicable with some homomorphism h, then $\mathit{Chase}(D, \Pi)$ contains exactly one atom of the form $h'(\mathit{head}(\rho))$ such that h' extends h to the existential variables in the head of ρ. Intuitively, each rule is applied exactly once for every applicable homomorphism.

Consider a DB D and a Datalog$^{\pm}$ program Π (the former the EDB for the latter). As a logical theory, $\Pi \cup D$ may have multiple models, but the model $M = \mathit{Chase}(D, \Pi)$ turns out to be a correct representative for the class of models: for every BCQ $\mathcal{Q}$, $\Pi \cup D \models \mathcal{Q}$

iff $M \models Q$ [14]. There are classes of Datalog$^\pm$ that, even with an infinite chase, allow for decidable or even tractable CQA in the size of the EDB. Much effort has been made in identifying and characterizing interesting syntactic classes of programs with this property (see [8] for an overview). In this direction, *warded* Datalog$^\pm$ was introduced in [3, 4, 17], as a particularly well-behaved fragment of Datalog$^\pm$, for which CQA is tractable. We briefly recall and illustrate it here, for which we need some preliminary notions.

A position p in Datalog$^\pm$ program Π is *affected* if: (a) an existential variable appears in p, or (b) there is $\rho \in \Pi$ such that a variable x appears in p in $head(\rho)$ and all occurrences of x in $body(\rho)$ are in affected positions. $Aff(\Pi)$ and $NonAff(\Pi)$ denote the sets of affected, resp. non-affected, positions of Π. Intuitively, $Aff(\Pi)$ contains all positions where the chase may possibly introduce a null.

▶ **Example 4.** Consider the following program:

$$\exists z_1\ R(y_1, z_1) \leftarrow P(x_1, y_1) \tag{1}$$
$$\exists z_2\ P(x_2, z_2) \leftarrow S(u_2, x_2, x_2), R(u_2, y_2) \tag{2}$$
$$\exists z_3\ S(x_3, y_3, z_3) \leftarrow P(x_3, y_3), U(x_3). \tag{3}$$

By the first case, $R[2], P[2], S[3]$ are affected. By the second case, $R[1], S[2]$ are affected. Now that $S[2], S[3]$ are affected, also $P[1]$ is. We thus have $Aff(\Pi) = \{P[1], P[2], R[1], R[2], S[2], S[3]\}$ and $NonAff(\Pi) = \{S[1], U[1]\}$.

For a rule $\rho \in \Pi$, and a variable x: (a) $x \in body(\rho)$ is *harmless* if it appears at least once in $body(\rho)$ at a position in $NonAff(\Pi)$. $Harmless(\rho)$ denotes the set of harmless variables in ρ. Otherwise, a variable is called *harmful.* Intuitively, harmless variables will always be instantiated to constants in any chase step, whereas harmful variables may be instantiated to nulls. (b) $x \in body(\rho)$ is *dangerous* if $x \notin Harmless(\rho)$ and $x \in head(\rho)$. $Dang(\rho)$ denotes the set of dangerous variables in ρ. These are the variables which may propagate nulls into the atom created by a chase step.

▶ **Example 5** (ex. 4 cont.)**.** x_1 and y_1 are both harmful but only y_1 is dangerous for (1); u_2 is harmless, x_2 is dangerous, y_2 is harmful but not dangerous for (2); x_3 is harmless and y_3 is dangerous for (3).

Now, a rule $\rho \in \Pi$ is *warded* if either $Dang(\rho) = \emptyset$ or there exists an atom $A \in body(\rho)$, the ward, such that (1) $Dang(\rho) \subseteq var(A)$ and (2) $var(A) \cap var(body(\rho) \setminus \{A\}) \subseteq Harmless(\rho)$. A program Π is *warded* if every rule $\rho \in \Pi$ is warded.

▶ **Example 6** (ex. 5 cont.)**.** Rule (1) is trivially warded with the single body atom as the ward. Rule (2) is warded by $S(u_2, x_2, x_2)$: variable x_2 is the only dangerous variable and u_2 (the only variable shared by the ward with the rest of the body) is harmless. Actually, the other body atom $R(u_2, y_2)$ contains the harmful variable y_2; but it is not dangerous and not shared with the ward. Finally, rule (3) is warded by $P(x_3, y_3)$; the other atom $U(x_3)$ contains no affected variable. Since all rules are warded, the program Π is warded.

Datalog$^\pm$ can be extended with safe, stratified negation in the usual way, similarly as stratified Datalog [1]. The resulting Datalog$^{\pm,\neg s}$ can also be given a chase-based semantics [10]. The notions of affected/non-affected positions and harmless/harmful/dangerous variables carry over to a Datalog$^{\pm,\neg s}$ program Π by considering only the program Π^{pos} obtained from Π by deleting all negated body atoms. For warded Datalog$^\pm$, only a restricted form of stratified negation is allowed – so-called *stratified ground* negation. This means that we require for every rule $\rho \in \Pi$: if ρ contains a negated atom $P(t_1,\ldots,t_n)$, then every t_i must be either a constant (i.e, $t_i \in \mathbf{C}$) or a harmless variable. Hence, negated atoms can never contain a null in the chase. We write Datalog$^{\pm,\neg sg}$ for programs in this language.

The class of warded Datalog$^{\pm,\neg sg}$ programs extends the class of Datalog$^{\neg s}$ programs. Warded Datalog$^{\pm,\neg sg}$ is expressive enough to capture query answering of a core fragment of SPARQL under certain OWL 2 QL entailment regimes [16], and this task can actually be accomplished in polynomial time (data complexity) [3, 4]. Hence, Datalog$^{\pm,\neg sg}$ constitutes a very good compromise between expressive power and complexity. Recently, a powerful query engine for warded Datalog$^{\pm,\neg sg}$ has been presented [6, 7], namely the VADALOG system.

3 Datalog$^{\pm}$-Based Bag Semantics for Datalog$^{\neg s}$

We now provide a set-semantics that represents the bag semantics for a Datalog program Π with a multiset EDB D via the transformation into a Datalog$^{\pm}$ program Π^+ over a set EDB D^+ obtained from D. For this, we assume w.l.o.g., that the set of *nulls*, $\mathbf{B}$, for a Datalog$^{\pm}$ program is partitioned into two infinite ordered sets $\mathcal{I} = \{\iota_1, \iota_2, \ldots\}$, for unique, global tuple identifiers (tids), and $\mathcal{N} = \{\eta_1, \eta_2, \ldots\}$, for usual nulls in Datalog$^{\pm}$ programs. Given a multiset EDB D and a program Π, instead of using colors and syntactically different derivation trees, we will use elements of $\mathcal{I}$ to identify both the elements of the EDB and the tuples resulting from applications of the chase.

For every predicate $P(\ldots)$, we introduce a new version $P(.\,;\ldots)$ with an extra, first argument (its 0-th position) to accommodate a tid, which is a null from $\mathcal{I}$. If an atom $P(\bar{a})$ appears in D as n duplicates, we create the tuples $P(\iota_1';\bar{a}),\ldots,P(\iota_n';\bar{a})$, with the ι_i' pairwise different nulls from $\mathcal{I}$ as tids, and not used to identify any other element of D. We obtain a *set* EDB D^+ from the multiset EDB D. Given a rule in Π, we introduce tid-variables (i.e. appearing in the 0-th positions of predicates) and existential quantifiers in the rule head, to formally generate fresh tids when the rule applies. More precisely, a rule in Π of the form $\rho\colon\ H(\bar{x}) \ \leftarrow\ A_1(\bar{x}_1), A_2(\bar{x}_2), \ldots, A_k(\bar{x}_k)$, with $k > 0$, $\bar{x} \subseteq \cup_i \bar{x}_i$, becomes the Datalog$^{\pm}$ rule $\rho^+\colon\ \exists z\, H(z;\bar{x}) \leftarrow A_1(z_1;\bar{x}_1), A_2(z_2;\bar{x}_2), \ldots, A_k(z_k;\bar{x}_k)$, with fresh, different variables $z, z_1, \ldots, z_k$. The resulting Datalog$^{\pm}$ program Π^+ can be evaluated according to the usual *set semantics* on the set EDB D^+ via the chase: when the instantiated body $A_1(\iota_1';\bar{a}_1)$, $A_2(\iota_2';\bar{a}_2), \ldots, A_k(\iota_k';\bar{a}_k)$ of rule ρ^+ becomes true, then the new tuple $H(\iota;\bar{a})$ is created, with ι the first (new) null from $\mathcal{I}$ that has not been used yet, i.e., the tid of the new atom.

