Semantic Web 0 (0) 1
IOS Press

Multiset semantics in SPARQL, Relational Algebra and Datalog

Renzo Angles [a], Claudio Gutierrez [b] and Daniel Hernández [c]
[a] *Department of Computer Science, University of Talca, Chile*
E-mail: rangles@utalca.cl
[b] *Department of Computer Science, University of Chile, and IMFD , Chile*
E-mail: cgutierr@dcc.uchile.cl
[c] *Instute for Artificial Intelligence, University of Stuttgart, Germany*
E-mail: daniel.hernandez@ki.uni-stuttgart.de

Abstract. The paper studies and determines the algebraic and logic structure of the multiset semantics of the core patterns of SPARQL formed by AND, UNION, OPTIONAL, FILTER, MINUS and SELECT. We show that it conforms a robust fragment with the same expressive power of well-known logical and relational query languages, named Datalog and Relational Algebra. Indeed, we identify and develop a logical formalism for multisets, namely non-recursive Datalog with safe negation, with the extension to multiset developed by Mumick et al., and a multiset relational algebra (projection, selection, natural join, arithmetic union and except), based on the framework defined by Dayal et al., and prove that these three formalisms have the same expressive power viewed as query languages.

Keywords: Query Languages, Multisets, SPARQL, Datalog, Relational Algebra

1. Introduction

Informally speaking, multisets are sets where each element could occur multiple times, that is, the number of "copies" of each element matters. In the field of databases, the notion of multisets (also called "duplicates" or "bags")[1] has been studied in several contexts, including programming languages [2, 3], bag languages [4–9], relational algebra [10–12], Datalog [13–17], SQL [18, 19], SPARQL [20–23] and data integration [24].

The incorporation of multisets in query languages like SQL or SPARQL is essentially due to practical concerns: duplicate elimination is expensive and duplicates might be required for some applications, e.g., for aggregation. Although this design decision may be debatable (e.g., see [25]), today multisets are an established reality in database systems [26, 27].

The classical theory behind declarative query languages (like SQL and SPARQL) are formalisms (relational algebra or relational calculus) that for sets have a clear and intuitive semantics for users, developers and theoreticians [28]. The same cannot be said of their extensions to multisets, whose theory is complex (particularly containment of queries) and their practical use not always clear [26]. Worst, there exist several possible ways of extending set relational operators to multisets, which makes the study and design of multiset semantics for query languages challenging.

To illustrate the variety of possible semantics, let us summarize the different extensions to multisets of set operators found in the literature. Consider the following multisets $A = \{\!|a, a, a, b|\!\}$ and $B = \{\!|a, a, d|\!\}$.

[1] There seems to be no agreement on the best terminology [1, p. 27]. In this paper, we will use the word "multiset".

Table 1

Possible ways of extending set operators with multiset semantics in SQL and SPARQL. The table shows several extended relational algebra operations for multisets currently present (or possible to implement) in SQL and SPARQL. Given multisets A and B, and element x, $A(x)$ and $B(x)$ represent the respective multiplicities of x.

Operation	**Operator**	**Multiplicity for x**	**SQL**	**SPARQL**
Projection	$\pi_X(A)$	$\sum_{s\in A,\, s[x]=t} A(s)$	`SELECT` X `FROM` A	`SELECT` X `WHERE` A
Selection	$\sigma_\varphi(A)$	$\begin{cases} A(x) & \text{if } x \text{ satisfies } \varphi, \\ 0 & \text{otherwise.} \end{cases}$	`SELECT` $\star$ `FROM` A `WHERE` φ	A `FILTER` (φ)
Cartesian product	$A \times B$	$A(x) \times B(x)$	A `CROSS JOIN` B	A `AND` B
Join	$A \bowtie_\varphi B$	$A(x) \times B(x)$	$(A$ `CROSS JOIN` $B)$ `WHERE` φ	A `AND` B
Max-union	$A \sqcup B$	$\max(A(x), B(x))$	$(A$ `UNION ALL` $B)$ `EXCEPT ALL` $(A$ `INTERSECT ALL` $B)$	–
Arithmetic union	$A \uplus B$	$A(x) + B(x)$	A `UNION ALL` B	A `UNION` B
Min-intersection	$A \cap B$	$\min(A(x), B(x))$	A `INTERSECT ALL` B	–
Max-intersection	$A \sqcap B$	$A(x) \times B(x)$	A `NATURAL JOIN` B	A `AND` B
Arithmetic difference	$A - B$	$\max(0, A(x) - B(x))$	A `EXCEPT ALL` B	–
Existential negation	$A \setminus B$	$\begin{cases} A(x) & \text{if } B(x) = 0, \\ 0 & \text{otherwise.} \end{cases}$	`SELECT` $\star$ `FROM` A `WHERE` x `NOT IN` (B)	A `MINUS` B

- Types of union: the *max-union* takes the maximum number of occurrences of an element (e.g., $A \sqcup B = \{\!\{a, a, a, b, d\}\!\}$), and the *arithmetic union* adds up multiplicities (e.g., $A \uplus B = \{\!\{a, a, a, a, a, b, d\}\!\}$).
- Types of intersection: the *min-intersection* takes the minimum number of occurrences of each element in the intersection (e.g., $A \cap B = \{\!\{a, a\}\!\}$), and the *max-intersection* returns the product of the multiplicities of each element in the intersection (e.g., $A \sqcap B = \{\!\{a, a, a, a, a\}\!\}$).
- Types of difference: the *arithmetic difference* subtracts multiplicities of elements up to zero (e.g., $A - B = \{\!\{a, b\}\!\}$), and the *existential negation* returns the elements in the first multiset not occurring in the second one, but preserving the multiplicities (e.g., $A \setminus B = \{\!\{b\}\!\}$).

Regarding projection, selection and joins, in each case there is essentially a unique coherent possibility:

- The *Cartesian product* computes the product of the multiplicities (e.g., $A \times B =$ {{ (a,a), (a,a), (a,d), (a,a), (a,a), (a,d), (a,a), (a,a), (a,d), (b,a), (b,a), (b,d) }}).
- The *projection* reduces the number of attributes in each tuple, and gives rise to new multiplicities for resulting tuples (e.g., $\pi_A(A \times B) =$ {{ $a, a, a, a, a, a, a, a, a, b, b, b$ }}).
- The *selection* operation returns the tuples satisfying a given condition, but keeping multiplicities. (e.g., the expression $\sigma_{[1]='a'}(A)$ returns the multiset $\{\!\{a, a, a\}\!\}$).
- The *join* operation also results in the product of the multiplicities of the solutions of the operands, as it is expressed as a Cartesian product followed by a selection.

Table 1 shows a summary of the above operators, and their corresponding implementation in SQL and SPARQL. Note that SQL can express all the operators, whereas SPARQL does not support max-union, min-intersection, and arithmetic difference.

The reader can imagine that combining these operators does not result in a simple nor intuitive algebra, as with classical set operations. In fact, a natural question arises: Are there "reasonable", "well-behaved" (e.g., from an algebraic point of view) groups of these operations for multisets? There are some positive answers.

Dayal et al. [10] introduced a multiset relational algebra formed by the operators of projection, selection, join, max-union, arithmetic union, min-intersection and arithmetic difference. Dayal remarks that max-union, min-

intersection and arithmetic difference treat all copies of a tuple as being identical (i.e. indistinguishable), and form a Boolean algebra. In this sense, these operators combine well with selection in the sense that expected identities such as $\sigma_{P\vee Q}(r) = \sigma_P(r) \cup \sigma_Q(r)$, $\sigma_{P\wedge Q}(r) = \sigma_P(r) \cap \sigma_Q(r)$, and $\sigma_{P\wedge Q}(r) = \sigma_P(r) \bowtie \sigma_Q(r)$ are preserved for multisets. On the other hand, the arithmetic union treats all copies as being distinct, so it is necessary to support the property $\pi_X(R \uplus S) \equiv \pi_X(R) \uplus \pi_X(S)$, which is not satisfied by max-union.

Albert [4] studied the operators of selection, max-union, arithmetic union, min-intersection and arithmetic difference, demonstrating that some of the algebraic properties of sets fail for multisets. However, Albert identified some set-theoretic identities that are relevant for query optimization. Particularly, max-union and min-intersection satisfy the usual algebraic properties, and they correspond to disjunction and conjunction for Boolean selection, respectively; the arithmetic difference conforms to set difference when applied to sets, and corresponds to negation for Boolean selection.

Besides the algebraic coherence, there are two important problems related to multisets: expressive power and complexity. The expressive power and complexity of query languages for bags are studied in several works [5–7, 9]. Similar studies were conducted in the context of Datalog [14–17]. Console et al. [12] studied the expressive power of fragments of bag relational algebra, and of the complexity of computing certain and possible answers.

Objectives and Contributions. As can be seen from Table 1, the current design of SPARQL chooses a particular combination of multiset operators. Several questions arise: Are they extensible to other operators? Do they reflect a simple algebraic or logic known fragment? What is the abstract algebra behind the relational operations?

In this article, we present a formal study of the semantics of multisets in SPARQL, that attempts to answer some of these questions. Specifically, we investigate and identify a good formalism that captures the current semantics of multisets of the core of SPARQL, namely graph patterns. Then, inspired by a well-known version of Datalog, we provide a simple formalism to define a formal semantics for multisets in SPARQL. With these tools, and developing a corresponding algebra, we study the expressive power given by multisets in SPARQL patterns, including a comparison with representative query languages.

Thus, the concrete contributions of our research are the following:

(1) We show that a formalism coming from a logical field, the well-behaved fragment of *non-recursive Datalog with safe negation* (nr-Datalog$^\neg$), is the one that matches the semantics of multisets in SPARQL. More precisely, the natural extension of the usual (set) semantics of Datalog to multisets developed by Mumick et al. [29], which we call *non-recursive multiset Datalog with safe negation* (NRMD$^\neg$).

(2) We develop the relational counterpart of this fragment, using the framework defined by Dayal et al. [10], and come up with a *Multiset Relational Algebra* (MRA) that captures precisely the multiset semantics of the core relational patterns of SPARQL. MRA is based on the operators named projection (π), selection (σ), natural join ($\bowtie$), union ($\uplus$) and filter difference ($\setminus$). As a side effect, this approach gives a new relational view of SPARQL (closer to classical relational algebra and hence more intuitive for people trained in SQL). The identification of this algebra and the proof of the correspondence with the relational core of SPARQL are the main contributions of this paper.

(3) We identify precise equivalent fragments in SPARQL, Multiset Datalog, and Multiset Relational Algebra, and present the translations among these fragments. In Table 2 it is shown a glimpse of these correspondences, whose details are worked in the paper.

This paper extends a previously published conference paper [23]. Herein, we provide extended discussion throughout, we extend the study for some operators that were introduced in the version 1.1 of SPARQL after the publication of our previous work, and we extend the analysis to consider also bag semantics. Some of the additional contributions of this paper are from Hernandez's Ph.D. thesis [30]. The outline is as follows:

In this introduction we pose the problem addressed, the idea that guides the approach, and a summary of findings. Section 2 presents basic background and notations needed to follow the paper. The languages studied in this article are described in Section 3 (SPARQL), Section 4 (NRMD$^\neg$), and Section 5 (MRA). The equivalences in expressive power of these languages is shown in Section 6 (SPARQL $\equiv$ NRMD$^\neg$), Section 7 (MRA $\equiv$ NRMD$^\neg$), and Section 8 (SPARQL $\equiv$ MRA). The related work and the conclusions are presented in Section 9.

Table 2

SCHEMA OF CORRESPONDENCES AMONG: Multiset SPARQL pattern operators; Multiset Relational Algebra (MRA) expressions; Non-Recursive Datalog with safe Negation (NRMD$^\neg$) rules; and SQL expressions. The operator EXCEPT is not part of SPARQL, but it replaces the standard operators MINUS and OPT without changing the expressiveness of the fragment. In MRA, $\uplus$ is the arithmetic union and $\setminus$ is the multiset filter difference. SPARQL patterns are assumed normalized, that is, variables in the filter condition are in the schema of the filtered pattern, and operators EXCEPT AND UNION assume operands with the same schema. Patterns P_1 and P_2 occurring in the SPARQL pattern are associated to atoms L_1 and L_2 in the NRMD$^\neg$ translation, and relations r_1 and r_2 in the MRA translations, respectively.

SPARQL	NRMD$^\neg$	MRA	SQL
SELECT $\mathcal{X}$ P_1	$L \leftarrow L_1, \text{null}(\mathcal{X} \setminus \mathcal{X}_1)$	$\pi_{\mathcal{X}}(r_1) \bowtie \text{null}(\mathcal{X} \setminus \mathcal{X}_1)$	`SELECT` $\mathcal{X}$ `FROM` r_1 `NATURAL JOIN` $\text{null}(\mathcal{X} \setminus \mathcal{X}_1)$
P_1 FILTER $X = a$	$L \leftarrow L_1, X = a$	$\sigma_{X=a}(r_1)$	`FROM` r_1 `WHERE` $X = a$
P_1 AND P_2	$L \leftarrow v_1(L_1), v_2(L_2), \text{comp}(v_1, v_2, \mathcal{X})$	$\pi_{\bar{\mathcal{X}}}(\rho_{v_1}(r_1) \bowtie \rho_{v_2}(r_2) \bowtie \text{comp}(v_2, v_2, \mathcal{X}))$	`SELECT` $\mathcal{X}$ `FROM` r_1 `NATURAL JOIN` r_2 `NATURAL JOIN` $\text{comp}(v_2, v_2, \mathcal{X})$
P_3 UNION P_4	$L \leftarrow L_1$; $L \leftarrow L_2$	$r_1 \uplus r_2$	r_1 `UNION ALL` r_2
P_1 EXCEPT P_2	$L \leftarrow L_1, \neg L_2$	$r_1 \setminus r_2$	r_1 `EXCEPT` r_2

2. Preliminaries

This section provides the concepts and formal notation we will follow regarding multisets and the expressive power of query languages.

2.1. Multisets

Informally, a multiset is an unordered collection of elements where each element may occur more than once. Formally, a *multiset* is a tuple $M = (S, card)$ where S is the underlying set of M (containing the distinct elements), and $card : S \to \mathbb{N}^+$ is a function that defines the cardinality or multiplicity in M of each element $a \in S$. We write $\text{set}(M) = S$ to denote that the underlying set of M is S. Given a positive natural number n, we write $\text{card}(a, M) = n$ to denote that $a \in \text{set}(M)$ and the cardinality of a in M is n, and, abusing notation, we write $\text{card}(a, M) = 0$ if $a \notin \text{set}(M)$. We say that $a \in M$ when $\text{card}(a, M) \geqslant 1$.

2.2. Comparing the expressive power of query languages

Next we present the notion of query language and two notions of expressive power used in this paper.

Definition 1 (Query language)**.** *A query language $\mathcal{L}$ is a quadruple $(\mathcal{Q}, \mathcal{D}, \mathcal{S}, \text{Eval})$, where $\mathcal{Q}$ is the set of queries in $\mathcal{L}$, $\mathcal{D}$ is the set of databases in $\mathcal{L}$, $\mathcal{S}$ is the set of results in $\mathcal{L}$, and $\text{Eval} : \mathcal{Q} \times \mathcal{D} \to \mathcal{S}$ is the query evaluation function of $\mathcal{L}$.*

Let $\mathcal{L} = (\mathcal{Q}, \mathcal{D}, \mathcal{S}, \text{Eval})$ be a query language. Two queries $Q_1, Q_2 \in \mathcal{Q}$ are said to be *equivalent*, denoted $Q_1 \equiv Q_2$, if for every database $D \in \mathcal{D}$, it holds that $\text{Eval}(Q_1, D) = \text{Eval}(Q_2, D)$, i.e., they return the same result for all input databases.

Given a query language $(\mathcal{Q}, \mathcal{D}, \mathcal{S}, \text{Eval})$, a query $Q \in \mathcal{Q}$ determines a function $q : \mathcal{D} \to \mathcal{S}$ defined as $q(D) = \text{Eval}(Q, D)$, called the *query function* of Q. Two queries Q_1 and Q_2 are thus equivalent, denoted $Q_1 \equiv Q_2$, if they determine the same query function.

In this context, the *expressive power* of a query language $\mathcal{L}$ is understood as the set of all query functions that are expressible by $\mathcal{L}$. Abiteboul et al. [28] summarizes how this notion is used to compare the expressive power of relational algebra, Datalog, and relational calculus. In the context of SPARQL, Zhang and Van den Bussche [31], Kontchakov et al. [32], and Angles and Gutierrez [33] use this notion to compare different fragments of SPARQL.

The query languages studied in this paper do not hold the aforementioned property of having a common set of databases and results. Thus, we need an extended version of the notion of expressive power as in Definition 2 below.

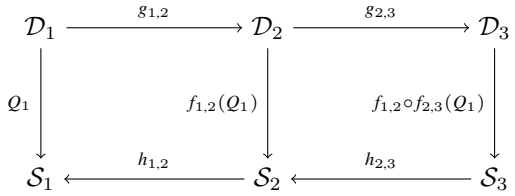

Fig. 1. Transitivity of language containment. The figure represents three languages $\mathcal{L}_i = (\mathcal{Q}_i, \mathcal{D}_i, \mathcal{S}_i, \mathrm{Eval}_i)$ where $i \in \{1, 2, 3\}$. The containment of a language $\mathcal{L}_i$ in $\mathcal{L}_{i+1}$ is given by the simulation $(f_{i,i+1},\ g_{i,i+1},\ h_{i,i+1})$. The transitive containment of $\mathcal{L}_1$ in $\mathcal{L}_3$ is given by the simulation $(f_{1,2} \circ f_{2,3},\ g_{1,2} \circ g_{2,3},\ h_{2,3} \circ h_{1,2})$ where $\circ$ denotes the composition of functions (e.g., $g_{1,2} \circ g_{2,3}$ denotes the function from D_1 to D_3 that results from composing $g_{1,2}$ and $g_{2,3}$).

This extended notion is implicit in the translations by Polleres [34], Angles and Gutierrez [35, 36], and Polleres and Wallner [37].

Definition 2 (Generalized expressive power)**.** *Given two query languages $\mathcal{L}_1 = (\mathcal{Q}_1, \mathcal{D}_1, \mathcal{S}_1, \mathrm{Eval}_1)$ and $\mathcal{L}_2 = (\mathcal{Q}_2, \mathcal{D}_2, \mathcal{S}_2, \mathrm{Eval}_2)$, we say that $\mathcal{L}_1$ is contained in $\mathcal{L}_2$ if and only if there exist functions $g : \mathcal{D}_1 \to \mathcal{D}_2$ (called the* database translation*), $f : \mathcal{Q}_1 \to \mathcal{Q}_2$ (called the* query translation*), and $h : \mathcal{S}_2 \to \mathcal{S}_1$ (called the* results translation*), such that for every $Q \in \mathcal{Q}_1$ and database $D \in \mathcal{D}_1$ it holds that*

$$\mathrm{Eval}_1(Q, D) = h(\mathrm{Eval}_2(f(Q), g(D))).$$

If that is the case, we say that the triple (f, g, h) is a simulation *of $\mathcal{L}_1$ in $\mathcal{L}_2$. We say that the languages $\mathcal{L}_1$ and $\mathcal{L}_2$ have the same expressive power, denoted $\mathcal{L}_1 \cong \mathcal{L}_2$, if and only if $\mathcal{L}_1$ is contained in $\mathcal{L}_2$ and $\mathcal{L}_2$ is contained in $\mathcal{L}_1$.*

Observe that, like the classical notion and as it is desirable for a notion of expressive power, the extended notion defined above defines a partial order: the containment relation on the equivalence classes over the relation $\cong$. Indeed, the reflexivity and antisymmetry follow directly from the definition, whereas the transitivity is shown in Figure 1.

2.3. Comparing SPARQL, NRMD$^\neg$ and MRA

In the remainder of this paper, we define three families of query languages: Non-recursive Multiset Datalog with Safe Negation (NRMD$^\neg$), Multiset Relational Algebra (MRA) and a core fragment of SPARQL. After defining these languages, we present simulations that show the equivalence among these three families of query languages. These simulations are depicted in Figure 2.

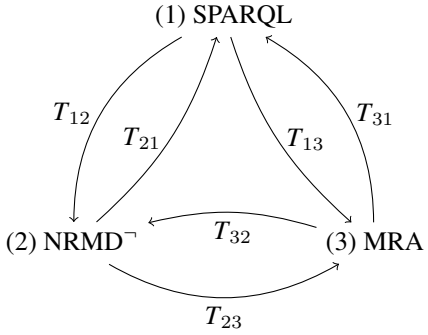

Fig. 2. The triangle of simulations among SPARQL, Non-Recursive Multiset Datalog with Safe Negation (NRMD$^\neg$), and Multiset Relational Algebra (MRA) described in this paper. The query languages are identified by numbers, and T_{ij} denotes the simulation of language i using language j.

3. Multiset SPARQL

SPARQL [38, 39] is the standard query language for RDF. In this paper we study a fragment of SPARQL, the "relational core", described by Angles and Gutierrez [36], which considers the operators FILTER, SELECT, AND, UNION, and EXCEPT. This fragment captures essentially the graph pattern queries in SPARQL. In fact, it has been proved [32, 36] that it is mutually expressible with the standard-core consisting of the operators FILTER, SELECT, AND, UNION, OPTIONAL, and MINUS. (In what follows when speaking of "SPARQL" we will mean this fragment).

3.1. RDF Graphs

Assume two disjoint infinite sets $\mathbf{I}$ and $\mathbf{L}$, called IRIs and literals, respectively. An *RDF term* is an element in the set $\mathbf{T} = \mathbf{I} \cup \mathbf{L}$. An *RDF triple* is a triple $(s, p, o) \in \mathbf{I} \times \mathbf{I} \times \mathbf{T}$ where s is called the *subject*, p is called the *predicate* and o is called the *object*. An *RDF graph* (just graph from now on) is a set of RDF triples. The *union* of graphs, $G_1 \cup G_2$, is the set theoretical union of their sets of triples.

Note: In addition to $\mathbf{I}$ and $\mathbf{L}$, RDF and SPARQL admit as terms anonymous resources called blank nodes. In this paper, we do not include them to help focus on the issues arising from multisets. Avoiding blank nodes does not affect the results presented in this paper. Indeed, in SPARQL, blank nodes in the data can be consistently replaced by IRIs and produce equivalent query results, and blank nodes in queries can be replaced by fresh variables without changing the semantics of the query [40].

3.2. SPARQL Syntax

Assume the existence of an infinite set $\mathbf{V}$ of variables disjoint from $\mathbf{T}$ (RDF terms). A *filter condition* is defined recursively as follows: (i) If $?X, ?Y \in \mathbf{V}$ and $c \in \mathbf{T}$ then $(?X = c)$, $(?X = ?Y)$ and $\mathrm{bound}(?X)$ are atomic filter conditions; (ii) If φ_1,φ_2 are filter conditions then $(\varphi_1 \wedge \varphi_2)$,$(\varphi_1 \vee \varphi_2)$ and $\neg\varphi_1$ are complex filter conditions.

A SPARQL *pattern* is defined recursively as follows:

- A triple from $(\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{L} \cup \mathbf{V})$ is a pattern called a *triple pattern*.
- If P_1 and P_2 are patterns then $(P_1 \,\mathrm{AND}\, P_2)$, $(P_1 \,\mathrm{UNION}\, P_2)$, and $(P_1 \,\mathrm{EXCEPT}\, P_2)$ are patterns.
- If P is a pattern and φ is a filter condition then $(P \,\mathrm{FILTER}\, \varphi)$ is a pattern.
- If W is a set of variables and P_1 is a pattern then $(\mathrm{SELECT}\, W\, P_1)$ is a pattern.

3.3. SPARQL Semantics

A *solution mapping* (or just *mapping* from now on) is a partial function $\mu : \mathbf{V} \to \mathbf{T}$ where the domain of μ, denoted $\mathrm{dom}(\mu)$, is the subset of $\mathbf{V}$ where μ is defined. We write $\mu_\emptyset$ to denote the mapping with empty domain (i.e., $\mathrm{dom}(\mu_\emptyset) = \emptyset$). Given $?X \in \mathbf{V}$ and $c \in \mathbf{T}$, we write $\mu(?X) = c$ to denote that μ maps variable $?X$ to term c. Given a finite set of variables W, the restriction of a mapping μ to W, denoted $\mu_{|W}$, is a mapping μ' which satisfies that $\mathrm{dom}(\mu') = W \cap \mathrm{dom}(\mu)$ and $\mu'(?X) = \mu(?X)$ when $?X \in \mathrm{dom}(\mu')$. Given a mapping μ, a variable $?X \in \mathrm{dom}(\mu)$ and a variable $?Y \notin \mathrm{dom}(\mu)$, the function $\lambda_{?X/?Y}(\mu)$ returns a mapping μ' which satisfies that $\mathrm{dom}(\mu') = (\mathrm{dom}(\mu) \setminus \{?X\}) \cup \{?Y\}$ and $\mu'(?Z) = \mu(?Z)$ for every variable $?Z \in \mathrm{dom}(\mu) \setminus \{?X\}$, and $\mu'(?Y) = \mu(?X)$. Two solution mappings μ_1, μ_2 are *compatible*, denoted $\mu_1 \sim \mu_2$, when for all $?X \in \mathrm{dom}(\mu_1) \cap \mathrm{dom}(\mu_2)$ they satisfy that $\mu_1(?X) = \mu_2(?X)$, that is, when $\mu_1 \cup \mu_2$ is also a mapping. Note that two mappings with disjoint domains are always compatible.

Let Ω be a multiset of solution mappings. The *domain of variables* in Ω, denoted $\mathrm{dom}(\Omega)$, is defined as the set union of the domains of the variables occurring in the solution mappings of Ω. Given a mapping μ, the cardinality of μ in Ω will be denoted as $\mathrm{card}(\mu, \Omega)$. If $\mu \notin \Omega$ then $\mathrm{card}(\mu, \Omega) = 0$.

The evaluation of a filter condition φ under a mapping μ, denoted $\mu(\varphi)$, is defined in a three-valued logic with values *true*, *false* and *error*. We say that μ satisfies φ when $\mu(\varphi) = \mathit{true}$. The semantics of $\mu(\varphi)$ is defined recursively as follows:

Table 3
Evaluation of complex filter conditions [38, §17.2], where μ is a solution mapping, and φ_1,φ_2 are filter conditions.

$\mu(\varphi_1)$	$\mu(\varphi_2)$	$\mu(\varphi_1) \wedge \mu(\varphi_2)$	$\mu(\varphi_1) \vee \mu(\varphi_2)$
true	true	true	true
true	false	false	true
true	error	error	true
false	true	false	true
false	false	false	false
false	error	false	error
error	true	error	true
error	false	false	error
error	error	error	error

$\mu(\varphi_1)$	$\neg(\mu(\varphi_1))$
true	false
false	true
error	error

- If φ is $?X = c$ and $c \in \mathbf{T}$, then: (a) If $?X \in \mathrm{dom}(\mu)$ then $\mu(\varphi) =$ *true* when $\mu(?X) = c$ and $\mu(\varphi) =$ *false* otherwise; (b) If $?X \notin \mathrm{dom}(\mu)$ then $\mu(\varphi) =$ *error*.
- If φ is $?X = ?Y$ and $?X, ?Y \in \mathrm{dom}(\mu)$, then $\mu(\varphi) =$ *true* when $\mu(?X) = \mu(?Y)$, and $\mu(\varphi) =$ *false* otherwise. If $?X \notin \mathrm{dom}(\mu)$ or $?Y \notin \mathrm{dom}(\mu)$ then $\mu(\varphi) =$ *error*.
- If φ is $\mathrm{bound}(?X)$ and $?X \in \mathrm{dom}(\mu)$ then $\mu(\varphi) =$ *true*; otherwise $\mu(\varphi) =$ *false*.
- If φ is a complex filter condition, then it is evaluated following the three valued logic shown in Table 3.

The evaluation of a pattern P on a graph G is defined as a function $[\![P]\!]_G$, which returns a multiset of mappings. Let P_1, P_2 be SPARQL patterns, φ be a filter condition and W be a set of variables. For simplicity of reading, denote $M = [\![P]\!]_G$, $M_1 = [\![P_1]\!]_G$, and $M_2 = [\![P_2]\!]_G$. The evaluation $[\![P]\!]_G$ is defined recursively as follows:

- If P is a triple pattern t then $\mathrm{set}(M) = \{\mu \mid \mathrm{dom}(\mu) = \mathrm{var}(t), \mu(t) \in G\}$, where $\mu(t)$ is the triple obtained by replacing the variables in t according to μ, and $\mathrm{card}(\mu, M) = 1$.
- If P is $(P_1 \,\mathrm{AND}\, P_2)$ then $\mathrm{set}(M) = \{\mu_1 \cup \mu_2 \mid \mu_1 \in M_1, \mu_2 \in M_2, \text{and } \mu_1 \sim \mu_2\}$ and $\mathrm{card}(\mu, M) = \sum_{\mu=\mu_1\cup\mu_2} \mathrm{card}(\mu_1, M_1) \times \mathrm{card}(\mu_2, M_2)$.
- If P is $(P_1 \,\mathrm{UNION}\, P_2)$ then $\mathrm{set}(M) = \{\mu \mid \mu \in M_1 \vee \mu \in M_2\}$ and $\mathrm{card}(\mu, M) = \mathrm{card}(\mu, M_1) + \mathrm{card}(\mu, M_2)$.
- If P is $(P_1 \,\mathrm{EXCEPT}\, P_2)$ then $\mathrm{set}(M) = \{\mu \mid \mu \in M_1, \mu \notin M_2\}$ and $\mathrm{card}(\mu, M) = \mathrm{card}(\mu, M_1)$.
- If P is $(P_1 \,\mathrm{FILTER}\, \varphi)$ then $\mathrm{set}(M) = \{\mu \mid \mu \in M_1, \mu(\varphi) =$ *true*$\}$ and $\mathrm{card}(\mu, M) = \mathrm{card}(\mu, M_1)$.
- If P is $(\mathrm{SELECT}\, W\, P_1)$ then
  $\mathrm{set}(M) = \{\mu' \mid \mu' = \mu_{|W} \wedge \mu \in M_1\}$ and
  $\mathrm{card}(\mu', M) = \sum_{\mu'=\mu_{|W}} \mathrm{card}(\mu, M_1)$.

To facilitate the translation from SPARQL to relational algebra and Datalog, we use the difference operator EXCEPT in SPARQL, called SetMinus by Kontchakov et al. [32]. Kontchakov et al. [32] proved that, over this fragment, the operator EXCEPT and the pair of standard operators $\{\mathrm{MINUS}, \mathrm{OPTIONAL}\}$ are mutually expressible.

3.4. Normalization of patterns

The solution mappings of a SPARQL pattern P may have different domains. To translate SPARQL to languages built upon relations, we require representing multisets of mappings as relations whose tuples have the same set of attributes. This set of attributes has to contain all variables that can appear in the solution mappings of P. The SPARQL specification [39] defines a finite set of variables, called *in-scope*, that include all variables of a SPARQL pattern P that can occur in solutions of P. To complete the relation, unbound values need to be denoted with a distinguished constant of the target languages.

Example 1. *Assume a pattern P with in-scope variables $?X$, $?Y$, and $?Z$ which returns the multiset of mappings $\Omega = \{\!\{\{?X \mapsto a\}, \{?X \mapsto b, ?Y \mapsto c\}, \{?Y \mapsto d\}\}\!\}$. Since all variables in the solution mappings are ensured to be in-scope variables of P, we can represent this multiset of mappings as the following relation ($\perp$ denotes the distinguished constant to denote unbound values):*

$$\left[\begin{array}{ccc} ?X & ?Y & ?Z \\ \hline a & \bot & \bot \\ b & c & \bot \\ \bot & d & \bot \end{array}\right].$$

In-scope variables are defined as follows. Let P_1, P_2 and P_3 be patterns, φ be a filter condition, and W be a set of variables. The set of *in-scope variables* of a pattern P, denoted $\mathrm{inScope}(P)$, is defined recursively as follows:

1. If P is a triple pattern then $\mathrm{inScope}(P) = \mathrm{var}(P)$;
2. If P is $(P_1 \text{ AND } P_2)$ or $(P_1 \text{ UNION } P_2)$ then $\mathrm{inScope}(P) = \mathrm{inScope}(P_1) \cup \mathrm{inScope}(P_2)$;
3. If P is $(P_1 \text{ FILTER } \varphi)$ or $(P_1 \text{ EXCEPT } P_2)$ then $\mathrm{inScope}(P) = \mathrm{inScope}(P_1)$;
4. If P is $(\text{SELECT } W\ P_1)$ then $\mathrm{inScope}(P) = W$.

So far, we have described how to translate the results of SPARQL queries to relations. However, languages built upon relations have some restrictions that difficult a straightforward translation of the SPARQL operations. The relational selection operation requires all attributes in the selection formula being attributes of the relation; the relational union is done over relations of the same schema; and the relational difference requires all variables in the subtrahend be instanced in the minuend. Conversely, SPARQL does not have these restrictions. We next present a normal form to simplify the translation from SPARQL to relational languages by satisfying the constraints of the target languages.

Definition 3 (SPARQL normal form)**.** *A pattern P is said to be in* normalized *or in* normal form *if the following conditions hold:*

1. *Every sub-pattern $(P_1 \text{ FILTER } \varphi)$ in P holds that $\mathrm{var}(\varphi) \subseteq \mathrm{inScope}(P_1)$;*
2. *Every sub-pattern $(P_1 \text{ UNION } P_2)$ in P holds that $\mathrm{inScope}(P_1) = \mathrm{inScope}(P_2)$;*
3. *Every sub-pattern $(P_1 \text{ EXCEPT } P_2)$ in P holds that $\mathrm{inScope}(P_1) = \mathrm{inScope}(P_2)$.*

Lemma 1. *Every SPARQL query (in the fragment described in Section 3.2) can be rewritten as an equivalent normalized SPARQL query.*

Proof. The conditions that make a pattern normalized refer to restrictions to the in-scope variables of patterns. Patterns that are not normalized include at least one sub-pattern that has either the form $(P_1 \text{ FILTER } \varphi)$, $(P_2 \text{ UNION } P_3)$, or $(P_2 \text{ EXCEPT } P_3)$, where φ contains a variable $?X \notin \mathrm{inScope}(P_1)$, and $\mathrm{inScope}(P_2) \neq \mathrm{inScope}(P_3)$. We next present a method to normalize these patterns.

Given a pattern P, and a finite set of variables X, $P \equiv (\text{SELECT } (\mathrm{inScope}(P) \cup X)\ P)$. Indeed, a mapping μ is a solution of pattern $(\text{SELECT } (\mathrm{inScope}(P) \cup X)\ P)$ if and only there exists a solution mapping μ' of pattern P such that $\mu = \mu'|_{\mathrm{inScope}(P) \cup X}$. By the definition of the in-scope variables, $\mathrm{dom}(\mu') \subseteq \mathrm{inScope}(P)$. Then, $\mathrm{dom}(\mu') \subseteq \mathrm{inScope}(P) \cup X$. Then, $\mu = \mu'$. Hence, $P \equiv (\text{SELECT } (\mathrm{inScope}(P) \cup X)\ P)$.

Let P'_1, P'_2, and P'_3 be the patterns defined as follows:

$$P'_1 = (\text{SELECT } (\mathrm{inScope}(P_1) \cup \mathrm{var}(\varphi))\ P_1),$$
$$P'_2 = (\text{SELECT } (\mathrm{inScope}(P_2) \cup \mathrm{inScope}(P_3))\ P_2),$$
$$P'_3 = (\text{SELECT } (\mathrm{inScope}(P_2) \cup \mathrm{inScope}(P_3))\ P_3).$$

Since $P'_1 \equiv P_1$, $P'_2 \equiv P_2$, and $P'_3 \equiv P_3$, the following equivalences are hold:

$$(P_1 \text{ FILTER } \varphi) \equiv (P'_1 \text{ FILTER } \varphi),$$
$$(P_2 \text{ UNION } P_3) \equiv (P'_2 \text{ UNION } P'_3),$$
$$(P_2 \text{ EXCEPT } P_3) \equiv (P'_2 \text{ EXCEPT } P'_3).$$

Unlike the patterns on the left side of these equivalences, the patterns on the right side are normalized. Indeed, by the definition of the inScope function, $\mathrm{var}(\varphi) \subseteq \mathrm{inScope}(P'_1)$ and $\mathrm{inScope}(P'_2) = \mathrm{inScope}(P'_3)$. Hence, these equivalences can be used to normalize SPARQL patterns. □

Example 2. *Let Q be the pattern $(P_1 \text{ UNION } P_2)$ where P_1 is the triple pattern $(?X, \text{is}, \text{person})$ and P_2 is the triple pattern $(?X, \text{email}, ?Y)$, and G be the RDF graph that includes the triples $(a, \text{is}, \text{person})$ and $(a, \text{email}, \text{a@ex.org})$. Pattern Q is not in normal form because variable $?Y$ is in $\text{inScope}(P_2)$, but not in $\text{inScope}(P_1)$. The normal form of pattern Q is the query Q' that results from replacing P_1 by the pattern $P_1' = (\text{SELECT } ?X\,?Y\,(?X, \text{is}, \text{person}))$. Patterns Q and Q' are equivalent because patterns P_1 and P_1' return the same answers, namely the multiset of mappings $\Omega_1 = \{\!\{\{?X \mapsto a\}\}\!\}$. Note that, variable $?Y$ is not in the answers of P_1 nor P_1'. However, variable $?Y$ is in $\text{inScope}(P_1')$ but not in $\text{inScope}(P_1)$. Using the in-scope variables of the patterns to translate the results of patterns P_1 and P_1' as relations we get the respective relations $\left[\begin{array}{c} ?X \\ \hline a \end{array}\right]$ and $\left[\begin{array}{cc} ?X & ?Y \\ \hline a & \perp \end{array}\right]$.*
Although both relations represent the same multiset of mappings, just the second relation has the same attributes as the result of pattern P_2, and thus can be operated with the relational union.

4. Non-Recursive Multiset Datalog with Safe Negation (NRMD$^\neg$

This section presents an extension of Datalog to support multiset semantics. Based on the work of Mumick et al. [29], a database is defined to allow duplicate facts, and the evaluation and multiplicity of a solution fact is given by the number of different proofs for that fact. We extended Mumick's formalism in [23] to provide a more complete formalism including negation, which we call MD$^\neg$. Additionally, we follow the work of Bertossi et al. [41] for the semantics of MD$^\neg$. We call *Non-Recursive Multiset Datalog with Safe Negation* (NRMD$^\neg$) to the fragment of MD$^\neg$ restricted to non-recursive queries.

4.1. NRMD$^\neg$ Syntax

Assume three disjoint sets: *variables*, *constants* and *predicate names*. A *term* is either a variable or a constant. An *atom* is an expression $p(t_1, \ldots, t_n)$ where p is a predicate name and each t_i is a term. An equality expression will be represented by an atom of the form $eq(t_1, t_2)$. A *literal* is either an atom (i.e. a *positive literal* A) or the negation of an atom (i.e. a *negative literal* $\neg A$). Given a literal L, we use $\text{var}(L)$ to denote the variables in L. A Horn Clause, or simply clause, is an expression containing at most one positive literal. There are three types of clauses: facts, rules and goals.

A *fact* is a positive literal which does not contain any variables. A *MD$^\neg$ Database* is a finite multiset of facts. The *vocabulary* of a MD$^\neg$ database D is a pair (P, α) where P is the set of predicate names occurring in the facts of D, and α is a function defining the arity of each predicate name in P, i.e. if $p(c_1, \ldots, c_n) \in D$ then $\alpha(p) = n$. The predicate names occurring in D are called *extensional*.

A *rule* is an expression $L_{n+1} \leftarrow L_1, \ldots, L_n$ where L_{n+1} is a positive literal with no constants called the head, and $L_1, \ldots, L_n$ ($n \geqslant 1$) is a set of literals called the *body*. A variable X occurs positively in a rule R if and only if X occurs in a positive literal in the body of R. A rule R is said to be *safe* if all the variables occurring in R occur positively in R. Additionally, we will assume that every literal in the body of a rule has a variable at least.

A *program* Π is a finite set of rules. The predicate names occurring in the head of the rules of Π are called *intensional*. A program Π is *safe* if all the rules of Π are safe. A *MD$^\neg$ program* is a safe program.

The *dependency graph* of a program Π is a digraph (N, E) where the set of nodes N is the set of predicates names that occur in the literals of Π, and there is an edge (p_1, p_2) in E if there is a rule in Π whose body contains the predicate name p_1, and whose head contains the predicate name p_2. A program is said to be *non-recursive* if its dependency graph is acyclic. A NRMD$^\neg$ program is a MD$^\neg$ that is non-recursive.

A *goal clause* is an atom with no constants. A MD$^\neg$ *query* is a pair (L, Π) where L is a goal clause, and Π is a MD$^\neg$ program. A NRMD$^\neg$ query is a MD$^\neg$ query (L, Π) such that Π is non-recursive. A NRMD$^\neg$ database is a MD$^\neg$ database.

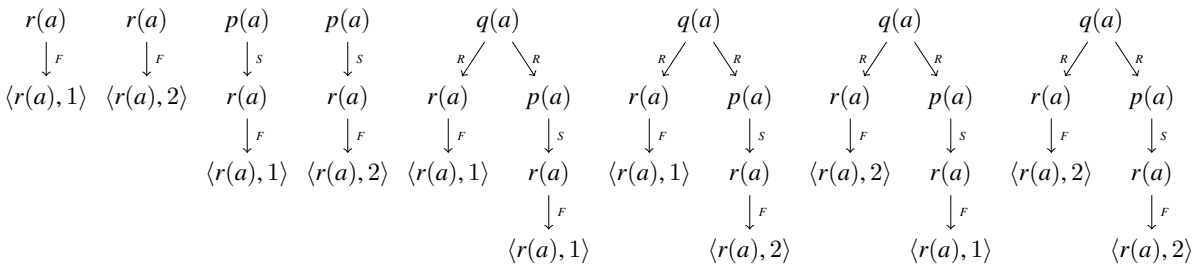

Fig. 3. Example of derivation trees. Let D be the database $\{F, F\}$ where F is the fact $r(a)$, and Π be the program $\{R, S\}$ where R is the rule $q(X) \leftarrow r(X), p(X)$ and S is the rule $p(X) \leftarrow r(X)$. This figure shows the derivation trees of Π with respect to D.

4.2. NRMD$^\neg$ Semantics

We follow the formalisms by Mumick et al. [29] and Bertossi et al. [41] that use a proof-theoretic semantics for NRMD$^\neg$ programs.

A *substitution* is a partial function θ from variables to constants. Given a literal L (positive or negative), and a substitution θ, we write $\theta(L)$ to denote the result of replacing all variables x occurring in L with $\theta(x)$. Informally, an answer to a query (L, Π) where Π is a NRMD$^\neg$ program, over a database D, will be a multiset of substitutions with the same domain, each one obtained from one proof showing that this substitution works. Let us state the formal version.

We will need the following definition, introduced by Mumick et al. [29], to identify the different copies of each duplicate element. The *colored set* of a multiset M, denoted $\mathrm{coloring}(M)$, is the set $C = \{\langle a, i_a\rangle \mid a \in \mathrm{set}(M) \text{ and } 1 \leqslant i_a \leqslant \mathrm{card}(a, M)\}$. The elements of $\langle a, i\rangle \in C$ are called the colored copies of a. Sometimes we will use the notation $\mathrm{coloring}^{-1}(C)$ to define the multiset defined by C when forgetting the "colors", and write $\mathrm{coloring}^{-1}(\langle a, i\rangle) = a$.

Let D be a NRMD$^\neg$ database and Π a NRMD$^\neg$ program. The *derivation trees* of Π with respect to D, denoted $\mathrm{dt}(\Pi, D)$, are defined recursively as follows:

1. Let F be a fact, $\mathrm{card}(F, D) = k$ with $k \geqslant 0$, and $\langle F, i\rangle$ be a colored copy of F. Then there are k derivation trees t_i (for $1 \leqslant i \leqslant k$) where each derivation tree t_i consists of:
   - two nodes F and $\langle F, i\rangle$;
   - root F; and
   - an edge $F \rightarrow \langle F, i\rangle$ with label $\langle F, i\rangle$.
2. Let $R \in \Pi$ be a rule of the form $L_{n+1} \leftarrow A_1, \ldots, A_m, \neg A_{m+1}, \ldots, \neg A_n$, where $A_1, \ldots, A_n$ are atoms and θ be a substitution. Then, for each tuple $(t_1, \ldots, t_m)$ of derivation trees $t_i \in \mathrm{dt}(\Pi, D)$ such that: (1) for each $i \leqslant m$, the root of t_i unifies with A_i under θ and (2) for each $j > m$ there is no derivation tree in $\mathrm{dt}(\Pi, D)$ whose root is $\theta(A_j)$; the rule R and substitution θ generates the derivation tree consisting of:
   - Root $\theta(L_{n+1})$;
   - (Ordered) edges $\theta(L_{n+1}) \rightarrow r_i$ with label R, where for $i = 1, ..., m$, r_i is the root of t_i and for $j > m$, $r_j = \theta(\neg(A_j))$.