▶ **Example 7** (ex. 2 cont.). The EDB D from Example 2 becomes

$$D^+ = \{Q(\iota_1;1,2,3),\ Q(\iota_2;1,2,5), Q(\iota_3;2,3,4), Q(\iota_4;2,3,4), T(\iota_5;4,1,2), T(\iota_6;4,1,2)\};^1$$

and program Π becomes $\Pi^+ = \{\rho_1^+, \rho_2^+, \rho_3^+\}$ with $\rho_1^+\colon\ \exists u P(u;x,y) \leftarrow R(u_1;x,y), S(u_2;x,y)$; $\rho_2^+\colon\ \exists u R(u;x,y) \leftarrow Q(u_1;x,y,z)$; $\rho_3^+\colon\ \exists u S(u;x,y) \leftarrow T(u_1;z,x,y)$.

The following is a 3-step chase sequence of D^+ and Π^+: $\{Q(\iota_1;1,2,3), T(\iota_4;4,1,2)\} \leadsto_{\rho_2^+} R(\iota_7;1,2) \leadsto_{\rho_3^+} S(\iota_{11};1,2) \leadsto_{\rho_1^+} P(\iota_{13};1,2)$.

Analogously to the depth-two and depth-three trees in Example 2, the chase produces 10 new atoms. In total, we get: $\mathit{Chase}(\Pi^+, D^+) = D^+ \cup \{R(\iota_7;1,2),\ R(\iota_8;1,2),\ R(\iota_9;2,3),$ $R(\iota_{10};2,3),\ S(\iota_{11};1,2),\ S(\iota_{12};1,2),\ P(\iota_{13};1,2),\ P(\iota_{14};1,2),\ P(\iota_{15};1,2),\ P(\iota_{16};1,2)\}$.

[1] Notice that this set version of D can also be created by means of Datalog$^{\pm}$ rules. For example, with the rule $\exists z Q(z;x,y,v) \leftarrow Q(x,y,v)$ for the EDB predicate Q.

In order to extend the PBB Semantics to Datalog$^{\neg s}$, we have to extend our transformation of programs Π into Π^+ to rules with negated atoms. Consider a rule ρ of the form:

$$\rho\colon\ H(\bar{x}) \ \leftarrow\ A_1(\bar{x}_1), \ldots, A_k(\bar{x}_k), \neg B_1(\bar{x}_{k+1}), \ldots, \neg B_\ell(\bar{x}_{k+\ell}), \tag{4}$$

with, $\bar{x}_{k+1}, \ldots, \bar{x}_{k+\ell} \subseteq \bigcup_{i=1}^{k} \bar{x}_i$; we transform it into the following two rules:

$$\begin{aligned} \rho^+\colon\ \exists z H(z;\bar{x}) \quad &\leftarrow \quad A_1(z_1;\bar{x}_1), \ldots, A_k(z_k;\bar{x}_k), \neg Aux_1^\rho(\bar{x}_{k+1}), \ldots, \neg Aux_\ell^\rho(\bar{x}_{k+\ell}), \\ Aux_i^\rho(\bar{x}_{k+i}) \quad &\leftarrow \quad B_i(z_i;\bar{x}_{k+i}),\ \ i = 1, \ldots, \ell. \end{aligned} \tag{5}$$

The introduction of auxiliary predicates Aux_i is crucial since adding fresh variables directly to the negated atoms would yield negated atoms of the form $\neg B_i(z_{k+i};\bar{x}_{k+i})$ in the rule body, which make the rule unsafe. The resulting Datalog$^{\pm,\neg s}$ program is from the desired class Datalog$^{\pm,\neg sg}$:

▶ **Theorem 8.** *Let Π be a Datalog$^{\neg s}$ program and let Π^+ be the transformation of Π into a Datalog$^{\pm,\neg s}$ program. Then, Π^+ is a warded Datalog$^{\pm,\neg sg}$ program.*

Operation col^{-1} of Section 2 inspires de-identification and multiset merging operations. Sometimes we use double braces, $\{\{\ldots\}\}$, to emphasize that the operation produces a *multiset*.

▶ **Definition 9.** *For a set D of tuples with tids, $\mathcal{DI}(D)$ and $\mathcal{SP}(D)$, for de-identification and set-projection, respectively, are: (a) $\mathcal{DI}(D) := \{\{P(\bar{c}) \mid P(t;\bar{c}) \in D \text{ for some } t\}\}$, a multiset; and (b) $\mathcal{SP}(D) := \{P(\bar{c}) \mid P(t;\bar{c}) \in D \ \text{ for some } t\}$, a set.*

▶ **Definition 10.** *Given a Datalog$^{\neg s}$ program Π and a multiset EDB D, the* program-based bag semantics *(PBB semantics) assigns to $\Pi \cup D$ the multiset:*

$$PBBS(\Pi, D) := \mathcal{DI}(Chase(\Pi^+, D^+)) = \{\{P(\bar{a}) \mid P(t;\bar{a}) \in Chase(\Pi^+, D^+)\}\}.$$

The main results in this section are the correspondence of PBB semantics and DTB semantics and the relationship of both with classical set semantics of Datalog:

▶ **Theorem 11.** *For a Datalog$^{\neg s}$ program Π with a multiset EDB D, $DTBS(\Pi, D) = PBBS(\Pi, D)$ holds.*

Proof Idea. The theorem is proved by establishing a one-to-one correspondence between DTs in $DT(\Pi, col(D))$ with a fixed root atom $P(\bar{c})$ and (minimal) chase-derivations of $P(\bar{c})$ from D^+ via Π^+. This proof proceeds by induction on the depth of the DTs and length of the chase sequences. ◀

▶ **Corollary 12.** *Given a Datalog (resp. Datalog$^{\neg s}$) program Π and a multiset EDB D, the set $\mathcal{SP}(Chase(\Pi^+, D^+))$ is the minimal model (resp. the standard model) of the program $\Pi \cup \mathcal{SP}(D)$.*

4 Bag Semantics for Datalog$^{\pm,\neg sg}$

In the previous section, we have seen that warded Datalog$^{\pm}$ (possibly extended with stratified ground negation) is well suited to capture the bag semantics of Datalog in terms of classical set semantics. We now want to go one step further and study the *bag semantics* for Datalog$^{\pm,\neg sg}$ itself. Note that this question makes a lot of sense given the results from [4], where it is shown that warded Datalog$^{\pm,\neg sg}$ captures a core fragment of SPARQL under certain OWL2 QL entailment regimes and the official W3C semantics of SPARQL is a bag semantics.

4.1 Extension of the DTB Semantics to Datalog$^{\pm,\neg sg}$

The definition of a DT-based bag semantics for Datalog$^{\pm,\neg sg}$ is not as straightforward as for Datalog$^{\neg s}$, since atoms in a model of $D \cup \Pi$ for a (multiset) EDB D and Datalog$^{\pm,\neg sg}$ program Π may have labelled nulls as arguments, which correspond to existentially quantified variables and may be arbitrarily chosen. Hence, when counting duplicates, it is not clear whether two atoms differing only in their choice of nulls should be treated as copies of each other or not. We therefore treat multiplicities of atoms analogously to multiple answers to single-atom queries, i.e., the multiplicity of an atom $P(t_1, \ldots, t_k)$ wrt. EDB D and program Π corresponds to the multiplicity of answer $(t_1, \ldots, t_k)$ to the query $Q = P(x_1, \ldots, x_k)$ over the database D and program Π. In other words, we ask for all instantiations $(t_1, \ldots, t_k)$ of $(x_1, \ldots, x_k)$ such that $P(t_1, \ldots, t_k)$ is true in *every* model of $D \cup \Pi$. It is well known that only ground instantiations (on $\mathbf{C}$) can have this property (see e.g. [14]). Hence, below, we restrict ourselves to considering duplicates of ground atoms containing constants from $\mathbf{C}$ only (in this section we are not using tid-positions 0). In the rest of this section, unless otherwise stated, "ground atom" means instantiated on $\mathbf{C}$; and programs belong to Datalog$^{\pm,\neg sg}$.