Let Π be a nr-Datalog$^\neg$ program, D a nr-Datalog$^\neg$ database and F a fact. A tree $t \in \mathrm{dt}(\Pi, D)$ is said to be *a proof* for fact F if the root of t is a colored version of F. The multiset of *atoms* of Π in D, denoted $\mathrm{atoms}(\Pi, D)$, is the multiset of facts F such that there is a proof for F in $\mathrm{dt}(\Pi, D)$, and the multiplicity of F in $\mathrm{atoms}(\Pi, D)$ is the number of proofs of F. Figure 3 shows the derivation trees that are proofs for the facts that are derived from an example MD program. The facts $r(a)$, $p(a)$ and $q(a)$ belong to $\mathrm{atoms}(\Pi, D)$ with multiplicities 2, 2, and 4.

The NRMD$^\neg$ query language over a vocabulary τ is the query language $(\mathcal{Q}, \mathcal{D}, \mathcal{S}, [\![\cdot]\!].)$ where:

1. $\mathcal{Q}$ is the set of NRMD$^\neg$ queries over τ;

2. $\mathcal{D}$ is the set of NRMD$^\neg$ databases over τ;
3. $\mathcal{S}$ is the set of NRMD$^\neg$ answers (i.e., pairs (V, M) where V is a set of variables and M is a multiset of substitutions θ with $\mathrm{dom}(\theta) = V$); and
4. $[\![\cdot]\!]$ is the function that receives a NRMD$^\neg$ query (L, Π) and a NRMD$^\neg$ database D, and returns a NRMD$^\neg$ answer (V, M) where $V = \mathrm{var}(L)$, $\mathrm{set}(M) = \{\theta \mid \theta(L) \in \mathrm{atoms}(\Pi, D) \text{ and } \mathrm{dom}(\theta) = V\}$, and $\mathrm{card}(\theta, M) = \mathrm{card}(\theta(L), \mathrm{atoms}(\Pi, D))$.

Observe that the domain of the solutions of a query (L, Π) is $\mathrm{var}(L)$. Abusing notation, we will say that it is also the domain of the query (L, Π), denoted $\mathrm{dom}((L, \Pi)) = \mathrm{var}(L)$.

The Multiset Datalog query language presented here, NRMD$^\neg$, differs from the version proposed by Bertossi et al. [41] in that we do not allow recursive programs nor constants in the head of rules. These restrictions permit to match the expressive power of the SPARQL fragment studied here.

4.3. Normalization of NRMD$^\neg$ programs

To simplify the translation from NRMD$^\neg$ to SPARQL and *MRA*, we assume that every NRMD$^\neg$ query is *normalized* into a query that contains only rules of the three following types:

$L_0 \leftarrow L_1,$	where $\mathrm{var}(L_0) \subseteq \mathrm{var}(L_1)$;	(projection rule)
$L_0 \leftarrow L_1, L_2,$	where $\mathrm{var}(L_0) = \mathrm{var}(L_1) \cup \mathrm{var}(L_2)$;	(join rule)
$L_0 \leftarrow L_1, \neg L_2,$	where $\mathrm{var}(L_2) = \mathrm{var}(L_1)$ and $\mathrm{var}(L_0) = \mathrm{var}(L_1)$.	(negation rule)

We next show the feasibility of this normalization.

Lemma 2. *Every NRMD$^\neg$ query is equivalent to a normalized NRMD$^\neg$ query.*

Proof. We provide a normalization algorithm that replaces every rule in the query by a set of rules that do not change the semantics of the query. Given a NRMD$^\neg$ query (L, Π), every rule $R \in \Pi$ has the form

$$p(\bar{X}) \leftarrow A_1, \ldots, A_m, \neg B_1, \ldots, \neg B_n,$$

where $A_1, \ldots, A_m$ are positive literals, and $\neg B_1, \ldots, \neg B_n$ are negative literals. For $1 \leqslant i \leqslant m$, let $\bar{Y}_i$ be the set of variables that consists of the variables occurring in the atoms $A_1, \ldots, A_i$. Then, we replace rule R by the minimal set of rules Π_R that includes the following rules:

1. Rules R_i^A, for $2 \leqslant i \leqslant m$, defined recursively as follows:
   (a) $R_2^A = q_2^A(\bar{Y}_2) \leftarrow A_1, A_2.$
   (b) $R_i^A = q_i^A(\bar{Y}_i) \leftarrow q_{i-1}^A(\bar{Y}_{i-1}), A_i.$
2. Rules R_j^B for $1 \leqslant j \leqslant n$, defined recursively as follows:
   (a) $R_0^B = r_0^B(\bar{Y}_m) \leftarrow q_m^A(\bar{Y}_m),$
   (b) $R_j^B = r_j^B(\bar{Y}_m) \leftarrow r_{j-1}^B(\bar{Y}_m), \neg B_j,$
3. A rule $R' = p(\bar{X}) \leftarrow r_n^B(\bar{Y}_m).$

Let (L, Π') be the query resulting from replacing rule R with the rules in Π_R. It is clear that the program is normal (recall that the original program is safe). Need to show that both programs are equivalent, that is, that the answers of query $(p(\bar{X}), \Pi)$ after the replacement are the same and have the same multiplicities. These two conditions follow from Claim 3 in the Appendix. □

5. Multiset Relational Algebra (MRA)

The multiset relational algebra used in this paper is based on the semantics defined by Dayal et al. [10]. This algebra considers the operations of *selection*, *projection*, *natural join* and *arithmetic union*. Additionally, we include operators for *renaming* and *filter difference* (or "except").

5.1. Multiset relations

Assume that $\mathbf{N}, \mathbf{A}, \mathbf{C}$ are disjoint infinite sets, where $\mathbf{N}$ is the domain of relation names, $\mathbf{A}$ is the domain of *attributes*, and $\mathbf{C}$ is the domain of *constants* or *values*.

A *relation schema* is given by a relation name $R \in \mathbf{N}$ and a set of attributes $\{A_1, \ldots, A_n\}$ where $A_i \in \mathbf{A}$ for $1 \leqslant i \leqslant n$. To simplify the notation, we will use the relation name R to denote the relation schema, and $\widehat{R}$ to denote the attributes of R. A *relational database schema* is a finite set of relation schemas.

A *tuple* over a relation schema $R = \{A_1, \ldots, A_n\}$ is a total mapping t from $\widehat{R}$ to $\mathbf{C}$. The value of tuple t on an attribute $A_i \in \widehat{R}$ will be denoted as $t(A_i)$. Given a set of attributes $U \subseteq \widehat{R}$ and a tuple t, we write $t[U]$ to denote the tuple t' with attributes U such that $t'(A) = t(A)$ for every attribute $A \in U$.

A *multiset relation* r over a relation schema R is a multiset of tuples over $\widehat{R}$. We write $\hat{r}$ to denote the relation schema R where the multiset relation r is defined. Given a tuple $t \in r$, we will use $\mathrm{card}(t, r)$ to denote the cardinality of tuple t in r.

A *relational database schema* is a set of relational schemas. Given a relational database schema $T = \{R_1, \ldots, R_n\}$, a *multiset relational database* over T is a set of multiset relations $\{r_1, \ldots, r_n\}$ where each relation r_i satisfies the schema R_i.

Let r_1, r_2 be two multiset relations, and $t_1 \in r_1$ and $t_2 \in r_2$ be tuples. We say that t_1 and t_2 are compatible, denoted $t_1 \sim t_2$, if (i) for every attribute $A \in \hat{r}_1 \cap \hat{r}_2$ it holds that $t_1(A) = t_2(A)$, or (ii) $\hat{r}_1 \cap \hat{r}_2 = \emptyset$. If t_1 and t_2 are compatible, then the merge of them, denoted $t_1 \cup t_2$, is the tuple t with attributes $\hat{r}_1 \cup \hat{r}_2$ where $t(A) = t_1(A)$ for each attribute $A \in \hat{r}_1$, and $t(B) = t_2(B)$ for each attribute $B \in \hat{r}_2 \setminus \hat{r}_1$.

5.2. Syntax of MRA

The multiset relational algebra defined in this paper includes the operators of selection (σ), projection (π), renaming (ρ), join ($\bowtie$), union ($\cup$), and except ($\setminus$). Next we describe the syntax of MRA expressions containing the above operators.

A *selection formula* ψ is a Boolean combination of equality expressions of the form $x = y$ where $x, y \in \mathbf{A} \cup \mathbf{C}$. We define a *MRA expression* E over a relational database schema T, and the attributes of E, denoted $\widehat{E}$, by mutual recursion as follows:

- A relation name $R \in T$ is a MRA expression E, and $\widehat{E} = \widehat{R}$.
- If E_1 is a MRA expression and ψ is a selection formula where the attributes occurring in ψ are included in $\widehat{E}_1$, then $\sigma_\psi(E_1)$ is a MRA expression E, and $\widehat{E} = \widehat{E}_1$.
- If E_1 is a MRA expression and $S \subseteq \widehat{E}_1$ is a set of attributes, them $\pi_S(E_1)$ is a MRA expression E, and $\widehat{E} = S$.
- If E_1 is a MRA expression and $A, B \in \mathbf{A}$ are attributes, then $\rho_{A/B}(E_1)$ is a MRA expression E, and $\widehat{E} = (\widehat{E}_1 \setminus A) \cup B$.
- If E_1 and E_2 are MRA expressions, then $(E_1 \bowtie E_2)$ is an MRA expression E, and $\widehat{E} = \widehat{E}_1 \cup \widehat{E}_2$.
- If E_1 and E_2 are MRA expressions and $\widehat{E}_1 = \widehat{E}_2$, then $(E_1 \setminus E_2)$ is a MRA expression E, and $\widehat{E} = \widehat{E}_1$.
- If E_1 and E_2 are MRA expressions and $\widehat{E}_1 = \widehat{E}_2$, then $(E_1 \cup E_2)$ is a MRA expression E, and $\widehat{E} = \widehat{E}_1$.

Note that a selection operation $\sigma_\psi(E_1)$ requires that attributes in the selection formula ψ be attributes of the MRA expression E_1; the projection operation $\pi_S(E_1)$ requires that S be a subset of the attributes of the MRA expression E_1; and that the union $E_1 \cup E_2$ and difference $E_1 \setminus E_2$ expressions require that expressions E_1 and E_2 have the same set of attributes.

5.3. Semantics of MRA

Given a selection formula ψ and a tuple t over a relation schema R, we will use $t \models \psi$ to denote that t satisfies ψ, and its evaluation is given as follows:

1. if ψ is $A = B$ where $A, B \in \widehat{R}$ are attributes, then $t \models \psi$ iff $t(A) = t(B)$;
2. if ψ is $A = c$ where $A \in \widehat{R}$ is an attribute and $c \in \mathbf{C}$ is a constant, then $t \models \psi$ iff $t(A) = c$;

3. if ψ is $c_1 = c_2$ where $c_1, c_2 \in \mathbf{C}$ are constants, then $t \models \psi$ iff c_1 is equal to c_2;
4. if ψ is $\psi_1 \wedge \psi_2$, then $t \models \psi$ iff $t \models \psi_1$ and $t \models \psi_2$;
5. if ψ is $\psi_1 \vee \psi_2$, then $t \models \psi$ iff $t \models \psi_1$ or $t \models \psi_2$;
6. if ψ is $\neg\psi_1$, then $t \models \psi$ iff $t \models \psi_1$ does not hold.

Now, the evaluation of a MRA expression E over a multiset relational database D is defined as a function $\mathrm{Eval}(E, D)$ which returns a multiset relation r holding $\hat{r} = \widehat{E}$.

Let D be a MRA database over a schema T and E, E_1, E_2 be MRA expressions. The evaluation of $\mathrm{Eval}(E, D)$ is the multiset relation r defined recursively as follows (assume that $\mathrm{Eval}(E_1, D) = r_1$, and $\mathrm{Eval}(E_2, D) = r_2$):

- If E is a relation name $R_1 \in T$, then r is the relation for the relation name R_1 in the database D.
- If E is $\sigma_\psi(E_1)$ then $\mathrm{set}(r) = \{t \mid t \in r_1 \text{ and } t \models \psi\}$ and $\mathrm{card}(t, r) = \mathrm{card}(t, r_1)$.
- If E is $\pi_S(E_1)$ then $\mathrm{set}(r) = \{t' \mid t' = t[S] \text{ and } t \in r_1\}$ and $\mathrm{card}(t', r) = \sum_{t \text{ with } t[S]=t'} \mathrm{card}(t, r_1)$.
- If E is $\rho_{A/B}(E_1)$ then r is the result from renaming in r_1 the attribute A as B.
- If E is $(E_1 \bowtie E_2)$ then $\mathrm{set}(r) = \{t_1 \cup t_2 \mid t_1 \in r_1,\ t_2 \in r_2,\ \text{and } t_1 \sim t_2\}$ and $\mathrm{card}(t_1 \cup t_2, r) = \mathrm{card}(t_1, r_1) \times \mathrm{card}(t_2, r_2)$.
- If E is $(E_1 \cup E_2)$ then $\mathrm{set}(r) = \{t \mid t \in r_1 \text{ or } t \in r_2\}$ and $\mathrm{card}(t, r) = \mathrm{card}(t, r_1) + \mathrm{card}(t, r_2)$.
- If E is $(E_1 \setminus E_2)$ then $\mathrm{set}(r) = \{t \mid t \in r_1 \text{ and } t \notin r_2\}$ and $\mathrm{card}(t, r) = \mathrm{card}(t, r_1)$.

Hence, in MRA, the set of queries is the set of MRA expressions, the set of databases is the set of multiset relational databases, the set of results is the set of multiset relations, and the evaluation procedure is the aforementioned function Eval.

6. SPARQL $\equiv$ NRMD$^\neg$

This section presents the proof that SPARQL and Non-Recursive Multiset Datalog with Safe Negation (NRMD$^\neg$) have the same expressive power. The proof reduces to prove there is a simulation of SPARQL in NRMD$^\neg$ (Section 6.1) and of NRMD$^\neg$ in SPARQL (Section 6.2).

6.1. SPARQL to NRMD$^\neg$

This section describes—according to the notation in Figure 2—the simulation $T_{1,2}$, which includes the following translation functions:

- function $f_{1,2}$ which translates a SPARQL query into a NRMD$^\neg$ query;
- function $g_{1,2}$ which translates a SPARQL database into a NRMD$^\neg$ database; and
- function $h_{1,2}$ which translates a NRMD$^\neg$ query solution into a SPARQL query solution.

6.1.1. SPARQL database to NRMD$^\neg$ database

The basic idea is to translate each RDF triple into a Datalog atom. Additionally, we create an atom to encode all RDF terms, and an atom to encode the unbound value.

Definition 4 (Function $g_{1,2}$)**.** *Let τ be the vocabulary with predicate names* term, eq, triple, *and* null *with arities $\alpha(\mathrm{term}) = 1$, $\alpha(\mathrm{eq}) = 2$, $\alpha(\mathrm{triple}) = 3$, and $\alpha(\mathrm{null}) = 1$. Given an RDF graph G, the function $g_{1,2}(G)$ returns a NRMD$^\neg$ database D wich consists of the facts over the vocabulary τ defined according to the following rules:*

- $\mathrm{term}(t) \in D$ *and* $\mathrm{eq}(t, t) \in D$ *for each term $t \in G$;*
- $\mathrm{triple}(v_1, v_2, v_3) \in D$ *for each triple $(v_1, v_2, v_3) \in G$;*
- $\mathrm{null}(\bot) \in D$*, where $\bot$ is the constant reserved in RDF to encode unbounded values.*

6.1.2. SPARQL query to NRMD$^\neg$ query

In general terms, any SPARQL graph pattern can be translated into a set of NRMD$^\neg$ rules. However, there are some subtleties that need to be discussed before presenting the general translation rules.

An initial issue is the translation of a filter graph pattern $P = (P_1 \text{ FILTER } \varphi)$ where φ is a complex filter condition. In order to simplify the translation to Datalog, we need to transform P into a collection of filter graph patterns where every filter condition is an atomic filter condition.

Consider the following equivalences:

$$(P_1 \text{ FILTER } \varphi_1 \wedge \varphi_2) \equiv ((P_1 \text{ FILTER } \psi_1) \text{ FILTER } \varphi_2). \quad (1)$$

$$(P_1 \text{ FILTER } \varphi_1 \vee \varphi_2) \equiv (P_1 \text{ FILTER } \varphi_1) \text{ UNION} (P_1 \text{ FILTER } \varphi_2). \quad (2)$$

$$(P_1 \text{ FILTER } \neg(\varphi_1)) \equiv (P_1 \text{ EXCEPT}(P_1 \text{ FILTER } \varphi_1)). \quad (3)$$

Intuitively, these equivalences seem to be true, since similar equivalences are valid in set relational algebra, namely $\sigma_{\varphi_1 \wedge \varphi_2}(R) = \sigma_{\varphi_1}(\sigma_{\varphi_2}(R))$, $\sigma_{\varphi_1 \vee \varphi_2}(R) = \sigma_{\varphi_1}(R) \cup \sigma_{\varphi_2}(R)$, and $\sigma_{\neg\varphi_1}(R) = R \setminus \sigma_{\varphi_1}(R)$. Under set semantics, these three equivalences are valid. However, under bag semantics, just equivalence (1) is valid, and equivalences (2) and (3) present problems. Let us analyze them and provide valid equivalences.

- To see why equivalence (2) is not valid, consider the case where for a solution μ of the pattern P_1 the evaluation of formulas φ_1 and φ_2 are true. Then, μ is a solution of the queries in both sides of equivalence (2). However, the multiplicity differs. Indeed, the multiplicity of μ for the query on the right side is twice the multiplicity for the query on the left side. Hence, equivalence (2) is valid for set semantics but not for bag semantics.
- To see why equivalence (3) is not valid, consider the case where for a solution μ of the pattern P_1 formula φ_1 produces error. Then, formula $\neg\varphi_1$ also produces error. Thus, μ is not a solution to the query on the right side. On the other hand, since μ is a solution of P_1 but not a solution to query $(P_1 \text{ FILTER } \varphi_1)$, μ is a solution to the query on the right side. Hence, this equivalency is not valid because the queries do not have the same answers.

Intuitively, equivalence (2) is no longer valid when we change from set semantics to bag semantics, whereas equivalence (3) is no longer valid when we change from 2-valued logic to 3-valued logic. In the following, we show how to solve these problems.

Lemma 3 (Rewriting of disjoint filter conditions)**.** *We say that two filter conditions φ_1 and φ_2 are* disjoint*, if for every mapping μ it does not hold that $\mu(\varphi_1)$ and $\mu(\varphi_2)$ are simultaneously* true*. Equivalence (2) is true when φ_1 and φ_2 are disjoint.*

Proof. Given that φ_1 and φ_2 are disjoint, it applies that $\mu(\varphi_1)$ is *true* when $\mu(\varphi_2)$ is not *true* (and vice versa). So, it holds that $\mu(\varphi_1 \vee \varphi_2) =$ *true* if and only if $\mu(\varphi_1) =$ *true* or $\mu(\varphi_2) =$ *true*, and the multiplicity of μ on the left hand side is the sum of the multiplicities of μ in each of the terms of the right hand side. □

Now, consider the following equivalence:

$$\begin{aligned}(P_1 \text{ FILTER } \varphi_1 \vee \varphi_2) \equiv &(P_1 \text{ FILTER } \varphi_1 \wedge \neg\varphi_2) \text{ UNION} \\ &(P_1 \text{ FILTER } \neg\varphi_1 \wedge \varphi_2) \text{ UNION} \\ &(P_1 \text{ FILTER } \varphi_1 \wedge \varphi_2).\end{aligned} \quad (4)$$

Equivalence (4) solves one of the problems of equivalence (2), but it still has problems to evaluate formulas with errors. In order to solve them, we introduce the notion of "error filter condition."

Definition 5 (Error filter condition)**.** *Given a filter condition φ, the expression* $\text{Error}(\varphi)$ *denotes the filter condition defined recursively as follows:*

$$\text{Error}(\text{bound}(?X)) = \text{false},$$

$$\text{Error}(?X = a) = \neg\,\text{bound}(?X),$$

$$\begin{aligned}
\mathrm{Error}(?X = ?Y) &= (\neg\,\mathrm{bound}(?X) \wedge \mathrm{bound}(?Y)) \vee \\
&\quad (\mathrm{bound}(?X) \wedge \neg\,\mathrm{bound}(?Y)) \vee \\
&\quad (\neg\,\mathrm{bound}(?X) \wedge \neg\,\mathrm{bound}(?Y)),
\end{aligned}$$

$$\begin{aligned}
\mathrm{Error}(\varphi_1 \wedge \varphi_2) &= (\varphi_1 \wedge \mathrm{Error}(\varphi_2)) \vee \\
&\quad (\mathrm{Error}(\varphi_1) \wedge \varphi_2) \vee \\
&\quad (\mathrm{Error}(\varphi_1) \wedge \mathrm{Error}(\varphi_2)),
\end{aligned}$$

$$\begin{aligned}
\mathrm{Error}(\varphi_1 \vee \varphi_2) &= (\neg\varphi_1 \wedge \mathrm{Error}(\varphi_2)) \vee \\
&\quad (\mathrm{Error}(\varphi_1) \wedge \neg\varphi_2) \vee \\
&\quad (\mathrm{Error}(\varphi_1) \wedge \mathrm{Error}(\varphi_2)),
\end{aligned}$$

$$\mathrm{Error}(\neg\varphi_1) = \mathrm{Error}(\varphi_1).$$

Lemma 4. *For every filter condition φ and mapping μ it holds that $\mu(\varphi) =$ error if and only if $\mu(\mathrm{Error}(\varphi)) =$ true.*

Proof. This lemma is proved by induction on the structure of the filter condition (see Claim 1 in the appendix). □

Example 3. *Let φ be the filter condition $L \vee \neg L$ where L is the equality $?X = a$. According to Definition 5,* $\mathrm{Error}(\varphi)$ *will be the filter condition* $(\neg L \wedge \mathrm{Error}(\neg L)) \vee (\mathrm{Error}(L) \wedge \neg\neg L) \vee (\mathrm{Error}(L) \wedge \mathrm{Error}(\neg L))$. *Since $\neg\neg L$ is equivalent to L and* $\mathrm{Error}(\neg L)$ *is equivalent to* $\mathrm{Error}(L)$*, then* $\mathrm{Error}(\varphi)$ *is equivalent to* $(\neg L \wedge \mathrm{Error}(L)) \vee (\mathrm{Error}(L) \wedge L) \vee (\mathrm{Error}(L))$*, which is equivalent to* $(\varphi \wedge \mathrm{Error}(L)) \vee (\mathrm{Error}(L))$*, and then, equivalent to* $\mathrm{Error}(L)$*. According to Definition 5, we conclude that* $\mathrm{Error}(\varphi)$ *is equivalent to* $\neg\,\mathrm{bound}(?X)$.

There are three possible values for variable $?X$ in a mapping μ, namely $\mu(?X) = a$, $\mu(?X) = b$ (for a term $b \neq a$), and variable $?X$ is unbound in μ (denoted $\mu(?X) = \perp$). The following table shows the values for $\mu(\varphi)$ and $\mu(\mathrm{Error}(\varphi))$ for these three cases.

$\mu(?X)$	$\mu(\varphi)$	$\mu(\mathrm{Error}(\varphi))$
a	true	false
b	true	false
$\perp$	error	true

As defined by Lemma 4, the filter condition φ produces error for mappings μ where $\mu(\mathrm{Error}(\varphi)) =$ true*, and $\mu(\mathrm{Error}(\varphi))$ is either* true *or* false.

Now we present an equivalence for the disjunction that works in all cases.

Lemma 5 (Disjunction rewriting)**.** *Given two filter conditions φ_1 and φ_2, and a pattern P, the following equivalence holds for bag semantics:*

$$\begin{aligned}
(P\,\mathrm{FILTER}\,\varphi_1 \vee \varphi_2) \equiv\ & (P\,\mathrm{FILTER}\,\varphi_1 \wedge \varphi_2)\,\mathrm{UNION} \\
& (P\,\mathrm{FILTER}\,\varphi_1 \wedge \neg\varphi_2)\,\mathrm{UNION} \\
& (P\,\mathrm{FILTER}\,\neg\varphi_1 \wedge \varphi_2)\,\mathrm{UNION} \\
& (P\,\mathrm{FILTER}\,\varphi_1 \wedge \mathrm{Error}(\varphi_2))\,\mathrm{UNION} \\
& (P\,\mathrm{FILTER}\,\mathrm{Error}(\varphi_1) \wedge \varphi_2).
\end{aligned} \tag{5}$$

Proof. Since $\varphi \vee \neg\varphi \vee \mathrm{Error}(\varphi)$ is a tautology for every filter condition φ, the following equivalences hold:

$$\varphi_1 \equiv \varphi_1 \wedge (\varphi_2 \vee \neg\varphi_2 \vee \mathrm{Error}(\varphi_2)) \equiv (\varphi_1 \wedge \varphi_2) \vee (\varphi_1 \wedge \neg\varphi_2) \vee (\varphi_1 \wedge \mathrm{Error}(\varphi_2)),$$

$$\varphi_2 \equiv \varphi_2 \wedge (\varphi_1 \vee \neg\varphi_1 \vee \mathrm{Error}(\varphi_1)) \equiv (\varphi_2 \wedge \varphi_1) \vee (\varphi_2 \wedge \neg\varphi_1) \vee (\varphi_2 \wedge \mathrm{Error}(\varphi_1)).$$

Hence, the following equivalence holds:

$$\varphi_1 \vee \varphi_2 \equiv (\varphi_1 \wedge \varphi_2) \vee (\varphi_1 \wedge \neg\varphi_2) \vee (\neg\varphi_1 \wedge \varphi_2) \vee (\varphi_1 \wedge \mathrm{Error}(\varphi_2)) \vee (\mathrm{Error}(\varphi_1) \wedge \varphi_2).$$

Since all filter conditions in the disjunction of the right side of this equivalence are disjoint, by Lemma 3, we got equivalence (5). □

Finally, we provide a translation for filter graph patterns which have a negation. Under two-valued logic, the evaluation of a pattern P of the form $(P_1 \,\mathrm{FILTER}\, \neg\varphi)$ may be understood as "all solutions μ of P_1 except those where $\mu(\varphi)$ is true." Under 3-valued logic, the evaluation of P means "all solutions μ of P except those where $\mu(\varphi)$ is true or $\mu(\varphi)$ is error." Thus according to the latter meaning we have:

Lemma 6 (Negation rewriting)**.** *Given a filter condition φ, and a pattern P_1, the following equivalence holds:*

$$(P_1 \,\mathrm{FILTER}\, \neg\varphi) \equiv\ ((P_1 \,\mathrm{EXCEPT}\, (P_1 \,\mathrm{FILTER}\, \varphi)) \,\mathrm{EXCEPT}\, (P_1 \,\mathrm{FILTER}\, \mathrm{Error}(\varphi))). \tag{6}$$

Proof. The equivalence follows from the fact that the filter discards from the answers of P those answers μ such that $\mu(\varphi)$ is either *false* or *error*. □

Now we are ready to present the effectiveness of rewriting that allows for the reduction of complex filter conditions.

Definition 6 (Reduction of complex filter conditions)**.** *Given a pattern $(P_1 \,\mathrm{FILTER}\, \varphi)$, the filter-reduced pattern of it is the pattern that results of applying recursively the equivalences (1), (5), and (6) until in the resulting patterns only occur atomic formulas (i.e. no logical connectives).*

Lemma 7. *Given a pattern $(P_1 \,\mathrm{FILTER}\, \varphi)$, the procedure to reduce complex filter conditions described in Definition 6 produces a pattern equivalent to the original and with no logical connectives in filter conditions.*

Proof. This lemma is proved by induction on the structure of the filter condition in the pattern. The base case consists in a filter condition φ without logical connectives. The case where φ is $\varphi_1 \wedge \varphi_2$ is straightforward. The pattern $(P \,\mathrm{FILTER}\, \varphi)$ can be reduced to the pattern $((P \,\mathrm{FILTER}\, \varphi_1) \,\mathrm{FILTER}\, \varphi_2)$, and the inductive hypothesis can be applied on φ_1 and φ_2. The cases where φ is $\varphi_1 \vee \varphi_2$ or $\neg\varphi_1$ are more involved because the application of the respective equivalences eliminates a logical connective from φ but adds new logical connectives to the resulting filter conditions. The proof for the cases involving disjunction or negation follows from Claim 2 in the appendix. □

We are ready to present the translation from SPARQL patterns to Datalog queries. The translation follows essentially the idea presented by Polleres [34], adapted to multisets by Angles and Gutierrez [36], and improved by Hernández [30]. Specifically, we cover the following issues:

1. It considers the cases where a filter condition is evaluated as error. Some solutions are lost when these cases are not considered.
2. It considers that the equality $X = Y$ must be evaluated as true only if X and Y are bound. The translation is fixed by using the literal $\mathrm{eq}(X, Y)$ instead of a built-in equality $X = Y$. Since, atom $\mathrm{eq}(X, Y)$ is true only if X and Y are terms in the database, the translation of the filter-condition $X = Y$ is not evaluated as true when X and Y are unbound.

Let the function δ, given by the translation rules presented in Table 5, transforms a SPARQL graph pattern P into a set of Datalog rules $\delta(P)$. Note that function δ assigns a fresh predicate name to each pattern P and subpattern P_i of P in a non-deterministic way. Polleres [34] proposed a deterministic recursive method to assign predicate names to patterns. The function δ is the basis to present a general method to transform SPARQL queries into NRMD$^\neg$ queries.

Definition 7 (Function $f_{1,2}$)**.** *Given a SPARQL query P, the function $f_{1,2}(P)$ returns a NRMD$^\neg$ query (L, Π) where L is the goal atom $p(\bar{P})$ and Π is a Datalog program containing the rules produced by $\delta(P)$.*

Table 4

Definition of function δ which allows to translate a SPARQL graph pattern into a set of NRMD$^\neg$ rules. Given a graph pattern P, the function $\delta(P)$ returns a set of Datalog rules. $\bar{P}$ denotes a set of variables in lexicographical order. p_i is a fresh predicate name used to codify the graph pattern P_i.

Graph Pattern P	$\delta(P)$	where ...
(x_1, x_2, x_3)	$p(\bar{P}) \leftarrow \mathit{triple}(x_1, x_2, x_3)$	$\bar{P}$ contains the variables in $\{x_1, x_2, x_3\}$
$(P_1 \text{ AND } P_2)$	$p(\bar{P}) \leftarrow \nu_1(p_1(\bar{P}_1)), \nu_2(p_2(\bar{P}_2)), \{\text{comp}(\nu_1(X), \nu_2(X), X) \mid X \in \bar{P}_1 \cap \bar{P}_2\}$; $\text{comp}(X,X,X) \leftarrow \text{term}(X)$; $\text{comp}(X,Y,X) \leftarrow \text{term}(X), \text{null}(Y)$; $\text{comp}(X,Y,Y) \leftarrow \text{null}(X), \text{term}(Y)$; $\text{comp}(X,X,X) \leftarrow \text{null}(X)$; $\delta(P_1)$; $\delta(P_2)$	ν_1 and ν_2 are functions whose domain is $\bar{P}_1 \cap \bar{P}_2$, have disjoint range, and $\nu_i(L)$ denotes a copy of a literal L where its variables have been renamed according to function ν_i.
$(P_1 \text{ UNION } P_2)$	$p(\bar{P}) \leftarrow p_1(\bar{P}_1)$; $p(\bar{P}) \leftarrow p_2(\bar{P}_2)$; $\delta(P_1)$; $\delta(P_2)$	$\bar{P} = \bar{P}_1 = \bar{P}_2$
$(P_1 \text{ EXCEPT } P_2)$	$p(\bar{P}) \leftarrow p_1(\bar{P}_1), \neg p_2(\bar{P}_2)$; $\delta(P_1)$; $\delta(P_2)$	$\bar{P} = \bar{P}_1$
$(P_1 \text{ FILTER } x_1 = x_2)$	$p(\bar{P}) \leftarrow p_1(\bar{P}_1), \text{eq}(x_1, x_2)$; $\delta(P_1)$	$\bar{P} = \bar{P}_1$
$(P_1 \text{ FILTER bound}(?X))$	$p(\bar{P}) \leftarrow p_1(\bar{P}_1), \text{term}(?X)$; $\delta(P_1)$;	$\bar{P} = \bar{P}_1$
$(\text{SELECT } W\ P_1)$	$p(\bar{P}) \leftarrow p_1(\bar{P}_1), \text{null}(x_1), \ldots, \text{null}(x_n)$; $\delta(P_1)$	$\bar{P} = W$, and $x_1, \ldots, x_n$ are the variables that are in W but not in $\text{inScope}(P_1)$.

6.1.3. NRMD$^\neg$ solution to SPARQL solution

Note that main difference between a multiset of substitutions and a multiset of mappings is the representation of the SPARQL unobound values with $\perp$. Hence, a unbound value occurring in a substitution is translated into a unbound variable in the corresponding solution mapping.

Definition 8 (Function $h_{1,2}$)**.** *Given a multiset of Datalog substitutions Θ, the function $h_{1,2}(\Theta)$ returns a multiset of SPARQL solution mappings defined as*

$$\Omega = \{(\text{NotNull}(\theta), i) \mid (\theta, i) \in \Theta\},$$

where $\text{NotNull}(\theta)$ *returns a mapping μ satisfying that $\mu(X) = \theta(X)$ for every variable $X \in \text{dom}(\theta)$ such that $\theta(X) \neq \perp$.*

Lemma 8. *SPARQL can be simulated in NRMD$^\neg$.*

Proof. Need to show that, using the functions defined above, $(f_{1,2}, g_{1,2}, h_{1,2})$ is a simulation of SPARQL in NRMD$^\neg$. The proof is in the Claim 4 of the Appendix. □

6.2. Multiset Datalog to SPARQL

This section describes—according to the notation in Figure 2—the simulation $T_{2,1}$, which includes the following translation functions:

- function $f_{2,1}$ which translates a NRMD$^\neg$ query into a SPARQL query;
- function $g_{2,1}$ which translates a NRMD$^\neg$ database into a SPARQL database; and
- function $h_{2,1}$ which translates a SPARQL query solution into a NRMD$^\neg$ query solution.

6.2.1. NRMD$^\neg$ database to SPARQL database

In general terms, a fact $p(c_1,\ldots,c_n)$ can be translated into a set of triples of the form (u,α_i,c_i) where u is a fresh IRI that identifies the fact, and α_i is a reserved IRI which allows to describe that constant c_i is in the position i of the fact[2]. This idea is formalized next.

Assume that $A=\{\alpha_0,\alpha_1\ldots\}$ is an enumerable set of special IRIs used to codify positions in Datalog atoms, `NULL` is a special IRI, and any Datalog constant c has an equivalent SPARQL term (excluding the aforementioned especial IRIs).

Recall that the semantics of NRMD$^\neg$ relies on the notion of colored set of a multiset (see Section 4.2), which is the set containing the colored copies of the element of the multiset. If the NRMD$^\neg$ database D contains n copies of a fact F, then $\mathrm{coloring}(D)$ contains the colored copies $\langle F,1\rangle,\ldots,\langle F,n\rangle$ of fact F. For each colored copy $\langle F,i\rangle$ of F, we assume the existence of a fresh IRI $u_{\langle F,i\rangle}$, which we use to identify the colored copy.

Definition 9 (Function $g_{2,1}$)**.** *Given a multiset of NRMD$^\neg$ facts D, the function $g_{2,1}$ returns a set of RDF triples (i.e. an RDF graph) defined as*

$$g_{2,1}(D)=\{\mathtt{NULL},\mathtt{NULL},\mathtt{NULL}\}\bigcup_{\langle F,i\rangle\in\mathrm{coloring}(D)}\{(u_{\langle F,i\rangle},\alpha_0,p),(u_{\langle F,i\rangle},\alpha_1,c_1),\ldots,(u_{\langle F,i\rangle},\alpha_n,c_n)\}.$$

Intuitively, the SPARQL database corresponding to the NRMD$^\neg$ database D consists of a set of triples that describe each of the facts, and the inclusion of triple $(\mathtt{NULL},\mathtt{NULL},\mathtt{NULL})$ allows to ensure that the SPARQL database is not empty. The need of this additional triple is explained next when describing the translation from NRMD$^\neg$ queries to SPARQL.

6.2.2. NRMD$^\neg$ query to SPARQL query

A notable difference between NRMD$^\neg$ and SPARQL is the way both languages define the scope of variables. In NRMD$^\neg$ rules, variables are universally quantified, and they are not in the scope of the query. On the other hand, variables in a SPARQL query are divided into in-scope and non-in-scope (see Subsection 3.4). To see this difference, consider the NRMD$^\neg$ query $(q(X,Y),\Pi)$ where program Π consists of the single rule $R=q(Y,Z)\leftarrow p(X,Z,Y)$. Notice that the variables in the goal of the query do not correspond to the variables in the head of the rule R. To simplify the translation, we rename variables in rules according to the goal of the query. In this case, we rewrite R as the rule $R'=q(X,Y)\leftarrow p(X,Y,Z)$. Formally, given a literal $L=q(X_1,\ldots,X_n)$ and a rule R whose head is $q(Y_1,\ldots,Y_n)$, the *renamed rule of R with respect to L*, denoted $\mathrm{vr}(R,L)$, is the rule R' that is equivalent to R but have literal L as head. Intuitively, rule R' results from consistently renaming each variable Y_i as X_i, for $1\leqslant i\leqslant n$.

Let L be a positive literal $p(X_1,\ldots,X_n)$ and Π be a normalized NRMD$^\neg$ program. We define the function $\mathrm{gp}(L,\Pi)$ which translates L into a SPARQL graph pattern. The function gp is defined recursively as follows:

1. If predicate name p does not occur in the head of any rule of Π (i.e., p is extensional), then $\mathrm{gp}(L,\Pi)$ returns

   $$\mathrm{SELECT}\,\overline{X}\,((?Y,\alpha_0,p)\,\mathrm{AND}\,(?Y,\alpha_1,?X_1)\,\mathrm{AND}\cdots\mathrm{AND}\,(?Y,\alpha_n,?X_n)),$$

   where $\overline{X}=\mathrm{var}(L)$ and $?Y$ is a fresh variable.
2. Otherwise, if p occurs in the head of the rules $\{R_1,\ldots,R_n\}$ in Π (i.e., p is intensional), then $\mathrm{gp}(L)^D_\Pi$ returns:

   $$(T(\mathrm{vr}(R_1,L))\,\mathrm{UNION}\cdots\mathrm{UNION}\,T(\mathrm{vr}(R_n,L))),$$

   where, $T(\mathrm{vr}(R_i,L))$ is defined as follows:
   - If $\mathrm{vr}(R_i,L)$ is a projection rule $L\leftarrow L_1$ then $T(R_i)$ is $(\mathrm{SELECT}\,\overline{X}\;P_1)$ where $\overline{X}=\mathrm{var}(L)$ and $P_1=\mathrm{gp}(L_1,\Pi)$;
   - If $\mathrm{vr}(R_i,L)$ is a join rule $L\leftarrow L_1,L_2$ then $T(R_i)$ is $(P_1\,\mathrm{AND}\,P_2)$ where $P_1=\mathrm{gp}(L_1,\Pi)$ and $P_2=\mathrm{gp}(L_2,\Pi)$;

[2] An option can be the use of properties `rdf:_1`, `rdf:_2`, `rdf:_3`, ..., defined in the RDF Schema 1.1 vocabulary.

– If $\mathrm{vr}(R_i, L)$ is a negation rule $L \leftarrow L_1, \neg L_2$ then $T(R_i)$ is $(P_1 \,\mathrm{EXCEPT}\, P_2)$ where $P_1 = \mathrm{gp}(L_1, \Pi)$ and $P_2 = \mathrm{gp}(L_2, \Pi)$.

Note that, if $n = 1$ then $\mathrm{gp}(L, \Pi)$ can be reduced to $T(R_1)$.

Example 4. *Consider the NRMD$^\neg$ query* $(q(X), \Pi)$ *where program* Π *consists of the rule* $q(X) \leftarrow p(X, Y)$. *Then,* $\mathrm{gp}((q(X), \Pi))$ *is the SPARQL query*

$$\mathrm{SELECT}\ ?X\ ((?U, \alpha_0, p)\ \mathrm{AND}\ (?U, \alpha_1, ?X)\ \mathrm{AND}\ (?U, \alpha_2, ?Y)).$$

The function gp is not enough to translate NRMD$^\neg$ queries to SPARQL. Recall that a NRMD$^\neg$ answer is a pair (V, M) where V is a set of NRMD$^\neg$ variables and M is a set of NRMD$^\neg$ substitutions, and a SPARQL answer is a multiset Ω of SPARQL mappings. To conclude the translation, we need to define a function that, given an SPARQL answer Ω, returns a NRMD$^\neg$ answer (V, M). The issue is that we cannot compute the set V when multiset Ω is empty. For instance, an empty NRMD$^\neg$ database D is translated as the SPARQL database consisting of the set of triples $\{(\alpha_0, \alpha_0, \alpha_0)\}$ (see Subsection 6.2.1). The answer to the query $\mathrm{gp}((q(X), \Pi))$ in Example 4 returns an empty multiset of mappings, Ω, where the answer to the NRMD$^\neg$ query $(q(X), \Pi)$ is a pair $(\{X\}, M)$ such that M is an empty multiset of solutions. Hence, the SPARQL answer Ω does not contain the needed information to generate the set of variables $\{X\}$ in the answer of the NRMD$^\neg$ query.

To solve the aforementioned issue of having an empty SPARQL answer, we can extend the function gp with a query that introduces the variables of the query. This is done using the additional triple $\{\mathtt{NULL}, \mathtt{NULL}, \mathtt{NULL}\}$ we introduced in the translation. Given a set of NRMD$^\neg$ variables $V = \{X_1, \ldots, X_n\}$ we write $\mathrm{VarQuery(V)}$ to denote the SPARQL pattern

$$(\mathtt{NULL}, \mathtt{NULL}, ?X_1)\ \ \mathrm{AND}\ \cdots\ \mathrm{AND}\ (\mathtt{NULL}, \mathtt{NULL}, ?X_n).$$

The translation of a NRMD$^\neg$ query is then the union of the graph pattern computed by functions gp and $\mathrm{VarQuery}$.

Definition 10 (Function $f_{2,1}$)**.** *Given a NRMD$^\neg$ query* $Q = (L, \Pi)$ *where L is the goal clause, and* Π *a NRMD$^\neg$ program, the function* $f_{2,1}(Q)$ *returns a SPARQL graph pattern* $(\mathrm{gp}(L, \Pi)\ \mathrm{UNION}\ \mathrm{VarQuery}(\mathrm{var}(L)))$.

Example 5. *Consider the NRMD$^\neg$ query* $(q(X), \Pi)$ *in Example 4. Then,* $f_{2,1}((q(X), \Pi)$ *is the following SPARQL graph pattern:*

$$(\mathrm{SELECT}\ ?X\ ((?U, \alpha_0, p)\ \mathrm{AND}\ (?U, \alpha_1, ?X)\ \mathrm{AND}\ (?U, \alpha_2, ?X)))\ \mathrm{UNION}\ (\mathtt{NULL}, \mathtt{NULL}, ?X).$$

The result of evaluating the NRMD$^\neg$ query on an empty set of facts D is the pair $(\{X\}, M)$ *where M is an empty multiset of NRMD$^\neg$ substitutions, whereas the result of evaluating the graph pattern* $f_{2,1}((q(X), \Pi))$ *on the SPARQL database* $g_{2,1}(D) = \{(\mathtt{NULL}, \mathtt{NULL}, \mathtt{NULL})\}$ *is the SPARQL answer* $\Omega = \{\{?X \mapsto \alpha_0\}\}$. *Intuitively, the mapping* $\{?X \mapsto \mathtt{NULL}\}$ *does not codify a NRMD$^\neg$ substitution, but the variables in the domain of NRMD$^\neg$ substitutions.*

6.2.3. SPARQL solution to NRMD$^\neg$ solution

Since a SPARQL mapping can be seen also as a NRMD$^\neg$ substitution, the translation from SPARQL mappings to NRMD$^\neg$ substitutions requires no modifications, except for the mapping $\{?X_1 \mapsto \mathtt{NULL}, \ldots, ?X_n \mapsto \mathtt{NULL}\}$ which is used to codify the solution variables.