In order to define the multiplicity of a ground atom $P(t_1, \ldots, t_k)$ wrt. a (multiset) EDB D and a warded Datalog$^{\pm,\neg sg}$ program Π, we adopt the notion of *proof tree* used in [4, 11], which generalizes the notion of derivation tree to Datalog$^{\pm,\neg sg}$. We consider first positive Datalog$^{\pm}$ programs. A proof tree (PT) for an atom A (possibly with nulls) wrt. (a set) EDB D and Datalog$^{\pm}$ program Π is a node- and edge-labelled tree with labelling function λ, such that: (1) The nodes are labelled by atoms over $\mathbf{C} \cup \mathbf{B}$. (2) The edges are labelled by rules from Π. (3) The root is labelled by A. (4) The leaf nodes are labelled by atoms from D. (5) The edges from a node N to its child nodes $N_1, \ldots, N_k$ are all labelled with the same rule ρ. (6) The atom labelling N corresponds to the result of a chase step where $\mathit{body}(\rho)$ is instantiated to $\{\lambda(N_1), \ldots, \lambda(N_k)\}$ and $\mathit{head}(\rho)$ becomes $\lambda(N)$ when instantiating the existential variables of $\mathit{head}(\rho)$ with fresh nulls. (7) If M' (resp. N') is the parent node of M (resp. N) such that $\mathit{nulls}(\lambda(M')) \setminus \mathit{nulls}(\lambda(M))$ and $\mathit{nulls}(\lambda(N')) \setminus \mathit{nulls}(\lambda(N))$ share at least one null, then the entire subtrees rooted at M' and at N' must be isomorphic (i.e., the same tree structure and the same labels). (8) If, for two nodes M and N, $\lambda(M)$ and $\lambda(N)$ share a null v, then there exist ancestors M' of M and N' of N such that M' and N' are siblings corresponding to two body atoms A and B of rule ρ with $x \in \mathit{var}(A)$ and $x \in \mathit{var}(B)$ for some variable x and ρ is applied with some substitution γ which sets $\gamma(x) = v$; moreover, v occurs in the labels of the entire paths from M' to M and from N' to N. A proof tree for Example 13 below is shown in Figure 2, left. As with derivation trees, we assume that siblings in the proof tree are arranged in the order of the corresponding body atoms in the rule ρ labelling the edge to the parent node (cf. Section 2.1).

Intuitively, a PT is a tree-representation of the derivation of an atom by the chase. The parent/child relationship in a PT corresponds to the head/body of a rule application in the chase. Condition (7) above refers to the case that a non-ground atom is used in two different rule applications within the chase sequence. In this case, the two occurrences of this atom must have identical proof sub-trees. A PT can be obtained from a chase-derivation by *reversing the edges and unfolding the chase graph into a tree by copying some of the nodes* [4]. By definition of the chase in Section 2.2, it can never happen that the same null is created by two different chase steps. Note that the nulls in $\mathit{nulls}(\lambda(M')) \setminus \mathit{nulls}(\lambda(M))$ (and, likewise in $\mathit{nulls}(\lambda(N')) \setminus \mathit{nulls}((\lambda(N)))$ are precisely the newly created ones. Hence, if $\lambda(M')$ and $\lambda(N')$ share such a null, then $\lambda(M')$ and $\lambda(N')$ are the same atom and the subtrees rooted at these nodes are obtained by unfolding the same subgraph of the chase graph. Condition (8) makes sure that we use the same null v in a PT only if this is required by a join condition of some

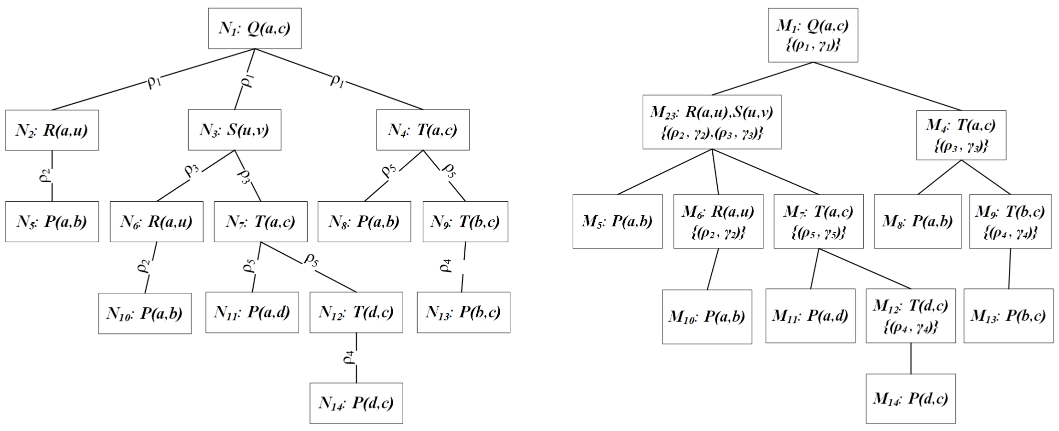

Figure 2 Proof tree (left) and witness (right) for atom $Q(a,c)$ in Examples 13 and 20.

rule ρ; otherwise nulls are renamed apart.

▶ **Example 13.** Let $\Pi = \{\rho_1, \ldots, \rho_5\}$ be the Datalog$^\pm$ program with $\rho_1\colon\ Q(x,w) \leftarrow R(x,y), S(y,z), T(x,w)$; $\rho_2:\ \exists z\ R(x,z) \leftarrow P(x,y)$; $\rho_3:\ \exists z\ S(x,z) \leftarrow R(w,x), T(w,y)$; $\rho_4: T(x,y) \leftarrow P(x,y)$; $\rho_5\colon\ T(x,y) \leftarrow P(x,z), T(z,y)$. This program belongs to Datalog$^{\pm,\neg sg}$, and – although not necessary to build a proof tree for it – we notice that it is also warded: $\mathit{Aff}(\Pi) = \{R[2], S[2], S[1]\}$; and all other positions are not affected. Rule ρ_3 is warded with ward $R(w,x)$ (where x is the only dangerous variable in this rule). All other rules are trivially warded because they have no dangerous variables.

Now let $D = \{P(a,b), P(b,c), P(a,d), P(d,c)\}$. A possible proof tree for $Q(a,c)$ is shown in Figure 2 on the left. It is important to note that nodes N_2 and N_6 introducing labelled null u are labelled with the same atom and give rise to identical subtrees. Of course, $R(a,u)$ could also result from a chase step applying ρ_2 to $P(a,d)$. However, this would generate a null different from u and, subsequently, the nulls in $R(a,\_)$ and $S(\_,v)$ (in N_2 and N_3) would be different, and rule ρ_1 could not be applied anymore. In contrast, the nodes N_4 and N_7 with label $T(a,c)$ span different subtrees. This is desired: there are two possible derivations for each occurrence of atom $T(a,c)$ in the PT.

Here we deviate slightly from the definition of PTs in [4], in that we allow the same *ground* atom to have different derivations. This is needed to detect duplicates and to make sure that PTs in fact constitute a generalization of the derivation trees in Section 2.1. Moreover, condition (8) is needed to avoid non-isomorphic PTs by "unforced" repetition of nulls (i.e., identical nulls that are not required by a join condition further up in the tree). Analogous to the generalization in Section 2.1 of DTs for Datalog to DTs for Datalog$^{\neg s}$, it is easy to generalize proof trees to Datalog$^{\pm,\neg sg}$. Here it is important that we only allow stratified *ground* negation. Hence, analogously to DTs for Datalog$^{\neg s}$, we allow negated ground atoms $\neg A$ to occur as node labels of leaf nodes in a PT, provided that the positive atom A has no PT. Moreover, it is no problem to allow also multiset EDBs since, as in Section 2.1, we can keep duplicates apart by means of a coloring function *col*.