Definition 11 (SPARQL answers as NRMD$^\neg$ answers)**.** *Let* Ω *be a multiset of SPARQL solution mappings that includes a mapping* $\mu_{V \mapsto \mathtt{NULL}}$ *with multiplicity 1 where* $\mathrm{dom}(\mu_{V \mapsto \mathtt{NULL}}) = V$ *and* $\mu(?X) = \mathtt{NULL}$ *for every variable* $?X \in \mathrm{dom}(\mu_{V \mapsto \mathtt{NULL}})$, *and for every mapping* $\mu' \in \Omega$ *it is hold that* $\mathrm{dom}(\mu') = V$. *The NRMD$^\neg$ answer for* Ω, *denoted* $h_{2,1}(\Omega)$, *is the pair* (V, M) *where M is the multiset of substitutions* θ *defined as follows:*

1. *Given an SPARQL mapping* $\mu = \{?X_1 \mapsto c_1, \ldots, ?X_n \mapsto c_n\}$ *the corresponding NRMD$^\neg$ substitution for mapping* μ *is the substitution* $\theta_\mu = \{X_1/c_1, \ldots, X_n/c_n\}$ *where the NRMD$^\neg$ variable* X_i *corresponds to the SPARQL variable* $?X_i$.

2. $\mathrm{set}(M) = \{\theta_\mu \mid \mu \in \Omega \setminus \{\mu_{V \mapsto \mathtt{NULL}}\}\}$.
3. $\mathrm{card}(\theta_\mu, M) = \mathrm{card}(\mu, \Omega)$.

Lemma 9. *NRMD$^\neg$ can be simulated in SPARQL.*

Proof. This is a long but straightforward induction on Datalog queries using as hypothesis that $(f_{2,1}, g_{2,1}, h_{2,1})$ is a simulation of NRMD$^\neg$ in SPARQL. The details of this proof are in the appendix (Claim 5). □

6.3. SPARQL and NRMD$^\neg$ have the same expressive power

Putting together the simulations between NRMD$^\neg$ and SPARQL presented in the previous sections, we can state the following theorem:

Theorem 1. *SPARQL and NRMD$^\neg$ have the same expressive power.*

Proof. If follows from Lemma 8 and Lemma 9. □

7. MRA ≡ NRMD$^\neg$

This section presents the simulations that prove that Multiset Relational Algebra (MRA) and Non-recursive Multiset Datalog with Safe Negation (NRMD$^\neg$) have the same expressive power.

7.1. MRA to Multiset NRMD$^\neg$

This section describes—according to the notation in Figure 2—the simulation T_{32} which includes the following translation functions:

- function f_{32} which translates a MRA query into a NRMD$^\neg$ query;
- function g_{32} which translates a MRA database into a NRMD$^\neg$ database; and
- function h_{32} which translates a NRMD$^\neg$ query solution into a MRA query solution.

7.1.1. MRA database to NRMD$^\neg$ database

Recall that a MRA database is a set of relations (where each relation is a multiset of tuples), and a NRMD$^\neg$ database is a set of facts. First, we define a method to translate a MRA relation into a multiset of facts. Then, we define a method to translate a set of MRA relations into a multiset of NRMD$^\neg$ facts.

Assume the existence of functions that map: MRA relation names to NRMD$^\neg$ predicate names, MRA attributes to NRMD$^\neg$ variables, and MRA constants to NRMD$^\neg$ constants. Given a relation schema R, we write $\vec{R}$ to denote a tuple containing the attributes of R in lexicographical order.

Given a multiset relation r, defined under a relation schema R, with $\vec{R} = (A_1, \ldots, A_n)$, the function $\Sigma(r)$ returns a multiset of Datalog facts defined as follows: For each tuple t in r, $\Sigma(r)$ contains a fact f of the form $p(c_1, \ldots, c_n)$ where p is R, every c_i is $t(A_i)$, and the multiplicity of f in $\Sigma(r)$ is given by the multiplicity of t in r.

Definition 12 (Function g_{32})**.** *Given a MRA database D, the function g_{32} returns a multiset of NRMD$^\neg$ facts D' defined as follows:*

- *For each MRA relation r in D, D' contains the facts returned by $\Sigma(r)$;*
- *For each constant c in D, D' contains a fact* $\mathrm{eq}(c, c)$.

Note that the multiset of Datalog facts D' is defined over the vocabulary that includes as predicate names all the relation names in D, and as arity of the predicate name R the number of attributes of the relation name R.

7.1.2. MRA query to NRMD$^\neg$ query

First, we need to provide a recursive method to reduce MRA selection formulas into atomic formulas. Such method is based on the following equivalences where E is an MRA expression, and ψ, ψ_1, and ψ_2 are selection formulas:

$$\sigma_{\psi_1 \wedge \psi_2}(E) \equiv \sigma_{\psi_2}(\sigma_{\psi_1}(E)), \tag{7}$$

$$\sigma_{\psi_1 \vee \psi_2}(E) \equiv \sigma_{\psi_1 \wedge \neg\psi_2}(E) \cup \sigma_{\neg\psi_1 \wedge \psi_2}(E) \cup \sigma_{\psi_1 \wedge \psi_2}(E), \tag{8}$$

$$\sigma_{\neg\psi}(E) \equiv E \setminus \sigma_{\psi}(E). \tag{9}$$

The proof of the validity of the above equivalences follows directly from the semantics of the selection operator. In particular, Equivalence 8 is rather involved because separates the disjunction in a union of three disjoint multiset relations in order to preserve the multiplicity of each solution. Using the above equivalence, we get the following lemma.

Lemma 10. *For every MRA expression E, there exists an equivalent MRA expression E' satisfying that all selection formulas in E' are atomic.*

Proof. The proof follows from induction in the number k of Boolean connectives in selection formulas occurring in an MRA expression E. The base case is $k = 0$ and thus all selection formulas are atomic. If $k > 0$, then the expression includes a selection expression whose formula has either the form $\psi_1 \wedge \psi_2$, $\neg\psi$, or $\psi_1 \vee \psi_2$. In the first two cases, equivalences 7 and 9 reduce by one of the Boolean connectives of the expression. In the third case, the consecutive application of equivalences 8, 7, and 9 (in that order) reduces by one the number of Boolean connectives. Hence, we produce an equivalent query with $k - 1$ Boolean connectives. □

Definition 13 (Function f_{32})**.** *Let Q be a MRA query (i.e. an MRA expression), where selection formulas are atomic (i.e., have no Boolean connectives). The function $f_{32}(Q)$ returns a NRMD$^\neg$ query (L, Π) where L is a goal clause of the form $q(\vec{Q})$ where q is a predicate name corresponding to Q, $\vec{Q}$ are the variables corresponding to the attributes in the schema $\widehat{Q}$ (sorted in lexicographical order), and Π is a set of NRMD$^\neg$ rules (i.e. a NRMD$^\neg$ program) created by applying recursively the rules shown in Table 5.*

Note that function f_{32} assigns a fresh predicate name to each operation in query Q by following a non-deterministic approach. (Although it is not difficult to define deterministic ways; like in the translation from *SPARQL* to NRMD$^\neg$, in this paper, we omit in how intensional predicate names are assigned).

Table 5

Definition of function Γ which translates an MRA expression into a set of Datalog rules. Given a MRA expression E, the recursive function $\Gamma(E)$ returns a set of NRMD$^\neg$ rules where: $q_i(\bar{A})$ is a positive literal related to the MRA expression E_i, q_i is a fresh predicate name, $\bar{A}_i$ denotes a set of variables, $\vec{R}$ denotes the attributes in schema $\widehat{R}$, sorted in lexicographical order.

MRA expression E_0	$\Gamma(E_0)$	where ...
R	$q_0(\bar{A}_0) \leftarrow R(\vec{R})$	$\bar{A}_0 = \vec{R}$
$(E_1 \bowtie E_2)$	$q_0(\bar{A}_0) \leftarrow q_1(\bar{A}_1), q_2(\bar{A}_2); \Gamma(E_1); \Gamma(E_2)$	$\bar{A}_0 = \bar{A}_1 \cup \bar{A}_2$
$(E_1 \cup E_2)$	$q_0(\bar{A}_0) \leftarrow q_1(\bar{A}_1); q_0(\bar{A}_0) \leftarrow q_2(\bar{A}_2); \Gamma(E_1); \Gamma(E_2)$	$\bar{A}_0 = \bar{A}_1 = \bar{A}_2$
$(E_1 \setminus E_2)$	$q_0(\bar{A}_0) \leftarrow q_1(\bar{A}_1), \neg q_2(\bar{A}_2); \Gamma(E_1); \Gamma(E_2)$	$\bar{A}_0 = \bar{A}_1$
$\pi_S(E_1)$	$q_0(\bar{A}_0) \leftarrow q_1(\bar{A}_1); \Gamma(E_1)$	$\bar{A}_0 = S$
$\rho_{A/B}(E_1)$	$q_0(\bar{A}_0) \leftarrow q_1(\bar{A}_1), \mathrm{eq}(A, B); \Gamma(E_1)$	$\bar{A}_0 = (\bar{A}_1 \setminus \{A\}) \cup \{B\}$
$\sigma_{A=B}(E_1)$	$q_0(\bar{A}_0) \leftarrow q_1(\bar{A}_1), \mathrm{eq}(A, B); \Gamma(E_1)$	$\bar{A}_0 = \bar{A}_1$

7.1.3. NRMD$^\neg$ solution to MRA solution

Recall that an NRMD$^\neg$ answer is a pair (V, M) where V is a set of variables (i.e. the domain of the solutions), and M is a multiset of NRMD$^\neg$ substitutions. On the other hand, an MRA answer is a multiset relation. Next, we define a function h_{32} which transforms an NRMD$^\neg$ answer into an MRA answer.

Definition 14 (Function h_{32})**.** *Given a NRMD$^\neg$ answer $A = (V, M)$, the function $h_{32}(A)$ returns a multiset relation R where: the schema of R is given by the set of attributes V (assume a simple transformation of variables to attribute names); for each substitution θ in M, there is a tuple t in R satisfying that $t(X) = \theta(X)$ for every attribute $X \in \widehat{R}$, and* $\mathrm{card}(t, R) = \mathrm{card}(\theta, M)$.

Lemma 11. *MRA can be simulated in NRMD$^\neg$.*

Proof. Let f_{32}, g_{32}, h_{32} denote respectively the functions stated in definitions 12, 14 and 13. The proof of this theorem follows from the claim that (f_{32}, g_{32}, h_{32}) simulates MRA in NRMD$^\neg$ by using induction in the structure of queries. The proof of this claim is in the appendix (Claim 6). □

7.2. NRMD$^\neg$ to MRA

This section describes—according to the notation in Figure 2—the simulation T_{23} which includes the following translation functions:

- f_{23}, which translates a NRMD$^\neg$ query into a MRA query;
- g_{23}, which translates a NRMD$^\neg$ database into a MRA database; and
- h_{23}, which translates a MRA query solution into a NRMD$^\neg$ query solution.

7.2.1. NRMD$^\neg$ database to MRA database

Recall that an NRMD$^\neg$ database is a set of facts, and an MRA database is a set of relations (where each relation is a multiset of tuples). First, we define a method to translate a multiset of facts with the same predicate name into a relation R. Let M be a multiset of NRMD$^\neg$ facts having the same predicate name, i.e. every fact in M has the form $p(t_1, \ldots, t_n)$. The function $\psi(M)$ returns a MRA relation R where: the relation schema $\widehat{R}$ of R is given by the relation name p and the set of attributes $\{A_1, \ldots, A_n\}$, where each attribute name has the form *att_i* with $1 \leqslant i \leqslant n$; for each fact $p(t_1, \ldots, t_n)$ in M there is a tuples t in R satisfying that $t(A_i) = t_i$.

Next, we define function g_{23} which allows translating a multiset of facts into a set of relations.

Definition 15 (Function g_{23})**.** *Let M be a multiset of NRMD$^\neg$ facts M (i.e. an NRMD$^\neg$ database), and $\{p_1, \ldots, p_n\}$ are the predicate names in M. The function $g_{23}(M)$ returns a set of relations (i.e. a MRA database) $\{R_1, \ldots, R_n\}$ where $R_i = \psi(M_i)$ such that M_i is the subset of NRMD$^\neg$ facts of M having the predicate name p_i.*

7.2.2. NRMD$^\neg$ query to MRA query

Let Π be a normalized NRMD$^\neg$ program. We define, by mutual recursion, functions $\delta_1(L, \Pi)$ and $\delta_2(r, \Pi)$ to translate (respectively) literals and rules into MRA expressions.

Given a literal L in Π of the form $p(X_1, \ldots, X_n)$, the function $\delta_1(L, \Pi)$ is defined as follows:

1. If predicate name p does not occur in the head of any rule of Π, then $\delta_1(L, \Pi)$ returns the MRA expression $\rho_{A_1/X_1}(\cdots \rho_{A_n/X_n}(R) \cdots)$ where R is the relation name associated to p;
2. Otherwise, if p occurs in the head of the rules $\{r_1, \ldots, r_m\}$ in Π, then $\delta_1(L, \Pi)$ returns the MRA expression $(E_1 \cup (E_2 \cup (\cdots E_m) \cdots)$ where each E_i is a MRA expression returned by $\delta_2(r_i, \Pi)$.

Given a rule r in Π, the function $\delta_2(r, \Pi)$ is defined as follows:

- If r is a projection rule $L_0 \leftarrow L_1$ then $\delta_2(r, \Pi)$ returns the MRA expression $\pi_S(E)$ where S is the set of variables $\mathrm{var}(L_0)$ and E is the MRA expression returned by $\delta_1(L_1, \Pi)$;
- If r is a join rule $L_0 \leftarrow L_1, L_2$ then $\delta_2(r, \Pi)$ returns the MRA expression $(E_1 \bowtie E_2)$ where E_1 and E_2 are the MRA expressions returned by $\delta_1(L_1, \Pi)$ and $\delta_1(L_2, \Pi)$ respectively;
- If r is a negation rule $L_0 \leftarrow L_1, \neg L_2$ then $\delta_2(r, \Pi)$ returns the MRA expression $(E_1 \setminus E_2)$ where E_1 and E_2 are the MRA expressions returned by $\delta_1(L_1, \Pi)$ and $\delta_1(L_2, \Pi)$ respectively.

Definition 16 (Function f_{23})**.** *Given a normalized NRMD$^\neg$ query $Q = (L, \Pi)$ where L is the goal clause, and Π a NRMD$^\neg$ program, the function $f_{23}(Q)$ returns a MRA query defined by $\delta_1(L, \Pi)$.*

7.2.3. MRA solution to NRMD$^\neg$ solutions

Recall that a MRA solution is a multiset relation, and a NRMD$^\neg$ solution is a pair (V, M) where V is a set of variables (i.e. the domain of the solutions), and M is a multiset of substitutions. Next, we define a function h_{23} which transforms an MRA answer into an NRMD$^\neg$ answer.

Since a MRA tuple can be seen (interpreted) also as a Datalog substitution, the translation from MRA tuples to Datalog substitutions requires essentially no modifications.

Definition 17 (Function h_{23})**.** *Given a MRA relation R with schema $\widehat{R} = \{A_1, \ldots, A_n\}$, the function $h_{23}(R)$ returns an NRMD$^\neg$ answer $A = (V, M)$ where: V is a set of variables $\{X_1, \ldots, X_n\}$ where variable X_i corresponds to attribute A_i (assume a simple transformation of attribute names to variable names); and, for each tuple t in R, there is a substitution θ in M satisfying that $\theta(X_i) = t(A_i)$ for every attribute $A_i \in \widehat{R}$, and $\mathrm{card}(\theta, M) = \mathrm{card}(t, R)$.*

Lemma 12. *NRMD$^\neg$ can be simulated in MRA.*

Proof. This is a long but straightforward induction on Datalog queries using as hypothesis that (f_{23}, g_{23}, h_{23}) is a simulation of NRMD$^\neg$ in MRA. The details of this proof are in the appendix (Claim 7). □

7.3. NRMD$^\neg$ and MRA have the same expressive power

Putting together the simulations between NRMD$^\neg$ and MRA stated in this section, we get the following theorem:

Theorem 2. *MRA and NRMD$^\neg$ have the same expressive power.*

Proof. If follows from lemmas 11 and 12. □

8. SPARQL ≡ MRA

This section presents the simulations that prove that SPARQL and Multiset Relational Algebra (MRA) have the same expressive power.

8.1. MRA to SPARQL

This section describes—according to the notation in Figure 2—the simulation T_{31} which includes the following translation functions:

- function f_{31} which translates a MRA query into a SPARQL query;
- function g_{31} which translates a MRA database into a SPARQL database; and
- function h_{31} which translates a SPARQL query solution into a MRA query solution.

8.1.1. MRA database to SPARQL database

Recall that a MRA database is a set of MRA relations (where each MRA relation is a multiset of tuples), and a SPARQL database is a set of RDF triples (i.e. an RDF graph). First, we define a method to translate a MRA relation into a set of RDF triples. Then, we define a method to translate a set of MRA relations.

Assume the existence of functions that map: relation names to IRIs, and relation attributes to IRIs.

Given a MRA relation r, the function $\beta(r)$ returns a set of triples defined as follows: for each tuple t in r, $\beta(r)$ contains a triple (u_t, u_b, u_r) where u_t is a IRI which identifies the tuple t, u_r is an IRI which identifies the relation r, and u_b is a IRI which describes that u_t is a tuple of u_r; and, for each attribute A in $\widehat{r}$, $\beta(r)$ contains a triple of the form (u_t, u_A, v_A) where u_A is a IRI which identifies the attribute A, and v_A is the value $t(A)$.

Definition 18 (Function g_{31})**.** *Given a MRA database D, the function $g_{31}(D)$ returns a multiset of RDF triples D' defined as follows:*

- *For each multiset relation r in D, D' contains the RDF triples returned by $\beta(r)$;*
- *D' contains a triple $(\texttt{NULL}, \texttt{NULL}, \texttt{NULL})$ where NULL is a special IRI. Like in the simulation of NRMD$^\neg$ with SPARQL, the simulation of MRA with SPARQL uses this especial triple to retrieve the variables that are attributes of the MRA answer.*

8.1.2. MRA query to SPARQL query

Recall that an MRA answer is a multiset relation r over a set of attributes $\widehat{r}$. Unlike MRA, SPARQL answers do not specify a set of variables for which solutions are defined. For example, the evaluation of the triple pattern $(?X, ?Y, ?Z)$ over an empty RDF graph results in an empty multiset Ω. The reference to the variables is not carried in the SPARQL answer. Like with the simulation of NRMD$^\neg$ with SPARQL, we need to define a SPARQL pattern to retrieve the answer variables.

Given an MRA expression E, with attributes $\widehat{E} = \{X_1, \ldots, X_n\}$, we write $\mathrm{AttrQuery}(E)$ to denote the SPARQL pattern $(\texttt{NULL}, \texttt{NULL}, ?X_1)$ AND $\cdots$ AND $(\texttt{NULL}, \texttt{NULL}, ?X_n)$, where, for $1 \leqslant i \leqslant n$, variable $?X_i$ is the corresponding SPARQL variable for the MRA attribute X_i.

Example 6. *Consider the MRA expression $r \bowtie s$ where $\widehat{r} = \{X, Y\}$ and $\widehat{s} = \{Y, Z\}$. Then, $\mathrm{AttrQuery}(E) = (\texttt{NULL}, \texttt{NULL}, ?X)$ AND $(\texttt{NULL}, \texttt{NULL}, ?Y)$ AND $(\texttt{NULL}, \texttt{NULL}, ?Z)$, where $?X$, $?Y$, and $?Z$ are the corresponding SPARQL variables for attributes X, Y, and Z.*

Recall that a MRA query is an MRA expression, and a SPARQL query is a SPARQL graph pattern. We will show that every type of MRA expression can be translated to a specific type of SPARQL graph pattern. Table 6 shows the translation rules which are the basis for the following definition.

Definition 19 (Function f_{31})**.** *Given an MRA expression E, the function f_{31} returns a SPARQL graph pattern defined by $(\Upsilon(E)$ UNION $\mathrm{AttrQuery(E)})$.*

Table 6
Definition of function Υ which translates an MRA expression into a SPARQL pattern.

MRA expression E	SPARQL pattern $\Upsilon(E)$	where ...
R	(SELECT $?X_1 \cdots ?X_n$ $((?Y, u_b, u_r)$ AND $((?Y, u_1, ?X_1)$ AND $(\cdots$ AND $(?Y, u_2, ?X_n) \cdots)$	u_r is the IRI that identifies R, $?Y$ is a variable used to match every tuple of R, and X_i is a variable that corresponds to the attribute A_i in schema $\widehat{R}$.
$(E_1 \ltimes E_2)$	$(P_1$ AND $P_2)$	$P_1 = \Upsilon(E_1)$ and $P_2 = \Upsilon(E_2)$.
$(E_1 \cup E_2)$	$(P_1$ UNION $P_2)$	$P_1 = \Upsilon(E_1)$ and $P_2 = \Upsilon(E_2)$.
$(E_1 \setminus E_2)$	$(P_1$ EXCEPT $P_2)$	$P_1 = \Upsilon(E_1)$ and $P_2 = \Upsilon(E_2)$.
$\pi_S(E_1)$	(SELECT $W P_1$)	$P_1 = \Upsilon(E_1)$ and W is the set of variables corresponding to the attributes in S.
$\rho_{A/B}(E_1)$	$\mathrm{subs}_{?X/?Y}(P_1)$	$P_1 = \Upsilon(E_1)$, $?X$ is the variable that corresponds to attribute A, $?Y$ is the variable that corresponds to attribute B, and $\mathrm{subs}_{?X/?Y}(P_1)$ denotes the renaming of variable $?X$ with variable $?Y$ in the SPARQL query P_1 (see Appendix A).
$\sigma_\psi(E_1)$	$(P_1$ FILTER $\varphi)$	$P_1 = \Upsilon(E_1)$, and φ is a filter condition equivalent to the selection condition ψ.

8.1.3. SPARQL solution to MRA solution

Recall that a SPARQL solution is a multiset of mappings, and a MRA solution is a multiset relation. Intuitively, a multiset of mappings Ω can be transformed into a MRA relation r where the attributes in $\widehat{r}$ are the variables in the domain of Ω. This notion is defined next.