Finally, we can define proof trees $\mathcal{T}$ and $\mathcal{T}'$ as equivalent, denoted $\mathcal{T} \equiv \mathcal{T}'$, if one is obtained from the other by renaming of nulls. We can thus normalize PTs by assuming that nulls in node labels are from some fixed set $\{u_1, u_2 \ldots\}$ and that these nulls are introduced in the labels of the PT by some fixed-order traversal (e.g., top-down, left-to-right).

For a PT $\mathcal{T}$, we define $\mathit{Atoms}(\mathcal{T})$ as col^{-1} of the root when $\mathcal{T}$ is a single-node tree, and the root-label of $\mathcal{T}$, otherwise. For a set of PTs $\mathfrak{T}$: $\mathit{Atoms}(\mathfrak{T}) := \biguplus\{\mathit{Atoms}(\mathcal{T}) \mid \mathcal{T} \in \mathfrak{T}\}$, which is a multiset that multi-contains D. If $\mathit{PT}(\Pi,\ \mathit{col}(D))$ is the set of normalized, *non-equivalent* PTs, the *proof-tree bag (PTB) semantics* for Π is the multiset $\mathit{PTBS}(\Pi, D) := \mathit{Atoms}(\mathit{PT}(\Pi, \mathit{col}(D)))$. For a ground atom A, $\mathit{mult}(A, \mathit{PTBS}(\Pi, D))$ denotes the multiplicity of A in the multiset $\mathit{PTBS}(\Pi, D)$.

▶ **Example 14** (ex. 13 cont.). To compute $\mathit{PTBS}(\Pi, D)$ for Π and D from Example 13, we have to determine all proof trees of all ground atoms derivable from Π and D. In Figure 2, we have already seen one proof tree for $Q(a, c)$; and we have observed that the sub-proof tree of $R(a, u)$ rooted at nodes N_2 and N_6 could be replaced by a child node with label $P(a, d)$ (either both subtrees or none has to be changed). In total, the ground atom $Q(a, c)$ has 8 different proof trees wrt. Π and D (multiplying the 2 possible derivations of atom $R(a, u)$ with 4 derivations for the two occurrences of the atom $T(a, c)$ in nodes N_4 and N_7). The other ground atoms derivable from Π and D are $T(a, c)$ (with 2 possible PTs as discussed in Example 13) and the atoms $T(i, j)$ for each $P(i, j)$ in D. Hence, we have $\mathit{PTBS}(\Pi, D) = D \cup \{T(a, b), T(b, c), T(a, d), T(d, a), T(a, c), T(a, c), Q(a, c), \ldots, Q(a, c)\}$.

Clearly, every Datalog$^{\neg s}$ program is a special case of a warded Datalog$^{\pm,\neg sg}$ program. It is easy to verify that the PTB semantics indeed generalizes the DTB semantics:

▶ **Proposition 15.** Let Π be a Datalog$^{\neg s}$ program and D an EDB (possibly with duplicates). Then $\mathit{DTBS}(\Pi, D) = \mathit{PTBS}(\Pi, D)$ holds.

4.2 Extension of the PBB Semantics to Datalog$^{\pm,\neg sg}$

We now extend also the PBB semantics to Datalog$^{\pm,\neg sg}$ programs Π. First, in all predicates of Π, we add position 0 to carry tid-variables. Then every rule $\rho \in \Pi$ of the form $\rho:\ \exists\bar{y}H(\bar{y}, \bar{x}) \leftarrow A_1(\bar{x}_1), A_2(\bar{x}_2), \ldots, A_k(\bar{x}_k)$, with $k > 0$, $\bar{x} \subseteq \cup_i \bar{x}_i$, becomes the rule $\rho^+:\ \exists z\exists\bar{y}\ H(z; \bar{y}, \bar{x}) \leftarrow A_1(z_1; \bar{x}_1), A_2(z_2; \bar{x}_2), \ldots, A_k(z_k; \bar{x}_k)$, with fresh, different variables $z, z_1, \ldots, z_k$. Now consider a rule $\rho \in \Pi$ of the form $\rho:\ \exists\bar{y}H(\bar{y}\bar{x}) \leftarrow A_1(\bar{x}_1), \ldots, A_k(\bar{x}_k), \neg B_1(\bar{x}_{k+1}), \ldots, \neg B_\ell(\bar{x}_{k+\ell})$, with $\bar{x}_{k+1}, \ldots, \bar{x}_{k+\ell} \subseteq \bigcup_{i=1}^{k} \bar{x}_i$; we transform it into the following two rules:

$$\begin{aligned} \rho':\ \exists z\exists\bar{y}H(z; \bar{y}, \bar{x}) &\quad\leftarrow\quad A_1(z_1; \bar{x}_1), \ldots, A_k(z_k; \bar{x}_k), \neg \mathit{Aux}_1^{\rho}(\bar{x}_{k+1}), \ldots, \neg \mathit{Aux}_\ell^{\rho}(\bar{x}_{k+\ell}), \\ \mathit{Aux}_i^{\rho}(\bar{x}_{k+i}) &\quad\leftarrow\quad B_i(z_i; \bar{x}_{k+i}),\ \ i = 1, \ldots, \ell. \end{aligned} \tag{6}$$

Analogously to Theorem 8, the resulting program also belongs to Datalog$^{\pm,\neg sg}$. Finally, as in Section 3, the ground atoms in the multiset EDB D are extended in D^+ by nulls from $\mathcal{I} = \{\iota_1, \iota_2, \ldots\}$ as tids in position 0 to keep apart duplicates. For an instance I, we write $I_\downarrow$ to denote the set of atoms in I such that all positions except for position 0 (the tid) carry a ground term (from $\mathbf{C}$). Analogously to Definition 10, we then define the *program-based bag semantics* (PBB semantics) of a Datalog$^{\pm,\neg sg}$ program Π and multiset EDB D as $\mathit{PBBS}(\Pi, D) := \mathcal{DI}(\mathit{Chase}(\Pi^+, D^+)_\downarrow) = \{\!\{P(\bar{a}) \mid P(\bar{a}) \text{ is ground and } P(t; \bar{a}) \in \mathit{Chase}(\Pi^+, D^+)\}\!\}$. For a ground atom A (here, without nulls or tids), $\mathit{mult}(A, \mathit{PBBS}(\Pi, D))$ denotes the multiplicity of A in the multiset $\mathit{PBBS}(\Pi, D)$.

▶ **Example 16** (ex. 13 and 14 cont.). The transformations of Datalog$^{\pm,\neg sg}$ program Π and multiset EDB D from Example 13 are $D^+ = \{P(\iota_1, a, b), P(\iota_2, b, c), P(\iota_3, a, d), P(\iota_4, d, c)\}$ and $\Pi^+ = \{\rho_1^+, \rho_2^+, \rho_3^+, \rho_4^+, \rho_5^+\}$ with: $\rho_1^+:\ (\exists u)Q(u, x, w) \leftarrow R(v_1, r, y), S(v_2, y, z), T(v_3, x, w)$; $\rho_2^+:\ (\exists u, z)R(u, x, z) \leftarrow P(v, x, y)$; $\rho_3^+:\ (\exists u, z)S(u, x, z) \leftarrow R(v_1, w, x), T(v_2, w, y)$; $\rho_4^+:\ (\exists u)T(u, x, y) \leftarrow P(v, x, y)$; $\rho_5^+:\ (\exists u)T(u, x, y) \leftarrow P(v_1, x, z),\ T(v_2, z, y)$.

Applying program Π^+ to instance D^+ gives, for example, the following chase sequence for deriving the atoms $T(\iota_5, d, c), T(\iota_7, a, c), T(\iota_8, b, c), Q(\iota_{11}, a, c)$, which correspond to the ground atoms in the proof tree in Figure 2, left: $D^+ \rightsquigarrow_{\rho_4^+} T(\iota_5, d, c) \rightsquigarrow_{\rho_2^+} R(\iota_6, a, u) \rightsquigarrow_{\rho_5^+} T(\iota_7, a, c) \rightsquigarrow_{\rho_4^+} T(\iota_8, b, c) \rightsquigarrow_{\rho_3^+} S(\iota_9, u, v) \rightsquigarrow_{\rho_5^+} T(\iota_{10}, a, c) \rightsquigarrow_{\rho_1^+} Q(\iota_{11}, a, c)$. Implicitly, we thus also turn the PTs rooted at N_4, N_7, and N_9 into chase sequences. In total, we get in $PBBS(\Pi, D)$ precisely the atoms and multiplicities as in $PTBS(\Pi, D)$ for Example 14.

In principle, the above transformation Π^+ and the PBB semantics are applicable to any program Π in Datalog$^{\pm,\neg sg}$. However, in the first place, we are interested in *warded* programs Π. It is easy to verify that wardedness of Π carries over to Π^+:

▶ **Proposition 17.** If Π is a warded Datalog$^{\pm,\neg sg}$ program, then so is Π^+.

Proof Idea. The key observation is that the only additional affected positions in Π^+ are the tids at position 0. However, the variables at these positions occur only once in each rule and are never propagated from the body to the head. Hence, they do not destroy wardedness. ◀

We conclude this section with the analogous result of Theorem 11:

▶ **Theorem 18.** *For ground atoms A, $mult(A, PTBS(\Pi, D)) = mult(A, PBBS(\Pi, D))$. Hence, for a warded Datalog$^{\pm,\neg sg}$ program Π and multiset EDB D, $PTBS(\Pi, D) = PBBS(\Pi, D)$.*

5 Decidability and Complexity of Multiplicity

For Datalog$^{\neg s}$ programs, the following problems related to duplicates have been investigated:

- FFE: Given a program Π, a database D, and a predicate P, decide if every derivable ground P-atom has a finite number of DTs.
- FEE: Given a program Π and a predicate P, decide if every derivable ground P-atom has a finite number of DTs *for every* database D.

It has been shown in [22] that FFE is decidable (even in PTIME data complexity), whereas FEE is undecidable. We extend this study by considering warded Datalog$^{\pm,\neg sg}$ instead of Datalog$^{\neg s}$, and by computing the concrete multiplicities in case of finiteness. We thus study the following problems:

- FINITENESS: For fixed warded Datalog$^{\pm,\neg sg}$ program Π: Given a multiset database D and a ground atom A, does A have finite multiplicity, i.e., is $mult(A, PBBS(\Pi, D))$ finite?
- MULTIPLICITY: For fixed warded Datalog$^{\pm,\neg sg}$ program Π: Given a multiset database D and a ground atom A, compute the multiplicity of A, i.e. $mult(A, PBBS(\Pi, D))$.

We will show that both problems defined above can be solved in polynomial time (data complexity). In case of the FINITENESS problem, we thus generalize the result of [22] for the FFE problem from Datalog$^{\neg s}$ to Datalog$^{\pm,\neg sg}$. In case of the MULTIPLICITY problem, no analogous result for Datalog or Datalog$^{\neg s}$ has existed before. The following example illustrates that, even if Π is a Datalog program with a single rule, we may have exponential multiplicities. Hence, simply computing all DTs or (in case of Datalog$^{\pm}$) all PTs is not a viable option if we aim at polynomial time complexity.

▶ **Example 19.** Let $D = \{E(a_0, a_1), E_(a_1, a_2), \ldots, E(a_{n-1}, a_n)\} \cup \{P(a_0, a_1), C(b_0), C(b_1)\}$ for $n \geq 1$ and let $\Pi = \{\rho\}$ with $\rho \colon P(x, y) \leftarrow P(x, z), E(z, y), C(w)$. Intuitively, E can be considered as an edge relation and P is the corresponding path relation for paths starting at a_0. The C-atom in the rule body (together with the two C-atoms in D) has the effect that there are always 2 possible derivations to extend a path. It can be verified by induction over

i, that atom $P(a_0, a_i)$ has 2^{i-1} possible derivations from D via Π. In particular, $P(a_0, a_n)$ has multiplicity 2^{n-1}.

Note that, if we add atom $E(a_1, a_1)$ to D (i.e., a self-loop, so to speak), then every atom $P(a_0, a_i)$ with $i \geq 1$ has infinite multiplicity. Intuitively, the infinitely many different derivation trees correspond to the arbitrary number of cycles through the self-loop $E(a_1, a_1)$ for a path from a_0 to a_i.

Our PTIME-membership results will be obtained by appropriately adapting the tractability proof of CQA for warded Datalog$^\pm$ in [4], which is based on the algorithm ProofTree for deciding if $D \cup \Pi \models P(\bar{c})$ holds for database D, warded Datalog$^\pm$ program Π, and ground atom $P(\bar{c})$. That algorithm works in ALOGSPACE (data complexity), i.e., alternating logspace, which coincides with PTIME [12]. It assumes Π to be normalized in such a way that each rule in Π is either *head-grounded* (i.e., each term in the head is a constant or a harmless variable) or *semi-body-grounded* (i.e., there exists at most one body atom with harmful variables). Algorithm ProofTree starts with ground atom $P(\bar{c})$ and applies resolution steps until the database D is reached. It thus proceeds as follows:

- If $P(\bar{c}) \in D$, then accept. Otherwise, guess a head-grounded rule $\rho \in \Pi$ whose head can be matched to $P(\bar{c})$ (denoted as $\rho \rhd P(\bar{c})$). Guess an instantiation γ on the variables in the body of ρ so that $\gamma(\mathit{head}(\rho)) = P(\bar{c})$. Let $\mathcal{S} = \gamma(\mathit{body}(\rho))$.
- Partition $\mathcal{S}$ into $\{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$, such that each null occurring in $\mathcal{S}$ occurs in exactly one $\mathcal{S}_i$, and each set $\mathcal{S}_i$ is chosen subset-minimal with this property. The purpose of these sets $\mathcal{S}_i$ of atoms is to keep together, in the parallel universal computations of ProofTree, the nulls in $\mathcal{S}$ until the atom in which they are created is known.
- Universally select each set $\mathcal{S}' \in \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$ and "prove" it: If $\mathcal{S}'$ consists of a single ground atom $P'(\bar{c}')$, then call ProofTree recursively for D, Π, $P'(\bar{c}')$. Otherwise, do the following:
  (1) For each atom $A \in \mathcal{S}'$, guess rule ρ_A with $\rho_A \rhd A$ and guess variable instantiation γ_A on the variables in the body of ρ_A such that $A = \gamma_A(\mathit{head}(\rho_A))$.
  (2) The set $\bigcup_{A \in \mathcal{S}'} \gamma_A(\mathit{body}(\rho_A))$ is partitioned as above and each component of this partition is proved in a parallel universal computation.

The key to the ALOGSPACE complexity of algorithm ProofTree is that the data structure propagated by this algorithm fits into logarithmic space. This data structure is given by a pair $(\mathcal{S}, \mathcal{R}_\mathcal{S})$, where $\mathcal{S}$ is a set of atoms (such that $|\mathcal{S}|$ is bounded by the maximum number of body atoms of the rules in Π) and $\mathcal{R}_\mathcal{S}$ is a set of pairs (z, x), where z is a null occurring in $\mathcal{S}$ and x is either an atom A (meaning that null z was created when the application of some rule ρ generated the atom A containing this null z) or the symbol ε (meaning that we have not yet found such an atom A). A *witness* for a successful computation of ProofTree is then given by a tree with existential and universal nodes, with an existential node N consisting of a pair $(\mathcal{S}, \mathcal{R}_\mathcal{S})$; a universal node N indicates the guessed rule ρ_A together with the instantiation γ_A for each atom $A \in \mathcal{S}$ at the (existential) parent node of N. The child nodes of each universal node N are obtained by partitioning $\bigcup_{A \in \mathcal{S}} \gamma_A(\mathit{body}(A))$ as described above and computing the corresponding set $\mathcal{R}_\mathcal{S}$. At the root, we thus have an existential node labelled $(\mathcal{S}, \mathcal{R}_\mathcal{S}) = (\{P(\bar{c})\}, \emptyset)$. Each leaf node is a universal node labelled with $(B, \emptyset)$ for some ground atom $B \in D$. Hence, such a node corresponds to an accept-state.