Definition 20 (Function h_{31})**.** *Let Ω be a multiset of mappings with $\mathrm{dom}(\mu) = V$ for every mapping $\mu \in \Omega$, and that includes the mapping $\mu_{V \mapsto \texttt{NULL}}$ with $\mathrm{dom}(\mu_{V \mapsto \texttt{NULL}}) = V$, $\mu_{V \mapsto \texttt{NULL}}(?X) = \texttt{NULL}$ for every variable $?X \in V$, . The function $h_{31}(\Omega)$ returns a multiset relation r where:*

- *For each variable $?X \in V$, the schema $\widehat{r}$ includes the MRA attribute A corresponding to variable $?X$.*
- *The tuple t_μ corresponding to a mapping μ with $\mathrm{dom}(\mu) = V$ is the tuple with attributes $\widehat{r}$ such that $t(A) = \mu(?X)$, for each MRA attribute $A \in \widehat{r}$ corresponding to a variable $?X \in V$.*
- $\mathrm{set}(r) = \{t_\mu \mid \mu \in \mathrm{set}(\Omega) \setminus \{\mu_{V \mapsto \mathtt{NULL}}\}\}$.
- $\mathrm{card}(t_\mu, r) = \mathrm{card}(\mu, \Omega)$

Lemma 13. *MRA can be simulated in SPARQL.*

Proof. Let f_{31}, g_{31}, h_{31} denote respectively the functions stated in definitions and 19, 18, and 20. The proof of this theorem follows from the claim that (f_{31}, g_{31}, h_{31}) simulates MRA in SPARQL by using induction in the structure of queries. The proof of this claim is in the appendix (Claim 8). □

8.2. SPARQL to MRA

This section describes—according to the notation in Figure 2—the simulation T_{13} which includes the following translation functions:

- function f_{13} which translates a SPARQL query into a MRA query;
- function g_{13} which translates a SPARQL database into a MRA database; and
- function h_{13} which translates a MRA query solution into a SPARQL query solution.

The translation presented here is inspired by the one presented by Cyganiak [42]. However, unlike Cyganiak, we do not use null values with the SQL semantics. Instead, we use a special constant, denoted $\perp$, used to codify unbound values.

8.2.1. SPARQL database to MRA database

Recall that a SPARQL database is a set of RDF triples (i.e. an RDF graph), and a MRA database is a set of MRA relations (where each MRA relation is a set of tuples). The translation of a set of RDF triples G will produce three multiset relations (without duplicates): Trip, which codifies the RDF triples in G; Null, introduced to manage the unbound values of SPARQL; and Comp, introduced to simulate the notion of compatibility between mappings.

Let $\perp$ be a special constant. Given a set of RDF triples G, the multiset relations Trip, Null, and Comp are defined as follows:

1. $\widehat{\mathrm{Trip}} = \{S, P, O\}$, $\mathrm{set}(\mathrm{Trip}) = \{\{S \mapsto s, P \mapsto p, O \mapsto o\} \mid (s, p, o) \in G\}$, and $\mathrm{card}(t, \mathrm{Trip}) = 1$ for every tuple $t \in \mathrm{set}(\mathrm{Trip})$.
2. $\widehat{\mathrm{Null}} = \{N\}$, $\mathrm{set}(\mathrm{Null}) = \{\{N \mapsto \perp\}\}$, and $\mathrm{card}(\{N \mapsto \perp\}, \mathrm{Null}) = 1$.
3. $\widehat{\mathrm{Comp}} = \{A_1, A_2, A\}$, $\mathrm{set}(\mathrm{Comp})$ includes the tuple $\{A_1 \mapsto \perp, A_2 \mapsto \perp, A \mapsto \perp\}$ and all tuples of the form $\{A_1 \mapsto a, A_2 \mapsto a, A \mapsto a\}$, $\{A_1 \mapsto \perp, A_2 \mapsto a, A \mapsto a\}$, and $\{A_1 \mapsto a, A_2 \mapsto \perp, A \mapsto a\}$ where a is an RDF term in G, and $\mathrm{card}(t, \mathrm{Comp}) = 1$ for every tuple $t \in \mathrm{set}(\mathrm{Comp})$.

Definition 21 (Function g_{13})**.** *Given a set of RDF triples D (i.e., an RDF graph), the function $g_{13}(D)$ returns a multiset relational database D' containing the multiset relations* Trip*,* Null*, and* Comp *defined above.*

8.2.2. SPARQL query to MRA query

First, we define function ϵ which allows translating a triple pattern (the simplest SPARQL query) into a MRA expression. Recall that Trip is a multiset relation that is obtained from a multiset of RDF triples, where $\widehat{\mathrm{Trip}} = \{S, P, O\}$.

Given a triple pattern $T = (s, p, o)$, the function $\epsilon(T)$ returns a MRA expression defined as follows[3]:

1. Let Σ_C and Σ_V be the sets of identities

$$\Sigma_C = \{S = s \mid s \notin \mathbf{V}\} \cup \{P = p \mid p \notin \mathbf{V}\} \cup \{O = o \mid o \notin \mathbf{V}\},$$
$$\Sigma_V = \{S = P \mid s, p \in \mathbf{V}\} \cup \{S = O \mid s, o \in \mathbf{V}\} \cup \{P = O \mid p, o \in \mathbf{V}\}.$$

[3] These rules are based on Cyganiak's translation [42].

2. Let $\mathrm{Selection}(s, p, o)$ be the MRA expression defined as follows:

$$\mathrm{Selection}(T) = \begin{cases} \sigma_{\bigwedge_{\varphi \in \Sigma_C \cup \Sigma_V}}(\mathrm{Trip}) & \text{if } \Sigma_C \cup \Sigma_V \neq \emptyset, \\ \mathrm{Trip} & \text{otherwise.} \end{cases}$$

3. The *attributes of a variable* $?X$ occurring in triple pattern T is the set of attributes

$$\mathrm{Attr}(?X, T) = \{S \mid s = ?X\} \cup \{P \mid p = ?X\} \cup \{O \mid o = ?X\}.$$

   The first position of a variable $?X \in \mathrm{inScope}(T)$, denoted $\mathrm{minAttr}(?X, T)$, is the minimum in $\mathrm{Attr}(?X, T)$ assuming $S < P < O$. Intuitively, $\mathrm{minAttr}(?X, T)$ is the attribute that corresponds to the first position where variable $?X$ occurs in triple T.
4. Let $\mathrm{Renaming}(T)$ be the MRA expression that results for applying the renaming $\rho_{\mathrm{minAttr}(?X,T)/X}$ over $\mathrm{Selection}(T)$, for each variable $?X \in \mathrm{inScope}(T)$ and MRA attribute X corresponding to variable $?X$.
5. Let $\mathrm{Projection}(T)$ be the MRA expression $\pi_W(\mathrm{Renaming}(T))$ where W is the set of attributes corresponding to the in-scope variables in triple pattern T.
6. $\epsilon(T)$ is the MRA expression $\mathrm{Projection}(T)$.

Example 7 (Simulation of SPARQL triple patterns with MRA)**.** *The following are examples of the translation of a SPARQL triple pattern T with MRA, using function ϵ.*

- *if T is $(?X, b, c)$ then $\epsilon(T)$ returns $\pi_{?X}(\rho_{S/?X}(\sigma_{P=b \wedge O=c}(\mathrm{Trip})))$;*
- *if T is $(a, ?Y, c)$ then $\epsilon(T)$ returns $\pi_{?Y}(\rho_{P/?Y}(\sigma_{S=a \wedge O=c}(\mathrm{Trip})))$;*
- *if T is $(?X, ?Y, c)$ then $\epsilon(T)$ returns $\pi_{?X,?Y}(\rho_{S/?X}(\rho_{P/?Y}(\sigma_{O=c}(\mathrm{Trip}))))$;*
- *if T is $(?X, b, ?X)$ then $\epsilon(T)$ returns $\pi_{?X}(\rho_{S/?X}(\sigma_{P=b \wedge S=O}(\mathrm{Trip})))$.*

Second, we define a function γ which allows translating a SPARQL filter condition into a MRA selection condition. Like in the translation from SPARQL to NRMD$^\neg$, it is not necessary to translate complex filter conditions (SPARQL) to complex selection formulas (MRA) because SPARQL queries can be normalized to avoid logical connectives.

Given an atomic filter condition φ, the function $\gamma(\varphi)$ is defined recursively as follows:

- If φ is $?X = c$ then $\gamma(\varphi)$ is $(\neg(X = \bot) \wedge X = c)$ where X is the attribute name corresponding to variable $?X$;
- If φ is $?X = ?Y$ then $\gamma(\varphi)$ is $((\neg(X = \bot) \wedge \neg(Y = \bot)) \wedge X = Y)$ where X and Y are the attribute names corresponding to variables $?X$ and $?Y$, respectively;
- If φ is $\mathrm{bound}(X)$ then $\gamma(\varphi)$ is $\neg(X = \bot)$ where X is the attribute name corresponding to variable $?X$.

In Definition 21, we introduced the relation named Comp to simulate the compatibility between mappings. For example, to simulate the SPARQL query $Q = (P_1 \,\mathrm{AND}\, P_2)$ we need to ensure that check if two pairs of mappings $\mu_1 \in [\![P_1]\!]_G$ and $\mu_2 \in [\![P_2]\!]_G$ are compatible, and if they are compatible, return the mapping $\mu = \mu_1 \cup \mu_2$ resulting from joining them. To explain how this operation is simulated with MRA, let $\mathrm{inScope}(P_1) \cap \mathrm{inScope}(P_2) = \{?X\}$ and tuples t_1 and t_2 correspond to mappings μ_1 and μ_2. To be compatible, either both mappings map variable $?X$ to the same value, or at least for one of the mappings, variable $?X$ is unbound. For tuples, an unbound variable $?X$ is represented with an attribute value $\bot$ (e.g., $t(X) = \bot$). Then, to check if tuples t_1 and t_2 are *compatible*, we need to rename the attribute name X corresponding to variable $?X$ as two attributes, namely X_1 and X_2 and check if there exists a tuple t_3 in the result of query $\rho_{A_1/X_1}(\rho_{A_2/X_2}(\mathrm{Comp}))$ that agrees with tuples t_1 and t_2 (i.e., $t_3(X_1) = t_1(X)$ and $t_3(X_2) = t_2(X)$) or agrees with either t_1 or t_2 whereas for the other tuple the value is $\bot$ (e.g., $t_3(X_1) = t_1(X)$ and $t_2(X_2) = \bot$). We recover the renamed attribute X for the attribute A in the relation named Comp. That is, for the compatibility we use the MRA expression $\rho_{A/X}(\rho_{A_1/X_1}(\rho_{A_2/X_2}(\mathrm{Comp})))$ which is generalized as follows for multiple common variables in the scope of patterns P_1 and P_3.

Let $\mathcal{X}$ be a finite set of attribute names, and ν_1 and ν_2 be two bijective functions that map each attribute $X \in \mathcal{X}$ to two different sets of attributes (i.e., the ranges of ν_1 and ν_2 are disjoint). Then, we write $\mathrm{Comp}(\nu_1, \nu_2, \mathcal{X})$ to denote the join of MRA expressions of the form $\rho_{A/X}(\rho_{A_1/\nu_1(X)}(\rho_{A_2/\nu_2(X)}(\mathrm{Comp})))$, for every attribute name $X \in \mathcal{X}$.

Let E be a MRA expression, and $\mathcal{X} = \{X_1, \ldots, X_n\}$ be a subset of the attribute names in $\widehat{E}$, and ν a bijective function that maps each attribute name in $\mathcal{X}$ to a fresh attribute name (i.e., $\nu(X) \notin \widehat{E}$). We call $\nu(E)$ to the MRA expression that renames each attribute name $X \in \mathcal{X}$ with $\nu(X)$. That is, $\nu(E) = \rho_{X_1/\nu(X_1)}(\cdots \rho_{X_n/\nu(X_n)}(E) \cdots)$.

Given two MRA expressions E_1 and E_2, assume two bijective functions ν_1 and ν_2 that map each attribute $X \in \widehat{E}_1 \cap \widehat{E}_2$ to two fresh attributes (i.e., $\nu_1(X), \nu_2(X) \notin \widehat{E}_1 \cup \widehat{E}_2$), and satisfy $\mathrm{range}(\nu_1) \cap \mathrm{range}(\nu_2) = \emptyset$. Then, we define the MRA operation $E_1 * E_2$ in terms of existing MRA operators as follows:

$$E_1 * E_2 = \pi_{\widehat{E}_1 \cup \widehat{E}_2}(\mathrm{Comp}(\nu_1, \nu_2, \widehat{E}_1 \cap \widehat{E}_2) \bowtie \nu_1(E_1) \bowtie \nu_2(E_2)).$$

Notice that the attribute names in the ranges of functions μ_1 and μ_2 in the definition of expression $E_1 * E_2$ do not matter because are not in the schema of the multiset that results from expression $E_1 * E_2$.

To translate SPARQL queries Q of the form $(\mathrm{SELECT}\ \mathcal{X}\ P)$ where the set of variables $\mathcal{X}$ include a variable that is not in the scope of P, we need to generate values $\perp$ to fill the tuples returned by the translated query. For example, if $\mathrm{inScope}(P) = \{?X\}$ and $\mathcal{X} = \{?X, ?Y\}$, then the MRA expression E that corresponds to the SPARQL pattern P can be extended with an attribute name Y by joining E with the MRA relation $\rho_{N/Y}(\mathrm{Null})$. Given a set $\mathcal{Y} = \{Y_1, \ldots, Y_n\}$ of attribute names, we define the MRA expression $\Delta(\mathcal{Y})$ as $\rho_{N/Y_1}(\mathrm{Null}) \bowtie \cdots \bowtie \rho_{N/Y_n}(\mathrm{Null})$.

We next present the translation from SPARQL patterns as MRA queries.

Definition 22 (SPARQL patterns as MRA queries)**.** *The translation rules in Table 7 define the function $f_{1,3}$ from graph patterns whose selection formulas have no Boolean connectives to MRA queries.*

Table 7

Definition of the function $f_{1,3}$, which takes a normalized SPARQL pattern P as input (without logical connectives in selection formulas) and returns an MRA query.

SPARQL pattern P	MRA query $f_{1,3}(P)$	where...
$P_\emptyset$	$\pi_\emptyset(\mathrm{Null})$	
(s, p, o)	$\epsilon(s, p, o)$	
$(P_1\ \mathrm{AND}\ P_2)$	$(f_{1,3}(P_1) * f_{1,3}(P_2))$	
$(P_1\ \mathrm{UNION}\ P_2)$	$(f_{1,3}(P_1) \cup f_{1,3}(P_2))$	
$(P_1\ \mathrm{EXCEPT}\ P_2)$	$(f_{1,3}(P_1) \setminus f_{1,3}(P_2))$	
$(\mathrm{SELECT}\ \mathrm{inScope}(P)\ P_1)$	$\pi_{\mathcal{A}}(f_{1,3}(P_1) \bowtie \Delta_{\mathcal{B}})$	$\mathcal{A}$ is the set of attribute names corresponding to the variables in set $\mathrm{inScope}(P)$ and $\mathcal{B}$ is the set of attribute names that correspond to variables that are in set $\mathrm{inScope}(P) \setminus \mathrm{inScope}(P_1)$.
$(P_1\ \mathrm{FILTER}\ \varphi)$	$\sigma_{\gamma(\varphi)}(f_{1,3}(P_1))$	

8.2.3. MRA solution to SPARQL solution

Intuitively, the translation of a MRA tuple t as a SPARQL solution mapping μ consists of removing from tuple t every attribute whose value is $\perp$, and viewing the result tuple as a SPARQL mapping μ. For example, the result of translating a tuple t with $\hat{t} = \{X, Y\}$, $t(X) = a$, and $t(Y) = \perp$, is the RDF mapping $\mu = \{?X \mapsto a\}$. Recall that we write $?X$ to denote the corresponding SPARQL variable for a MRA attribute X.

Definition 23 (Tuples to mappings)**.** *The function $h_{3,1}$ (from MRA answers to SPARQL answers) is defined as follows. Given a MRA tuple t, we write $f_{3,1}(t)$ to denote the SPARQL mapping μ such that: (1) $\mu(?X) = t(X)$ if $X \in \hat{t}$ and $t(X) \neq \perp$, and (2) variable $?Y$ is not in $\mathrm{dom}(\mu)$ if $Y \notin \hat{t}$ or $t(Y) = \perp$. Abusing of notation, $f_{1,3}$ is also the function that receives a MRA relation r and returns the multiset Ω of SPARQL mappings where $\mathrm{set}(\Omega) = \{\mu \mid$ there exist $t \in r$ such that $f_{3,1}(t) = \mu\}$ and the multiplicity of mapping $f_{3,1}(t)$ in Ω is the multiplicity of tuple t in r.*

Lemma 14. *SPARQL can be simulated in MRA.*

Proof. This is a long but straightforward induction on the structure of SPARQL queries using as hypothesis that (f_{13}, g_{13}, h_{13}) is a simulation of SPARQL in MRA. The details of this proof are in the appendix (Claim 9). □

8.3. MRA and SPARQL have the same expressive power

Putting together the simulations among MRA and SPARQL stated in this section, we get the following theorem:

Theorem 3. *MRA and SPARQL have the same expressive power.*

Proof. If follows from lemmas 13 and 14. □

9. Related work and Conclusions

To the best of our knowledge, the multiset semantics of SPARQL has not been systematically addressed. There are works that, when studying the expressive power of SPARQL, touched some aspects of this topic. Cyganiak [42] was among the first who gave a translation of a core fragment of SPARQL into relational algebra. Polleres [34] proved the inclusion of the fragment of SPARQL patterns with safe filters into Datalog by giving a precise and correct set of rules. Schenk [43] proposed a formal semantics for SPARQL based on Datalog, but concentrated on complexity more than expressiveness issues. Both, Polleres and Schenk did not consider multiset semantics of SPARQL in their translations. Perez et al. [44] gave the first formal treatment of multiset semantics for SPARQL. Angles and Gutierrez [45], Polleres [46] and Schmidt et al. [47] extended the set semantics to multiset semantics using this idea. Kaminski et al. [48] considered multisets in subqueries and aggregates in SPARQL. Recently, Angles et al. [49] implemented the translation from SPARQL to Datalog (inside the Vadalog system [50]). In none of these works was addressed the goal of characterizing the multiset algebraic and/or logical structure of the operators in SPARQL.

We studied the multiset semantics of the core SPARQL patterns, in order to shed light on the algebraic and logic structure of them. In this regard, the discoveries that: (1) the core fragment of SPARQL patterns matches precisely the multiset semantics of Datalog as defined by Mumick et al. [29]; and (2) this logical structure corresponds to a simple multiset algebra, namely the Multiset Relational Algebra (MRA); build a nice parallel to the one exhibit by classical set relational algebra and relational calculus.

Contrary to the rather chaotic variety of multiset operators in SQL, it is interesting to observe that in SPARQL there is a more coherent body of multiset operators. We suggest that this should be considered by designers in order to try to keep this clean design in future extensions of SPARQL.

Last, but not least, this study shows the complexities and challenges that the introduction of multisets brings to query languages, exemplified here in the case of SPARQL.

References

[1] J. Melton and A.R. Simon, *SQL:1999. Understanding Relational Language Components*, Morgan Kaufmann Publ., 2002.
[2] V. Breazu-Tannen and R. Subrahmanyam, Logical and computational aspects of programming with sets/bags/lists, in: *Automata, Languages and Programming*, J.L. Albert, B. Monien and M.R. Artalejo, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 1991, pp. 60–75.
[3] J.W. Lloyd, Programming with multisets, Technical Report, University of Bristol, 1998.
[4] J. Albert, Algebraic Properties of Bag Data Types, in: *Proc. of the Int. Conference on Very Large Data Bases (VLDB)*, 1991, pp. 211–219.
[5] L. Libkin and L. Wong, Some Properties of Query Languages for Bags, in: *Proc. of the Int. Workshop on Database Programming Languages (DBPL) - Object Models and Languages*, 1994, pp. 97–114.
[6] S. Grumbach, L. Libkin, T. Milo and L. Wong, Query languages for bags: expressive power and complexity, *SIGACT News* **27**(2) (1996), 30–44.
[7] S. Grumbach and T. Milo, Towards Tractable Algebras for Bags, *Journal of Computer and System Sciences* **52**(3) (1996), 570–588. doi:https://doi.org/10.1006/jcss.1996.0042. https://www.sciencedirect.com/science/article/pii/S0022000096900422.

[8] L.S. Colby and L. Libkin, Tractable iteration mechanisms for bag languages, in: *Database Theory — ICDT '97*, F. Afrati and P. Kolaitis, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 1997, pp. 461–475. ISBN 978-3-540-49682-3.

[9] L. Libkin and L. Wong, Query languages for bags and aggregate functions, *Journal of Computer and System Sciences* **55**(2) (1997), 241–272.

[10] U. Dayal, N. Goodman and R.H. Katz, An Extended Relational Algebra with Control over Duplicate Elimination, in: *Proc. of the Symposium on Principles of Database Systems (PODS)*, ACM, 1982, pp. 117–123.

[11] A. Klausner and N. Goodman, Multirelations - Semantics and languages, in: *Proc. of Very Large Databases*, 1985.

[12] M. Console, P. Guagliardo and L. Libkin, Fragments of bag relational algebra: Expressiveness and certain answers, *Information Systems* (2020), 101604. doi:https://doi.org/10.1016/j.is.2020.101604. https://www.sciencedirect.com/science/article/pii/S0306437920300855.

[13] I.S. Mumick, H. Pirahesh and R. Ramakrishnan, The Magic of Duplicates and Aggregates, in: *Proceedings of the 16th International Conference on Very Large Data Bases*, VLDB '90, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990, pp. 264–277. ISBN 1-55860-149-X. http://dl.acm.org/citation.cfm?id=645916.671834.

[14] I.S. Mumick, S.J. Finkelstein, H. Pirahesh and R. Ramakrishnan, Magic is Relevant, *SIGMOD Rec.* **19**(2) (1990), 247–258. doi:10.1145/93605.98734.

[15] S. Cohen, Equivalence of Queries That Are Sensitive to Multiplicities, *The VLDB Journal* **18**(3) (2009), 765–785. doi:10.1007/s00778-008-0122-1.

[16] F.N. Afrati, M. Damigos and M. Gergatsoulis, Query Containment Under Bag and Bag-set Semantics, *Information Processing Letters* **110**(10) (2010), 360–369.

[17] L. Bertossi, G. Gottlob and R. Pichler, Datalog: Bag Semantics via Set Semantics, in: *22nd International Conference on Database Theory (ICDT 2019)*, P. Barcelo and M. Calautti, eds, Leibniz International Proceedings in Informatics (LIPIcs), Vol. 127, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2019, pp. 16:1–16:19. ISSN 1868-8969. ISBN 978-3-95977-101-6. doi:10.4230/LIPIcs.ICDT.2019.16. http://drops.dagstuhl.de/opus/volltexte/2019/10318.

[18] P. Guagliardo and L. Libkin, A Formal Semantics of SQL Queries, Its Validation, and Applications, *Proc. VLDB Endow.* **11**(1) (2017), 27–39. doi:10.14778/3151113.3151116.

[19] W. Ricciotti and J. Cheney, Mixing Set and Bag Semantics, in: *Proceedings of the 17th ACM SIGPLAN International Symposium on Database Programming Languages*, DBPL 2019, Association for Computing Machinery, New York, NY, USA, 2019, pp. 70–73–. ISBN 9781450367189. doi:10.1145/3315507.3330202.

[20] A. Polleres and J.P. Wallner, On the relation between SPARQL1.1 and Answer Set Programming, *Journal of Applied Non-Classical Logics* **23**(1–2) (2013), 159–212.

[21] F. Geerts, T. Unger, G. Karvounarakis, I. Fundulaki and V. Christophides, Algebraic Structures for Capturing the Provenance of SPARQL Queries, *J. ACM* **63**(1) (2016). doi:10.1145/2810037.

[22] M. Kaminski, E.V. Kostylev and B. Cuenca Grau, Semantics and Expressive Power of Subqueries and Aggregates in SPARQL 1.1, in: *Proc. of the International Conference on World Wide Web*, 2016, pp. 227–238.

[23] R. Angles and C. Gutierrez, The Multiset Semantics of SPARQL Patterns, in: *15th International Semantic Web Conference (ISWC)*, LNCS, Springer, 2016, pp. 20–36.

[24] A. Hernich and P.G. Kolaitis, Foundations of information integration under bag semantics, in: *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2017, pp. 1–12. doi:10.1109/LICS.2017.8005104.

[25] C.J. Date, *Date on Database: Writings 2000-2006*, APress, 2006, Chapter Ch. 10: Double Trouble, Double Trouble.

[26] T.J. Green, Bag Semantics, in: *Encyclopedia of Database Systems*, 2009, pp. 201–206.

[27] G. Lamperti, M. Melchiori and M. Zanella, On Multisets in Database Systems, in: *Proceedings of the Workshop on Multiset Processing*, 2001, pp. 147–216.

[28] S. Abiteboul, R. Hull and V. Vianu, *Foundations of Databases*, Addison-Wesley, 1995.

[29] I.S. Mumick, H. Pirahesh and R. Ramakrishnan, The Magic of Duplicates and Aggregates, in: *Proc. of the Int. Conference on Very Large Data Bases (VLDB)*, 1990, pp. 264–277.

[30] D. Hernández, The Problem of Incomplete Data in SPARQL, Ph.D. dissertation, Universidad de Chile - Faculty of Physical and Mathematical Sciences, Santiago, Chile, 2020. https://repositorio.uchile.cl/handle/2250/178033.

[31] X. Zhang and J.V. den Bussche, On the primitivity of operators in SPARQL, *Inf. Process. Lett.* **114**(9) (2014), 480–485.

[32] R. Kontchakov and E.V. Kostylev, On Expressibility of Non-Monotone Operators in SPARQL, in: *Int. Conference on the Principles of Knowledge Representation and Reasoning*, 2016.

[33] R. Angles and C. Gutierrez, Negation in SPARQL, in: *Alberto Mendelzon Int. Workshop on Foundations of Data Management (AMW)*, 2016.

[34] A. Polleres, From SPARQL to rules (and back), in: *WWW*, ACM, 2007, pp. 787–796.

[35] R. Angles and C. Gutiérrez, The Expressive Power of SPARQL, in: *International Semantic Web Conference*, Lecture Notes in Computer Science, Vol. 5318, Springer, 2008, pp. 114–129.

[36] R. Angles and C. Gutiérrez, The Multiset Semantics of SPARQL Patterns, in: *International Semantic Web Conference (1)*, Lecture Notes in Computer Science, Vol. 9981, 2016, pp. 20–36.

[37] A. Polleres and J.P. Wallner, On the relation between SPARQL1.1 and Answer Set Programming, *J. Appl. Non Class. Logics* **23**(1–2) (2013), 159–212.

[38] E. Prud'hommeaux and A. Seaborne, SPARQL Query Language for RDF. W3C Recommendation, 2008.

[39] S. Harris and A. Seaborne, SPARQL 1.1 Query Language - W3C Recommendation, 2013.

[40] A. Hogan, M. Arenas, A. Mallea and A. Polleres, Everything You Always Wanted to Know About Blank Nodes, *Journal of Web Semantics* **27**(1) (2014).
[41] L.E. Bertossi, G. Gottlob and R. Pichler, Datalog: Bag Semantics via Set Semantics, in: *ICDT*, LIPIcs, Vol. 127, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 16:1–16:19.
[42] R. Cyganiak, A relational algebra for SPARQL, Technical Report, HPL-2005-170, HP Labs, 2005.
[43] S. Schenk, A SPARQL Semantics Based on Datalog, in: *Annual German Conference on Advances in Artificial Intelligence*, Vol. 4667, 2007, pp. 160–174.
[44] J. Pérez, M. Arenas and C. Gutierrez, Semantics of SPARQL, Technical Report, TR/DCC-2006-17, Department of Computer Science, University of Chile, 2006.
[45] R. Angles and C. Gutierrez, The Expressive Power of SPARQL, in: *Proc. of the Int. Semantic Web Conference (ISWC)*, 2008, pp. 114–129.
[46] A. Polleres, How (well) Do Datalog, SPARQL and RIF Interplay?, in: *Proc. of the Int. conference on Datalog in Academia and Industry*, 2012, pp. 27–30.
[47] M. Schmidt, M. Meier and G. Lausen, Foundations of SPARQL query optimization, in: *Proc. of the Int. Conference on Database Theory*, ACM, 2010, pp. 4–33.
[48] M. Kaminski, E.V. Kostylev and B.C. Grau, Semantics and Expressive Power of Subqueries and Aggregates in SPARQL 1.1., in: *Proceedings of the Int. Conference on World Wide Web (WWW)*, ACM, 2016, pp. 227–238.
[49] R. Angles, G. Gottlob, A. Pavlović, R. Pichler and E. Sallinger, SparqLog: A System for Efficient Evaluation of SPARQL 1.1 Queries via Datalog, *Proc. VLDB Endow.* **16**(13) (2023), 4240–4253–. doi:10.14778/3625054.3625061.
[50] L. Bellomarini, E. Sallinger and G. Gottlob, The Vadalog system: datalog-based reasoning for knowledge graphs, *Proc. VLDB Endow.* **11**(9) (2018), 975–987–. doi:10.14778/3213880.3213888.

Appendix A. Variable renaming in SPARQL

This appendix section defines function $\mathrm{subs}(\cdot,\cdot)$, which renames SPARQL variables. This function is used to simulate the MRA operator renaming $\rho_{A/B}$ (see Table 6). Note that function $\mathrm{subs}(\cdot,\cdot)$ is not an additional algebraic operation but an operation over expressions (i.e., a query rewriting). Intuitively, given a MRA query Q, a SPARQL pattern P that simulates Q, a renaming of MRA attributes A/B and a renaming of variables $?X/?Y$ where $?X$ and $?Y$ are the corresponding variables for attributes A and B, the query rewriting $\mathrm{subs}_{?X/?Y}(P)$ simulates the MRA query $\rho_{A,B}(Q)$. To this end, SPARQL variables are renamed in the pattern, instead of renaming query result attributes as MRA does.

Definition 24 (SPARQL Variable Renaming)**.** *Given two SPARQL variables $?X$ and $?Y$, we define the function $\nu_{?X/?Y} : \mathbf{I}\cup\mathbf{L}\cup\mathbf{V} \to \mathbf{I}\cup\mathbf{L}\cup\mathbf{V}$ as the function such that $\nu_{?X/?Y}(?X) = ?Y$ and $\nu_{?X/?Y}(s) = s$, for every $s \in (\mathbf{I}\cup\mathbf{L}\cup\mathbf{V})\setminus\{?X\}$. Given a SPARQL pattern P and two SPARQL variables $?X \in \mathrm{inScope}(P)$ and $?Y \notin \mathrm{inScope}(P)$, we write $\mathrm{subs}_{?X/?Y}(P)$ to denote the pattern defined recursively as follows:*

1. *If P is a triple pattern (s,p,o) then $\mathrm{subs}_{?X/?Y}(P) = (\nu_{?X/?Y}(s), \nu_{?X/?Y}(p), \nu_{?X/?Y}(o))$.*
2. *If P has the form $(P_1 \,\mathrm{AND}\, P_2)$ then $\mathrm{subs}_{?X/?Y}(P) = \mathrm{subs}_{?X/?Y}(P_1) \,\mathrm{AND}\, \mathrm{subs}_{?X/?Y}(P_2)$.*
3. *If P has the form $(P_1 \,\mathrm{UNION}\, P_2)$ then $\mathrm{subs}_{?X/?Y}(P) = \mathrm{subs}_{?X/?Y}(P_1) \,\mathrm{UNION}\, \mathrm{subs}_{?X/?Y}(P_2)$.*
4. *If P has the form $(P_1 \,\mathrm{EXCEPT}\, P_2)$ then $\mathrm{subs}_{?X/?Y}(P) = \mathrm{subs}_{?X/?Y}(P_1) \,\mathrm{EXCEPT}\, \mathrm{subs}_{?X/?Y}(P_2)$.*
5. *If P has the form $(P_1 \,\mathrm{FILTER}\, \varphi)$ then $\mathrm{subs}_{?X/?Y}(P) = (\mathrm{subs}_{?X/?Y}(P_1) \,\mathrm{FILTER}\, \nu_{?X/?Y}(\varphi))$ where, abusing of notation, $\nu_{?X/?Y}(\varphi)$ is the selection formula defined recursively as follows:*
   (a) *If φ has the form $a = b$, where $a, b \in \mathbf{V}\cup\mathbf{I}\cup\mathbf{I}$, then $\nu_{?X/?Y}(\varphi) = \nu_{?X/?Y}(a) = \nu_{?X/?Y}(b)$.*
   (b) *If φ has the form $\mathrm{bound}(?x)$ then $\nu_{?X/?Y}(\varphi) = \mathrm{bound}(\nu_{?X/?Y}(?x))$.*
   (c) *If φ has the form $\psi_1 \wedge \psi_2$ then $\nu_{?X/?Y}(\varphi) = \nu_{?X/?Y}(\psi_1) \wedge \nu_{?X/?Y}(\psi_2)$.*
   (d) *If φ has the form $\psi_1 \vee \psi_2$ then $\nu_{?X/?Y}(\varphi) = \nu_{?X/?Y}(\psi_1) \vee \nu_{?X/?Y}(\psi_2)$.*
   (e) *If φ has the form $\neg\psi$ then $\nu_{?X/?Y}(\varphi) = \neg\nu_{?X/?Y}(\psi)$.*
6. *If P has the form $(\mathrm{SELECT}\, W\, \mathrm{WHERE}\, P_1)$ then:*
   (a) *If $?Y \notin \mathrm{inScope}(P_1)$, then $\mathrm{subs}_{?X/?Y}(P) = (\mathrm{SELECT}\ (W \setminus \{?X\} \cup \{?Y\})\, \mathrm{WHERE}\ P_1)$.*
   (b) *Otherwise, $\mathrm{subs}_{?X/?Y}(P) = (\mathrm{SELECT}\ (W \setminus \{?X\} \cup \{?Y\})\, \mathrm{WHERE}\ \mathrm{subs}_{?Y/Z}(P_1))$, where $?Z$ is a fresh variable. We rename variable $?Y$ as $?Z$ when is not in-scope of P to avoid a variable name clash.*

Table 8

Truth values for the error formula of a conjunction. According to Definition 5, given a formula φ of the form $\varphi_1 \wedge \varphi_2$, the formula $\mathrm{Error}(\varphi)$ is the formula $\psi_1 \vee \psi_2 \vee \psi_3$ where ψ_1 is the formula $(\varphi_1 \wedge \mathrm{Error}(\varphi_2))$, ψ_2 is the formula $(\mathrm{Error}(\varphi_1) \wedge \varphi_2)$, and ψ_3 is the formula $(\mathrm{Error}(\varphi_1) \wedge \mathrm{Error}(\varphi_2))$. Given an arbitrary mapping μ, this table shows the possible truth values for formulas φ, $\mathrm{Error}(\varphi)$, and its components.

$\mu(\varphi_1)$	$\mu(\varphi_2)$	$\mu(\varphi)$	$\mu(\mathrm{Error}(\varphi_1))$	$\mu(\mathrm{Error}(\varphi_2))$	$\mu(\psi_1)$	$\mu(\psi_2)$	$\mu(\psi_3)$	$\mu(\mathrm{Error}(\varphi))$
true	*true*	*true*	*false* or *error*	*false* or *error*	*false* or *error*	*false* or *error*	*false* or *error*	*false* or *error*
true	*false*	*false*	*false* or *error*	*false* or *error*	*false* or *error*	*false*	*false* or *error*	*false* or *error*
true	*error*	*error*	*false* or *error*	*true*	*true*	*false* or *error*	*false* or *error*	*true*
false	*true*	*false*	*false* or *error*	*false* or *error*	*false*	*false* or *error*	*false* or *error*	*false* or *error*
false	*false*	*false*	*false* or *error*	*false* or *error*	*false*	*false*	*false* or *error*	*false* or *error*
false	*error*	*false*	*false* or *error*	*true*	*false*	*false* or *error*	*false* or *error*	*false* or *error*
error	*true*	*error*	*true*	*false* or *error*	*false* or *error*	*true*	*false* or *error*	*true*
error	*false*	*false*	*true*	*false* or *error*	*false* or *error*	*false*	*false* or *error*	*false* or *error*
error	*error*	*error*	*true*	*true*	*error*	*error*	*true*	*true*

Table 9

Truth values for the error formula of a disjunction. According to Definition 5, given a formula φ of the form $\varphi_1 \vee \varphi_2$, the formula $\mathrm{Error}(\varphi)$ is the formula $\psi_1 \vee \psi_2 \vee \psi_3$ where ψ_1 is the formula $(\neg\varphi_1 \wedge \mathrm{Error}(\varphi_2))$, ψ_2 is the formula $(\mathrm{Error}(\varphi_1) \wedge \neg\varphi_2)$, and ψ_3 is the formula $(\mathrm{Error}(\varphi_1) \wedge \mathrm{Error}(\varphi_2))$. Given an arbitrary mapping μ, this table shows the possible truth values for formulas φ, $\mathrm{Error}(\varphi)$, and its components.

$\mu(\varphi_1)$	$\mu(\varphi_2)$	$\mu(\varphi)$	$\mu(\mathrm{Error}(\varphi_1))$	$\mu(\mathrm{Error}(\varphi_2))$	$\mu(\psi_1)$	$\mu(\psi_2)$	$\mu(\psi_3)$	$\mu(\mathrm{Error}(\varphi))$
true	*true*	*true*	*false* or *error*	*false* or *error*	*false*	*false*	*false* or *error*	*false* or *error*
true	*false*	*true*	*false* or *error*	*false* or *error*	*false*	*false* or *error*	*false* or *error*	*false* or *error*
true	*error*	*true*	*false* or *error*	*true*	*false*	*false* or *error*	*false* or *error*	*false* or *error*
false	*true*	*true*	*false* or *error*	*false* or *error*	*false* or *error*	*false*	*false* or *error*	*false* or *error*
false	*false*	*false*	*false* or *error*	*false* or *error*	*false* or *error*	*false* or *error*	*false* or *error*	*false* or *error*
false	*error*	*error*	*false* or *error*	*true*	*true*	*false* or *error*	*false* or *error*	*true*
error	*true*	*true*	*true*	*false* or *error*	*false* or *error*	*false*	*false* or *error*	*false* or *error*
error	*false*	*error*	*true*	*false* or *error*	*false* or *error*	*true*	*false* or *error*	*true*
error	*error*	*error*	*true*	*true*	*error*	*error*	*true*	*true*

Appendix B. Proof of claims

B.1. Error filter condition

Claim 1. *For every SPARQL formula φ, the formula $\mathrm{Error}(\varphi)$ can be expressed as a formula of the form $\bigvee_{\psi \in C} \psi$ where C is a non-empty set of conjunctions of formulas belonging to one of the following types:*

1. *positive or negative literals (i.e., formulas of the form* false*, $?X = a$, $\neg(?X = a)$, $\neg(?X = ?Y)$, $\mathrm{bound}(?X)$, or $\neg\,\mathrm{bound}(?X)$),*
2. *formulas φ', $\neg\varphi'$, or $\mathrm{Error}(\varphi')$ such that φ' occurs in φ and φ' is strictly smaller than φ;*

and for every mapping μ, $\mu(\varphi) =$ error *if and only if there exists a unique formula $\psi \in C$ for which $\mu(\psi) =$* true*.*

Proof. We next show this result by induction on the structure of the query.

1. If φ has the form $\mathrm{bound}(?X)$ then $\mathrm{Error}(\varphi)$ is the formula *false*. Formula φ satisfies the claim. Indeed, $C = \{\textit{false}\}$ and $\mu(\mathrm{Error}(\varphi)) = \textit{false}$ for every mapping μ because formula φ does not produce error.
2. If φ has the form $?X = a$ then $\mathrm{Error}(\varphi)$ is the formula $\neg\,\mathrm{bound}(?X)$. Formula φ satisfies the claim. Indeed, $C = \{\neg\,\mathrm{bound}(?X)\}$ and $\mu(\mathrm{Error}(\varphi)) = \textit{true}$ if and only if variable $?X$ is unbound in μ, that is the unique case when formula φ produces error.

3. If φ has the form $?X = ?Y$ then $\mathrm{Error}(\varphi)$ is the formula $\psi_1 \vee \psi_2 \vee \psi_3$ where ψ_1 is the formula $(\neg\,\mathrm{bound}(?X) \wedge \mathrm{bound}(?Y))$, ψ_2 is the formula $(\mathrm{bound}(?X) \wedge \neg\,\mathrm{bound}(?Y))$, and ψ_3 is the formula $(\neg\,\mathrm{bound}(?X) \wedge \neg\,\mathrm{bound}(?Y))$. Formula φ satisfies the claim. Indeed, $C = \{\psi_1, \psi_2, \psi_3\}$, and by construction, only one formula in C can be true, and $\mu(\mathrm{Error}(\varphi)) = \mathit{true}$ if and only if $\mu(\varphi) = \mathit{error}$.
4. If φ has the form $\neg\varphi_1$ then $\mathrm{Error}(\varphi)$ is the formula $\mathrm{Error}(\varphi_1)$. In this case $C = \{\mathrm{Error}(\varphi_1)\}$. By the induction hypothesis, $\mu(\mathrm{Error}(\varphi_1)) = \mathit{true}$ if and only if $\mu(\varphi_1) = \mathit{error}$. Because $\neg\,\mathit{error}$ is *error*, we conclude that $\mu(\mathrm{Error}(\varphi)) = \mathit{true}$ if and only if $\mu(\varphi) = \mathit{error}$. Hence, formula φ satisfies the claim.
5. If φ has the form $\varphi_1 \wedge \varphi_2$ then $\mathrm{Error}(\varphi)$ is the formula $\psi_1 \vee \psi_2 \vee \psi_3$ where ψ_1 is the formula $(\varphi_1 \wedge \mathrm{Error}(\varphi_2))$, ψ_2 is the formula $(\mathrm{Error}(\varphi_1) \wedge \varphi_2)$, and ψ_3 is the formula $(\mathrm{Error}(\varphi_1) \wedge \mathrm{Error}(\varphi_2))$. The validity of the claim for formula φ is shown in Table 8. There are three cases where $\mu(\varphi) = \mathit{error}$:

   Case ET: $\mu(\varphi_1) = \mathit{error}$ and $\mu(\varphi_2) = \mathit{true}$,
   Case TE: $\mu(\varphi_1) = \mathit{true}$ and $\mu(\varphi_2) = \mathit{error}$,
   Case EE: $\mu(\varphi_1) = \mathit{error}$ and $\mu(\varphi_2) = \mathit{error}$.

   Note that if $\mu(\varphi_1) = \mathit{error}$ and $\mu(\varphi_2) = \mathit{false}$ then $\mu(\varphi) = \mathit{false}$ (because $\mathit{error} \wedge \mathit{false}$ is *false*).
   The values in the remaining columns can be computed using the inductive hypothesis. We next present case ET as an example. The other cases follow the same reasoning. In case ET, $\mu(\varphi_1) = \mathit{error}$ and $\mu(\varphi_2) = \mathit{true}$. By the induction hypothesis, $\mu(\mathrm{Error}(\varphi_1)) = \mathit{true}$ and $\mu(\mathrm{Error}(\varphi_2))$ is either *false* or *error*.

   (a) If $\mu(\mathrm{Error}(\varphi_2)) = \mathit{false}$ then:

$$\begin{aligned}\mu(\psi_1) &= \mu(\varphi_1) \wedge \mu(\mathit{Error}(\varphi_2))\\ &= \mathit{error} \wedge \mathit{false}\\ &= \mathit{false}\,.\end{aligned}$$

$$\begin{aligned}\mu(\psi_2) &= \mu(\mathit{Error}(\varphi_1)) \wedge \mu(\varphi_2)\\ &= \mathit{true} \wedge \mathit{true}\\ &= \mathit{true}\,.\end{aligned}$$

$$\begin{aligned}\mu(\psi_3) &= \mu(\mathit{Error}(\varphi_1)) \wedge \mu(\mathrm{Error}(\varphi_2))\\ &= \mathit{true} \wedge \mathit{false}\\ &= \mathit{false}\,.\end{aligned}$$

   Hence, $\mu(\mathrm{Error}(\varphi)) = \mu(\psi_1 \vee \psi_2 \vee \psi_3) = \mathit{false} \vee \mathit{true} \vee \mathit{false} = \mathit{true}$.
   (b) If $\mu(\mathrm{Error}(\varphi_2)) = \mathit{error}$ then:

$$\begin{aligned}\mu(\psi_1) &= \mu(\varphi_1) \wedge \mu(\mathit{Error}(\varphi_2))\\ &= \mathit{error} \wedge \mathit{error}\\ &= \mathit{error}\,.\end{aligned}$$

$$\begin{aligned}\mu(\psi_2) &= \mu(\mathit{Error}(\varphi_1)) \wedge \mu(\varphi_2)\\ &= \mathit{true} \wedge \mathit{true}\\ &= \mathit{true}\,.\end{aligned}$$

$$\begin{aligned}\mu(\psi_3) &= \mu(\mathit{Error}(\varphi_1)) \wedge \mu(\mathrm{Error}(\varphi_2))\\ &= \mathit{true} \wedge \mathit{error}\\ &= \mathit{error}\,.\end{aligned}$$

   Hence, $\mu(\mathrm{Error}(\varphi)) = \mu(\psi_1 \vee \psi_2 \vee \psi_3) = \mathit{error} \vee \mathit{true} \vee \mathit{error} = \mathit{true}$.
6. If φ has the form $\varphi_1 \vee \varphi_2$ then the validity of the claim for formula φ is shown in Table 9, following the same reasoning as for the previous case where φ is a conjunction $\varphi_1 \wedge \varphi_2$.