▶ **Example 20.** Recall Π and D from Example 13. A proof tree of ground atom $Q(a, c)$ is shown in Figure 2 on the left. On the right, we display the witness of the corresponding successful computation of ProofTree. Note that witnesses have a strict alternation of existential and universal nodes. In the witness in Figure 2, we have merged each existential node and its unique universal child node to make the correspondence between a PT and a witness yet more visible. In particular, on each depth level of the trees, we have exactly the same set of atoms (with the only difference, that in the witness these atoms are grouped together to sets $\mathcal{S}$ by null-occurrences). In this simple example, only node M_{23} contains such a group of atoms, namely the atoms from the nodes N_2 and N_3 in the PT.

In order not to overburden the figure, we have left out the sets $\mathcal{R}_\mathcal{S}$ carrying the information on the introduction of nulls. In this simple example, the only node with a non-empty set $\mathcal{R}_\mathcal{S}$ is M_6, with $\mathcal{R}_\mathcal{S} = \{(u, R(a, u)\}$, i.e. we have to pass on the information that the null u in node M_{23} was introduced by applying some rule (namely ρ_2) which generated the atom $R(a, u)$. We thus ensure (when proceeding from node M_6 to node M_{10}) that null u is introduced in the same way as before.

The information from the universal node below each existential node is displayed in the second line of each node: here we have pairs consisting of the guessed rule ρ plus the guessed instantiation γ for each atom in the corresponding set $\mathcal{S}$ (as mentioned above, $\mathcal{S}$ is a singleton in all nodes except for M_{23}). For instance, γ_1 in node M_1 denotes the substitution $\gamma_1 = \{x \leftarrow a, y \leftarrow u, z \leftarrow v, w \leftarrow c\}$. Only in node M_{23}, we have 2 pairs $(\rho_2, \gamma_2), (\rho_3, \gamma_3)$ with $\gamma_2 = \{x \leftarrow a, y \leftarrow b\}$ and $\gamma_3 = \{w \leftarrow a, x \leftarrow u, y \leftarrow c\}$. Note that the subscripts i of the γ_i's in Figure 2 are to be understood local to the node. For instance, the two different applications of rule ρ_4 in nodes M_9 and M_{12} are with the instantiations $\gamma_4 = \{x \leftarrow b, y \leftarrow c\}$ (in M_9) and $\gamma_4 = \{x \leftarrow d, y \leftarrow c\}$ (in M_{12}), respectively.

The above example illustrates the close relationship between PTs $\mathcal{T}$ and witnesses $\mathcal{W}$ of the ProofTree algorithm. However, our goal is a one-to-one correspondence, which requires further measures: for witness trees, we thus assume from now on (as in Example 20) that each existential node is merged with its unique universal child node and that nulls are renamed apart: that is, whenever nulls are introduced by a resolution step of ProofTree, then all nulls in $\mathit{nulls}(\gamma(\mathit{body}(\rho))) \setminus \mathit{nulls}(\gamma(\mathit{head}(\rho)))$ must be fresh (that is, they must not occur elsewhere in $\mathcal{W}$). Moreover, we eliminate redundant information from PTs and witnesses by pruning repeated subtrees: let N be a node in a PT $\mathcal{T}$ such that some null is introduced via atom A in $\lambda(N)$, and let N be closest to the root with this property, then prune all other subtrees below all nodes $M \neq N$ with $\lambda(M) = A$. Likewise, if a node in a witness $\mathcal{W}$ contains an atom A and the pair (z, A) in $\mathcal{R}_\mathcal{S}$, then we omit the resolution step for A. We refer to the reduced PT of $\mathcal{T}$ as $\mathcal{T}^*$ and to the reduced witness of $\mathcal{W}$ as $\mathcal{W}^*$. For instance, in the PT in Figure 2, we delete the node N_{10} because the information on how to derive atom $R(a, u)$ is already contained in the subtree rooted at node N_2. Likewise, in the witness in Figure 2, we omit the resolution step for atom $R(a, u)$ in node M_6, because, in this node, we have $\mathcal{R}_\mathcal{S} = \{(u, R(a, u)\}$. Hence, it is known from some resolution step "above" that u must be introduced via this atom and its derivation is checked elsewhere. We thus delete M_{10}.

It is straightforward to construct a reduced witness $\mathcal{W}^*$ from a given reduced PT $\mathcal{T}^*$, and vice versa. We refer to these constructions as $\mathcal{T}2\mathcal{W}$ and $\mathcal{W}2\mathcal{T}$, respectively: Given $\mathcal{T}^*$, we obtain $\mathcal{W}^*$ by a top-down traversal of $\mathcal{T}^*$ and merging siblings if they share a null. Moreover, if the label at a child node in $\mathcal{T}^*$ was created by applying rule ρ with substitution γ, then we add (ρ, γ) to the node label in $\mathcal{W}^*$. Finally, if such a rule application generates a new null z, then we add $(z, \gamma(\mathit{head}(\rho))$ to $\mathcal{R}_\mathcal{S}$ of every child node which still contains null z. Conversely, we obtain $\mathcal{T}^*$ from a reduced witness $\mathcal{W}^*$ by a bottom-up traversal of $\mathcal{W}^*$, where we turn every node labelled with k atoms into k siblings, each labelled with one of these atoms. The edge labels ρ in $\mathcal{T}^*$ are obtained from the corresponding (ρ, γ) labels in the node in $\mathcal{W}^*$. In summary, we have:

▶ **Lemma 21.** *There is a one-to-one correspondence between reduced proof trees $\mathcal{T}^*$ for a ground atom $P(\bar{c})$ and reduced witnesses $\mathcal{W}^*$ for its successful* ProofTree *computations. More precisely, given $\mathcal{T}^*$, we get a reduced witness as $\mathcal{T}2\mathcal{W}(\mathcal{T}^*)$; given $\mathcal{W}^*$, we get a reduced PT as $\mathcal{W}2\mathcal{T}(\mathcal{W}^*)$ with $\mathcal{W}2\mathcal{T}(\mathcal{T}2\mathcal{W}(\mathcal{T}^*)) = \mathcal{T}^*$ and $\mathcal{T}2\mathcal{W}(\mathcal{W}2\mathcal{T}(\mathcal{W}^*)) = \mathcal{W}^*$.*

So far, we have only considered warded Datalog$^{\pm}$, without negation. However, this restriction is inessential. Indeed, by the definition of warded Datalog$^{\pm,\neg sg}$, negated atoms can never contain a null in the chase. Hence, one can easily get rid of negation (in polynomial time data complexity) by computing for one stratum after the other the answer $Q(\Pi, D)$ to query Q with $Q \equiv P(x_1, \ldots, x_n)$, i.e., all ground atoms $P(a_1, \ldots, a_n)$ with $\Pi \cup D \models P(a_1, \ldots, a_n)$. We can then replace all occurrences of $\neg P(t_1, \ldots, t_n)$ in any rule of Π by the positive atom $P'(t_1, \ldots, t_n)$ (for a new predicate symbol P') and add to the instance all ground atoms $P'(a_1, \ldots, a_n)$ with $(a_1, \ldots, a_n) \notin Q(D, \Pi)$. Hence, all our results proved in this section for warded Datalog$^{\pm}$ also hold for warded Datalog$^{\pm,\neg sg}$.

Clearly, there is a one-to-one correspondence between proof trees PT $\mathcal{T}$ and their reduced forms $\mathcal{T}^*$. Hence, together with Lemma 21, we can compute the multiplicities by computing (reduced) witness trees. This allows us to obtain the results below:

▶ **Theorem 22.** *Let Π be a warded Datalog$^{\pm,\neg sg}$ program and D a database (possibly with duplicates). Then there exists a bound K which is polynomial in D, s.t. for every ground atom A:*

1. *A has finite multiplicity if and only if all reduced witness trees of A have depth $\leq K$.*
2. *If A has infinite multiplicity, then there exists at least one reduced witness tree of A whose depth is in $[K+1, 2K]$.*

Proof Idea. Recall that the data structure propagated by the ProofTree algorithm consists of pairs $(\mathcal{S}, \mathcal{R}_{\mathcal{S}})$. We call pairs $(\mathcal{S}, \mathcal{R}_{\mathcal{S}})$ and $(\mathcal{S}', \mathcal{R}_{\mathcal{S}'})$ *equivalent* if one can be obtained from the other by renaming of nulls. The bound K corresponds to the maximum number of non-equivalent pairs $(\mathcal{S}, \mathcal{R}_{\mathcal{S}})$ over the given signature and domain of D. By the logspace bound on this data structure, there can only be polynomially many (w.r.t. D) such pairs. For the first claim of the theorem, suppose that a (reduced) witness tree $\mathcal{W}^*$ has depth greater than K; then there must be a branch with two nodes N and N' with equivalent pairs $(\mathcal{S}, \mathcal{R}_{\mathcal{S}})$ and $(\mathcal{S}', \mathcal{R}_{\mathcal{S}'})$. We get infinitely many witness trees by arbitrarily often iterating the path between N and N'. ◀

In principle, Theorem 22 suffices to prove decidability of FINITENESS and design an algorithm for the MULTIPLICITY: just chase database D with the transformed warded Datalog program Π^+ up to depth $2K$. If the desired ground atom A extended by some tid is generated at a depth greater than K, then conclude that A has infinite multiplicity. Otherwise, the multiplicity of A is equal to the number of atoms of the form $(\iota; A)$ in the chase result. However, this chase of depth $2K$ may produce an exponential number of atoms and hence take exponential time. Below we show that we can in fact do significantly better:

▶ **Theorem 23.** *For warded Datalog$^{\pm,\neg sg}$ programs, both the FINITENESS problem and the MULTIPLICITY problem can be solved in polynomial time.*

Proof Idea. A decision procedure for the FINITENESS problem can be obtained by modifying the ALOGSPACE algorithm ProofTree from [4] in such a way that we additionally "guess" a branch in the witness tree with equivalent labels. The additional information thus needed also fits into logspace.

The MULTIPLICITY problem can be solved in polynomial time by a tabling approach to the ProofTree algorithm. We thus store for each (non-equivalent) value of (S, R_S) how many (reduced) witness trees it has and propagate this information upwards for each resolution step encoded in the (reduced) witness tree. ◀

6 Multiset Relational Algebra (MRA)

Following [20, 21], we consider multisets (or bags) M and elements e (from some domain) with non-negative integer multiplicities, $\mathit{mult}(e, M)$ (recall from Section 2.1 that, by definition, $e \in M$ iff $\mathit{mult}(e, M) \geq 1$). Now consider multiset relations R, S. Unless stated otherwise, we assume that R, S contain tuples of the same arity, say n. We define the following multiset operations of MRA: the *multiset union*, $\uplus$, is defined by $R \uplus S := T$, with $\mathit{mult}(e, T) := \mathit{mult}(e, R) + \mathit{mult}(e, S)$. *Multiset selection*, $\sigma^m_C(R)$, with a condition C, is defined as the multiset T containing all tuples in R that satisfy C with the same multiplicities as in R. For *multiset projection* $\pi^m_{\bar{k}}(R)$, we get the multiplicities $\mathit{mult}(e, \pi^m_{\bar{k}}(R))$ by summing up the multiplicities of all tuples in R that, when projected to the positions $\bar{k} = \langle i_1, \ldots, i_k \rangle$, produce e. For the *multiset (natural) join* $R \bowtie^m S$, the multiplicity of each tuple t is obtained as the product of multiplicities of tuples from R and of tuples from S that join to t.

For the *multiset difference*, two definitions are conceivable: Majority-based, or "monus" difference (see e.g. [15]), given by $R \ominus S := T$, with $\mathit{mult}(e, T) := \mathit{max}\{\mathit{mult}(e, R) - \mathit{mult}(e, S), 0\}$. There is also the "all-or-nothing" difference: $R \otimes S := T$, with $\mathit{mult}(e, T) := \mathit{mult}(e, R)$ if $e \notin S$ and $\mathit{mult}(e, T) := 0$, otherwise. Following [22], we have only considered $\otimes$ so far (implicitly, starting with Datalog$^{\neg s}$, in Section 2.1). The *multiset intersection* $\cap_m$ is not treated or used in any of [2, 20, 21, 22]. Extending the DTB semantics from [21] to $\cap_m$ would treat it as a special case of the join, which may be counter-intuitive, e.g., $\{a, a, b\} \cap_m \{a, a, a, c\} := \{a, a, a, a, a, a\}$. Alternatively, we could define "minority-based" intersection $R \cap_{mb} S$, which returns each element with its minimum multiplicity, e.g., $\{a, a, b\} \cap_{mb} \{a, a, a, c\} := \{a, a\}$.

We consider MRA with the following basic multiset operations: multiset union $\uplus$, multiset projection $\pi^m_{\bar{k}}$, multiset selection σ^m_C, with C a condition, multiset (natural) join $\bowtie^m$, and (all-or-nothing) multiset difference $\otimes$. We can also include *duplicate elimination* in MRA, which becomes operation $\mathcal{DI}$ of Definition 9 when using tids. Being just a projection in the latter case, it can be represented in Datalog. For the moment, we consider multiset-intersection as a special case of multiset (natural) join $\bowtie^m$. It is well known that the basic, set-oriented relational algebra operations can all be captured by means of (non-recursive) Datalog$^{\neg s}$ programs (cf. [1]). Likewise, one can capture MRA by means of (non-recursive) Datalog$^{\neg s}$ programs with multiset semantics (see e.g. [2]). Together with our transformation into set semantics of Datalog$^{\pm,\neg sg}$, we thus obtain:

▶ **Theorem 24.** *The Multiset Relational Algebra (MRA) can be represented by warded Datalog$^{\pm,\neg sg}$ with set semantics.*

As a consequence, the MRA operations can still be performed in polynomial-time (data complexity) via Datalog$^{\pm,\neg sg}$. This result tells us that MRA – applied at the level of an EDB with duplicates – can be represented in warded Datalog, and uniformly integrated under the same logical semantics with an ontology represented in warded Datalog.

We now retake multiset-intersection (and later also multiset-difference), which appears as $\cap_m$ and $\cap_{mb}$. The former does not offer any problem for our representation in Datalog$^{\pm}$ as above, because it is a special case of multiset join. In contrast, the *minority-based* intersection,

$\cap_{mb}$, is more problematic.[2] First, the DTB semantics does not give an account of it in terms of Datalog that we can use to build upon. Secondly, our Datalog$^{\pm}$-based formulation of duplicate management with MRA operations is set-theoretic. Accordingly, to investigate the representation of the bag-based operation $\cap_{mb}$ by means of the latter, we have to agree on a set-based reformulation $\cap_{mb}$. We propose for it a tid-based (set) representation, because tid creation becomes crucial to make it a deterministic operation. Accordingly, for multi-relations P and R with the same arity n (plus 1 for tids), we define:

$$P \cap_{mb} Q := \{(i; \bar{a}) \mid (i; \bar{a}) \in \left\{ \begin{array}{ll} P & \text{if } |\pi_0(\sigma_{1..n=\bar{a}}(P))| \leq |\pi_0(\sigma_{1..n=\bar{a}}(Q))| \\ Q & \text{otherwise} \end{array} \right. \}. \tag{7}$$

Here, we only assume that tids are *local* to a predicate, i.e. they act as values for a surrogate key. Intuitively, we keep for each tuple in the result the duplicates that appear in the relation that contains the minimum number of them. Here, π_0 denotes the projection on the 0-th attribute (for tids), and $\sigma_{1..n=\bar{a}}$ is the selection of those tuples which coincide with $\bar{a}$ on the next n attributes. This operation may be non-commutative when equality holds in the first case of (7) (e.g. $\{\langle 1; a\rangle\} \cap_{mb} \{\langle 2; a\rangle\} = \{\langle 1; a\rangle\} \neq \{\langle 2; a\rangle\} = \{\langle 2; a\rangle\} \cap_{mb} \{\langle 1; a\rangle\}$). Most importantly, it is non-monotonic: if any of the extensions of P or Q grows, the result may not contain the previous result,[3] e.g. $\{\langle 1; a\rangle\} \cap_{mb} \{\langle 2; a\rangle\} = \{\langle 1; a\rangle\} \not\supseteq \{\langle 2; a\rangle\} = (\{\langle 1; a\rangle\} \cup \{\langle 3; a\rangle\}) \cap_{mb} \{\langle 2; a\rangle\}$. (It is still non-monotonic under the DTB semantics.) We get the following inexpressibility results:

▶ **Proposition 25.** *The minority-based intersection with duplicates $\cap_{mb}$ as in (7) cannot be represented in Datalog, (positive) Datalog$^{\pm}$, or FO predicate logic (FOL). The same applies to the majority-based (monus) difference, $\ominus$.*

Proof Idea. By the non-monotonicity of $\cap_{mb}$, it is clear for Datalog and (positive) Datalog$^{\pm}$. For the inexpressibility in FOL, the key idea is that with the help of $\cap_{mb}$ or $\ominus$ we could express the *majority quantifier*, which is known to be undefinable in FOL [24, 25]. ◀

▶ **Proposition 26.** *The minority-based intersection with duplicates $\cap_{mb}$ as in (7) cannot be represented in Datalog$^{\neg s}$. The same applies to the majority-based difference, $\ominus$.*

Proof Idea. It can be shown that if a logic is powerful enough to express any of $\cap_{mb}$ or $\ominus$, then we could express in this logic – for sets A and B – that $|A| = |B \smallsetminus A|$ holds. However, the latter property cannot even be expressed in the logic $L^{\omega}_{\infty\omega}$ under finite structures [13, sec. 8.4.2] and this logic extends Datalog$^{\neg s}$. ◀

7 Conclusions and Future Work

We have proposed the specification of the bag semantics of Datalog in terms of warded Datalog$^{\pm}$ with set semantics and we have also extended this specification to the bag semantics of warded Datalog$^{\pm,\neg sg}$ itself. Our work underlines that warded Datalog$^{\pm}$ is indeed a well-chosen fragment of Datalog$^{\pm}$: it provides a mild extension of Datalog by the restricted use of existentially quantified variables in the rule heads, which suffices to capture certain forms of ontological reasoning [3, 17] and, as we have seen here, the bag semantics of Datalog.

[2] The *majority-based union* operation on bags, that returns, e.g. $\{\{a, a, b, c\}\} \cup_{mab} \{\{a, b, b\}\} := \{\{a, a, b, b, c\}\}$ should be equally problematic.

[3] We could redefine (7) by introducing new tids, i.e. tuples $(f(i), \bar{a})$, for each tuple $(i, \bar{a})$ in the condition in (7), with some function f of tids. The $f(i)$ could be the next tid values after the last one used so far in a list of them. The operation defined in this would still be non-monotonic.

Further extensions of this system, above all to SPARQL with bag semantics based on previous Datalog rewritings [2, 23] are under way. Another interesting direction for future work is to investigate further inexpressibility issues such as, e.g., whether warded Datalog$^{\pm,\neg sg}$ is expressive enough to capture $\cap_{mb}$ and $\ominus$. This work will also include the development of tools to address (in)expressibility results in Datalog$^{\pm}$, with or without negation.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison Wesley, 1994.

2 Renzo Angles and Claudio Gutiérrez. The Multiset Semantics of SPARQL Patterns. In *Proc. ISWC 2016*, volume 9981 of *LNCS*, pages 20–36, 2016. doi:10.1007/978-3-319-46523-4_2.

3 Marcelo Arenas, Georg Gottlob, and Andreas Pieris. Expressive languages for querying the semantic web. In *Proc. PODS'14*, pages 14–26. ACM, 2014. doi:10.1145/2594538.2594555.

4 Marcelo Arenas, Georg Gottlob, and Andreas Pieris. Expressive languages for querying the semantic web. *ACM Trans. Database Syst. (to appear)*, 2018.

5 Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9-10):1620–1654, 2011. doi:10.1016/j.artint.2011.03.002.

6 Luigi Bellomarini, Georg Gottlob, Andreas Pieris, and Emanuel Sallinger. Swift Logic for Big Data and Knowledge Graphs. In *Proc. IJCAI 2017*, pages 2–10. ijcai.org, 2017. doi:10.24963/ijcai.2017/1.

7 Luigi Bellomarini, Emanuel Sallinger, and Georg Gottlob. The Vadalog System: Datalog-based Reasoning for Knowledge Graphs. *PVLDB*, 11(9):975–987, 2018.

8 Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. *J. Artif. Intell. Res.*, 48:115–174, 2013.

9 Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. Datalog$^{\pm}$: a unified approach to ontologies and integrity constraints. In *Proc. ICDT 2009*, volume 361 of *ACM International Conference Proceeding Series*, pages 14–30. ACM, 2009.

10 Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57–83, 2012. doi:10.1016/j.websem.2012.03.001.

11 Andrea Calì, Georg Gottlob, and Andreas Pieris. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, 193:87–128, 2012. doi:10.1016/j.artint.2012.08.002.

12 Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981. doi:10.1145/322234.322243.

13 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory.* Springer, 2nd edition, 1999.

14 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005. doi:10.1016/j.tcs.2004.10.033.

15 Floris Geerts and Antonella Poggi. On database query languages for K-relations. *J. Applied Logic*, 8(2):173–185, 2010. doi:10.1016/j.jal.2009.09.001.

16 Birte Glimm, Chimezie Ogbuji, Sandro Hawke, Ivan Herman, Bijan Parsia, Axel Polleres, and Andy Seaborne. SPARQL 1.1 Entailment Regimes. W3C Recommendation 21 march 2013, W3C, 2013. URL: https://www.w3.org/TR/sparql11-entailment/.

17 Georg Gottlob and Andreas Pieris. Beyond SPARQL under OWL 2 QL Entailment Regime: Rules to the Rescue. In *Proc. IJCAI 2015*, pages 2999–3007. AAAI Press, 2015. URL: http://ijcai.org/Abstract/15/424.

18 Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *Proc. PODS'07*, pages 31–40. ACM, 2007. doi:10.1145/1265530.1265535.

19 David S. Johnson and Anthony C. Klug. Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies. *J. Comput. Syst. Sci.*, 28(1):167–189, 1984. `doi:10.1016/0022-0000(84)90081-3`.

20 Michael J. Maher and Raghu Ramakrishnan. Déjà Vu in Fixpoints of Logic Programs. In *Proc. NACLP 1989*, pages 963–980. MIT Press, 1989.

21 Inderpal Singh Mumick, Hamid Pirahesh, and Raghu Ramakrishnan. The Magic of Duplicates and Aggregates. In *Proc. VLDB 1990*, pages 264–277. Morgan Kaufmann, 1990. URL: `http://www.vldb.org/conf/1990/P264.PDF`.

22 Inderpal Singh Mumick and Oded Shmueli. Finiteness Properties of Database Queries. In *Proc. ADC '93*, pages 274–288. World Scientific, 1993.

23 Axel Polleres and Johannes Peter Wallner. On the relation between SPARQL1.1 and Answer Set Programming. *Journal of Applied Non-Classical Logics*, 23(1-2):159–212, 2013. `doi:10.1080/11663081.2013.798992`.

24 Johan van Benthem and Kees Doets. Higher-Order Logic. In Dov M. Gabbay and Franz Guenthner, editors, *Handbook of Philosophical Logic, Vol. I*, Synthese Library, Vol. 164, pages 275–329. D. Reidel Publishing Company, 1983.

25 Dag Westerståhl. Quantifiers in Formal and Natural. In Dov M. Gabbay and Franz Guenthner, editors, *Handbook of Philosophical Logic, Vol. 14*, pages 223–338. Springer, 2007.