□

B.2. Reduction of complex filter conditions

To prove the following claims, we introduce the notion of *reduction* and *reducible filter condition*. Section 6.1.2 presents three equivalences to transform a pattern with complex filter conditions into a pattern where all filter conditions are atomic. In this appendix, we show that these equivalences can be used to this end. For each equivalence $(P \,\text{FILTER}\, \varphi) \equiv P'$, we define a function that maps the filter condition φ to the set Σ_φ of filter conditions in pattern P'.

Consider the following equivalences:

$$(P \,\text{FILTER}\, \psi_1 \wedge \psi_2) \equiv ((P \,\text{FILTER}\, \psi_1) \,\text{FILTER}\, \psi_2),$$

$$\begin{aligned}(P \,\text{FILTER}\, \psi_1 \vee \psi_2) \equiv\ & (P \,\text{FILTER}\, \psi_1 \wedge \psi_2) \,\text{UNION}\\ & (P \,\text{FILTER}\, \psi_1 \wedge \neg\psi_2) \,\text{UNION}\\ & (P \,\text{FILTER}\, \neg\psi_1 \wedge \psi_2) \,\text{UNION}\\ & (P \,\text{FILTER}\, \psi_1 \wedge \text{Error}(\psi_2)) \,\text{UNION}\\ & (P \,\text{FILTER}\, \text{Error}(\psi_1) \wedge \psi_2),\end{aligned}$$

$$(P \,\text{FILTER}\, \neg\psi) \equiv ((P \,\text{EXCEPT}\, (P \,\text{FILTER}\, \psi)) \,\text{EXCEPT}\, (P \,\text{FILTER}\, \text{Error}(\psi))).$$

These three equivalences define the following functions, called *reduction rules*:

$$f_\wedge(\varphi) = \begin{cases} \{\psi_1, \psi_2\} & \text{if } \varphi \text{ has the form } \psi_1 \wedge \psi_2,\\ \{\varphi\} & \text{otherwise;}\end{cases}$$

$$f_\vee(\varphi) = \begin{cases} \{\psi_1 \wedge \psi_2,\ \psi_1 \wedge \neg\psi_2,\ \psi_1 \wedge \text{Error}(\psi_2),\ \neg\psi_1 \wedge \psi_2,\ \text{Error}(\psi_1) \wedge \psi_2\} & \text{if } \varphi \text{ has the form } \psi_1 \vee \psi_2,\\ \{\varphi\} & \text{otherwise;}\end{cases}$$

$$f_\neg(\varphi) = \begin{cases} \{\psi,\ \text{Error}(\psi)\} & \text{if } \varphi \text{ has the form } \neg\psi,\\ \{\varphi\} & \text{otherwise;}\end{cases}$$

Note that if the filter condition φ does not have the form of filter condition on the left side of the identity, we return the set $\{\varphi\}$. This captures the fact that the equivalence cannot be applied to reduce filter condition φ.

For convenience, we also define the reduction function that eliminates atomic formulas $f_\circ$ and a reduction that composes $f_\vee$ with $f_\wedge$, called $f_{\vee\wedge}$.

$$f_\circ(\varphi) = \begin{cases} \{\varphi\} & \text{if } \varphi \text{ is a complex filter condition,}\\ \emptyset & \text{if } \varphi \text{ is an atomic filter condition;}\end{cases}$$

$$f_{\vee\wedge}(\varphi) = \begin{cases} \{\psi_1, \psi_2,\ \neg\psi_1,\ \neg\psi_2,\ \text{Error}(\psi_1),\ \text{Error}(\psi_2)\} & \text{if } \varphi \text{ has the form } \psi_1 \vee \psi_2,\\ \{\varphi\} & \text{otherwise.}\end{cases}$$

For $r \in \{\wedge, \vee, \neg, \circ, \vee\wedge\}$, let F_r be the function that receives a set of filter conditions Σ and returns the set of filter conditions $F_r(\Sigma) = \bigcup_{\varphi\in\Sigma} f_r(\varphi)$. Given two sets of filter conditions Σ_1 and Σ_2 we write $\Sigma_1 \xrightarrow{r} \Sigma_2$ if $F_r(\Sigma_1) = \Sigma_2$. In this case, we say that $\Sigma_1 \xrightarrow{r} \Sigma_2$ is a *reduction*. We said that a filter condition φ is reducible if there is a finite sequence of reductions $\{\varphi\} \xrightarrow{r_1} \Sigma_1 \xrightarrow{r_2} \cdots \xrightarrow{r_n} \emptyset$. Intuitively, reductions are applied until all complex filter conditions are eliminated. It is not difficult to see that we can apply the aforementioned equivalences to transform every pattern P_1 to a pattern P_2 with no complex formulas if and only if every filter condition φ is reducible.

We next prove that every filter condition is reducible by induction on the structure of the filter condition. For this induction, we define the components of a filter condition φ, denoted $\text{comp}(\varphi)$, to be the set of filter conditions defined as follows: If φ is atomic, then $\text{comp}(\varphi) = \emptyset$; if $\varphi = \psi_1 \vee \psi_2$ or $\varphi = \psi_1 \wedge \psi_2$, then $\text{comp}(\varphi) = \{\psi_1, \psi_2\} \cup \text{comp}(\psi_1) \cup \text{comp}(\psi_2)$; and if $\varphi = \neg\psi$ then $\text{comp}(\varphi) = \{\psi\} \cup \text{comp}(\psi)$.

Claim 2. *Every filter condition φ is reducible.*

Proof. We prove this by induction using the following hypothesis: if φ is a filter condition where for each filter condition $\psi \in \mathrm{comp}(\varphi)$, ψ and $\mathrm{Error}(\psi)$ are reducible, then the filter conditions φ and $\mathrm{Error}(\varphi)$ are reducible.

1. If φ is $\mathrm{bound}(?X)$ then

$$\{\varphi\} \xrightarrow{\circ} \emptyset,$$
$$\{\mathrm{Error}(\varphi)\} = \{\mathit{false}\} \xrightarrow{\circ} \emptyset.$$

2. If φ is $?X = a$ then

$$\{\varphi\} \xrightarrow{\circ} \emptyset,$$
$$\{\mathrm{Error}(\varphi)\} = \{\neg\,\mathrm{bound}(?X)\} \xrightarrow{\neg} \{\mathrm{bound}(?X),\ \mathrm{Error}(\neg\,\mathrm{bound}(?X))\} = \{\mathrm{bound}(?X),\ \mathit{false}\} \xrightarrow{\circ} \emptyset.$$

3. If φ is $?X = ?Y$ then

$$\begin{aligned}
&\{\varphi\} \xrightarrow{\circ} \emptyset,\\
&\{\mathrm{Error}(\varphi)\} = \{\neg\,\mathrm{bound}(?X) \vee \neg\,\mathrm{bound}(?Y)\}\\
&\qquad \xrightarrow{\vee\wedge} \{\,\mathrm{bound}(?X),\ \mathrm{bound}(?Y),\ \neg\,\mathrm{bound}(?X),\ \neg\,\mathrm{bound}(?Y),\\
&\qquad\qquad \mathrm{Error}(\neg\,\mathrm{bound}(?X)),\ \mathrm{Error}(\neg\,\mathrm{bound}(?Y))\}\\
&\qquad = \{\mathrm{bound}(?X),\ \mathrm{bound}(?Y),\ \neg\,\mathrm{bound}(?X),\ \neg\,\mathrm{bound}(?Y),\ \mathit{false}\}\\
&\qquad \xrightarrow{\circ} \{\neg\,\mathrm{bound}(?X),\ \neg\,\mathrm{bound}(?Y)\}\\
&\qquad \xrightarrow{\neg} \{\mathrm{bound}(?X),\ \mathrm{Error}(\mathrm{bound}(?X)),\ \mathrm{bound}(?Y),\ \mathrm{Error}(\mathrm{bound}(?Y))\}\\
&\qquad = \{\mathrm{bound}(?X),\ \mathit{false},\ \mathrm{bound}(?Y),\ \mathit{false}\}\\
&\qquad \xrightarrow{\circ} \emptyset.
\end{aligned}$$

4. If φ is $\neg\psi_1$ then

$$\{\varphi\} \xrightarrow{\neg} \{\psi,\ \mathrm{Error}(\psi)\},$$
$$\{\mathrm{Error}(\varphi)\} = \{\mathrm{Error}(\psi)\}.$$

Since $\psi \in \mathrm{comp}(\varphi)$ and by inductive hypothesis, the filter conditions ψ and $\mathrm{Error}(\psi)$ are reducible. Hence, the filter conditions φ and $\mathrm{Error}(\varphi)$ are reducible.

5. If φ is $\psi_1 \wedge \psi_2$ then

$$\begin{aligned}
&\{\varphi\} \xrightarrow{\wedge} \{\psi_1,\ \psi_2\},\\
&\{\mathrm{Error}(\varphi)\} = \{\mathrm{Error}(\psi) \vee \mathrm{Error}(\psi_2)\}\\
&\qquad \xrightarrow{\vee\wedge} \{\,\mathrm{Error}(\psi_1),\ \mathrm{Error}(\psi_2),\\
&\qquad\qquad \neg\,\mathrm{Error}(\psi_1),\ \neg\,\mathrm{Error}(\psi_2),\\
&\qquad\qquad \mathrm{Error}(\mathrm{Error}(\psi_1)),\ \mathrm{Error}(\mathrm{Error}(\psi_2))\}\\
&\qquad = \{\mathrm{Error}(\psi_1),\ \mathrm{Error}(\psi_2),\ \neg\,\mathrm{Error}(\psi_1),\ \neg\,\mathrm{Error}(\psi_2),\ \mathit{false}\}.
\end{aligned}$$

Since $\psi_1, \psi_2 \in \mathrm{comp}(\varphi)$ and by the induction hypothesis, the filter conditions $\mathrm{Error}(psi_1)$ and $\mathrm{Error}(\psi_2)$ are reducible. To show that φ is reducible, we have to show that $\neg\,\mathrm{Error}(\psi_1)$ and $\neg\,\mathrm{Error}(\psi_2)$ are reducible.

$$\{\neg\,\mathrm{Error}(\psi_1)\} \xrightarrow{\neg} \{\mathrm{Error}(\psi),\ \mathrm{Error}(\mathrm{Error}(\psi_2))\} = \{\mathrm{Error}(\psi),\ false\}.$$

By the induction hypothesis, $\mathrm{Error}(\psi)$ is reducible. Hence, $\neg\,\mathrm{Error}(\psi_1)$ is reducible. Similarly, $\neg\,\mathrm{Error}(\psi_1)$ is reducible. Then, $\mathrm{Error}(\varphi)$ is reducible.

6. Let φ be $\psi_1 \vee \psi_2$. First, we show that φ is reducible.

$$\{\varphi\} \xrightarrow{\vee\wedge} \{\psi_1, \psi_2, \neg\psi_1, \neg\psi_2, \mathrm{Error}(\psi_1), \mathrm{Error}(\psi_2)\}$$

By the induction hypothesis on ψ_1 and ψ_2, ψ_1, ψ_2, $\mathrm{Error}(\psi_1)$, and $\mathrm{Error}(\psi_2)$ are reducible. To prove that φ is reducible, suffices to prove that $\neg\psi_1$ and $\neg\psi_2$ are reducible.

$$\{\neg\psi_1\} \xrightarrow{\neg} \{\psi_1, \mathrm{Error}(\psi_1)\}.$$

By the induction hypothesis in ψ_1, ψ_1 and $\mathrm{Error}(\psi_1)$ are reducible. Hence, $\neg\psi_1$ is reducible. Similarly, $\neg\psi_2$ is reducible. Hence, φ is reducible.
Second, we show that $\mathrm{Error}(\varphi)$ is reducible.

$$\{\mathrm{Error}(\varphi)\} = \{\mathrm{Error}(\psi_1) \wedge \mathrm{Error}(\psi_2)\} \xrightarrow{\wedge} \{\mathrm{Error}(\psi_1),\ \mathrm{Error}(\psi_2)\}.$$

By the induction hypothesis in ψ_1 and ψ_2, $\mathrm{Error}(\psi_1)$ and $\mathrm{Error}(\psi_2)$ are reducible. Hence, $\mathrm{Error}(\varphi)$ is reducible.

Hence, for every filter condition φ, the filter conditions φ and $\mathrm{Error}(\varphi)$ are reducible. □

B.3. Normalization of NRMD$^\neg$ queries

Claim 3 (Normalized NRMD$^\neg$)**.** *Let $(p(\bar{X}), \Pi)$ be a NRMD$^\neg$ query, and R be a rule in Π with form*

$$p(\bar{X}) \leftarrow A_1, \ldots, A_m, \neg B_1, \ldots, \neg B_n,$$

where $A_1, \ldots, A_m$ are positive literals, and $\neg B_1, \ldots, \neg B_n$ are negative literals. For $1 \leqslant i \leqslant m$, let $\bar{Y}_i$ be the set of variables that consists of the variables atoms $A_1, \ldots, A_i$. Consider the minimal set of rules Π_R that includes the following rules:

1. *Rules R_i^A, for $2 \leqslant i \leqslant m$, defined recursively as follows:*
   (a) $R_2^A = q_2^A(\bar{Y}_2) \leftarrow A_1, A_2$.
   (b) $R_i^A = q_i^A(\bar{Y}_i) \leftarrow q_{i-1}^A(\bar{Y}_{i-1}), A_i$.
2. *Rules R_j^B for $1 \leqslant j \leqslant n$, defined recursively as follows:*
   (a) $R_0^B = r_0^B(\bar{Y}_m) \leftarrow q_m^A(\bar{Y}_m)$,
   (b) $R_j^B = r_j^B(\bar{Y}_m) \leftarrow r_{j-1}^B(\bar{Y}_m), \neg B_j$,
3. *A rule $R' = p(\bar{X}) \leftarrow r_n^B(\bar{Y}_m)$.*

The NRMD$^\neg$ query $(p(\bar{X}), \Pi')$ that results from replacing rule R in query $(p(\bar{X}), \Pi)$ with the rules in Π_R is equivalent to query $(p(\bar{X}), \Pi)$.

Proof. We next prove this claim by induction on the numbers *m*, of positive literals, and *n*, of negative literals, in a rule *R*. The hypothesis of induction states that the query $(q(\bar{X}), \{R\})$ and its normalized query $(q(\bar{X}), \Pi_R)$ are equivalent. Since we assumed that every literal in the body of a rule must have at least one variable (see Section 4), to guarantee safeness, the body of the rule cannot include a negative literal without having at least a positive literal.

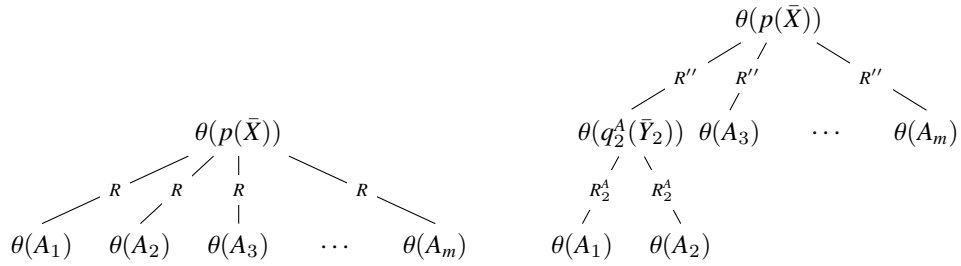

Fig. 4. Derivation trees for the ground literal $\theta(p(\bar{X}))$ regarding query $(p(\bar{X}), \{R\})$ (on the left), and query $(p(\bar{X}), \{R_2^A, R''\})$ (on the right). The children of the nodes labeled with the positive ground literals $\theta(A_i)$ are omitted.

1. If $m = 1$ and $n = 0$, rule R is already normalized because it is the projection rule $p(\bar{X}) \leftarrow A_1$.
2. If $m > 1$ and $n = 0$ then the normalization of rule R consists of a set Π_R of rules R_i^A, for $2 \leqslant i \leqslant m$, defined recursively as follows:

$$
\begin{aligned}
R_2^A &= q_2^A(\bar{Y}_2) \leftarrow A_1, A_2, \\
R_i^A &= q_i^A(\bar{Y}_i) \leftarrow q_{i-1}^A(\bar{Y}_{i-1}), A_i && \text{for } 2 \leqslant i \leqslant m, \\
R_0^B &= r_0(\bar{Y}_m) \leftarrow q_n^A(\bar{Y}_m), \\
R' &= p(\bar{X}) \leftarrow r_0^B(\bar{Y}_m).
\end{aligned}
$$

   By the induction hypothesis, the query $(p(\bar{X}), \{R_3^A, \ldots, R_m^A, R_0^B, R'\})$ is equivalent to the query $(p(\bar{X}), \{R''\})$ where R'' is the rule $p(\bar{X}) \leftarrow q_2^A(\bar{Y}_2), A_3, \ldots, A_m$. Hence, the query $(p(\bar{X}), \Pi_R)$ is equivalent to the query $(p(\bar{X}), \{R_2^A, R''\})$. To show that these queries are equivalent to query $(p(\bar{X}), \{R\})$, we need to show that they have the same answers, and each answer has the same multiplicities.
   Assume that a substitution θ is an answer to query $(p(\bar{X}), \{R\})$. Then, program $\{R\}$ has a derivation tree whose root is labeled with the ground literal $\theta(p(\bar{X}))$, has m children labeled with the ground literals $\theta(A_i)$, for $1 \leqslant i \leqslant m$, and the edges from the root to the children are labeled with rule R, as is shown in Figure 4 (on the left). Then, for $1 \leqslant i \leqslant m$, there is a derivation three whose root is labeled with the ground literal $\theta(A_i)$. The existence of the ground literals $\theta(A_i)$ as labels of derivation tree roots proves that the ground literal $\theta(p(\bar{X}))$ is inferred using the rules R_2^A and R'', as is shown in the Figure 4 (on the right). Then, if θ is an answer to query $(p(\bar{X}), \{R\})$ then θ is an answer to query $(p(\bar{X}), \{R_2^A, R''\})$. The same argument can be used in the contrary direction to prove that if θ is an answer to query $(p(\bar{X}), \{R_2^A, R''\})$ then θ is an answer to query $(p(\bar{X}), \{R\})$. Finally, the multiplicity of $\theta(p(\bar{X}))$ is, for both queries, the product of the multiplicities of $\theta(A_i)$, for $1 \leqslant i \leqslant m$. Hence, both queries are equivalent.
3. If $m > 1$ and $n > 0$ then the normalization of rule R consists of a set Π_R of rules R_i^A, for $2 \leqslant i \leqslant m$, defined recursively as follows:

$$
\begin{aligned}
R_2^A &= q_2^A(\bar{Y}_2) \leftarrow A_1, A_2, \\
R_i^A &= q_i^A(\bar{Y}_i) \leftarrow q_{i-1}^A(\bar{Y}_{i-1}), A_i && \text{for } 2 \leqslant i \leqslant m, \\
R_0^B &= r_0^B(\bar{Y}_m) \leftarrow q_m^A(\bar{Y}_m), \\
R_j^B &= r_j^B(\bar{Y}_m) \leftarrow r_{j-1}^B(\bar{Y}_m), \neg B_j && \text{for } 1 \leqslant j \leqslant n, \\
R' &= p(\bar{X}) \leftarrow r_n^B(\bar{Y}_m).
\end{aligned}
$$

   The rules above are equivalent to the following rules:

$$
R_2^A = q_2^A(\bar{Y}_2) \leftarrow A_1, A_2,
$$

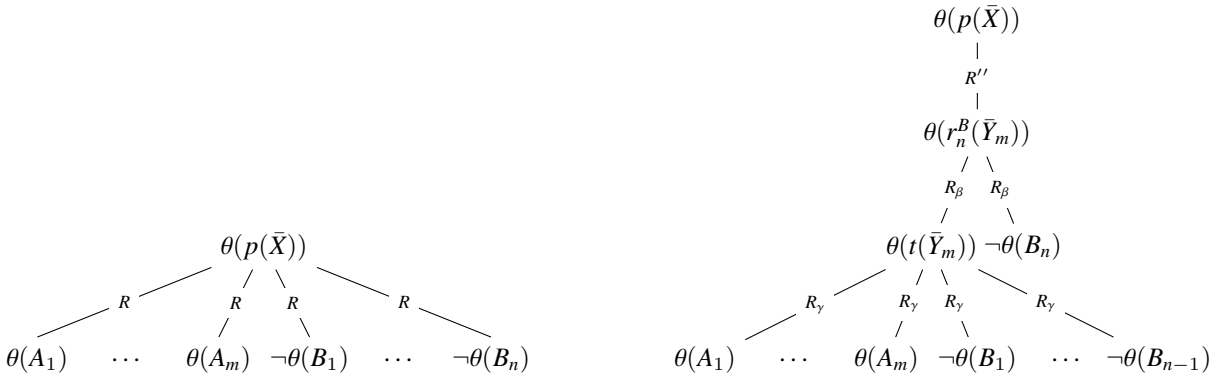

Fig. 5. Derivation trees for the ground atom $\theta(p(\bar{X}))$ regarding query $(p(\bar{X}),\{R\})$ (on the left), and query $(p(\bar{X}),\{R_2^A,R''\})$ (on the right). The children of the nodes labeled with the positive ground literals $\theta(A_i)$ are omitted. Nodes label with the negative ground literals $\neg\theta(B_j)$ have no children and do no derivation tree include the positive literal $\theta(B_j)$ as the root label.

$$
\begin{aligned}
R_i^A &= q_i^A(\bar{Y}_i) \leftarrow q_{i-1}^A(\bar{Y}_{i-1}), A_i && \text{for } 2 \leqslant i \leqslant m,\\
R_0^B &= r_0^B(\bar{Y}_m) \leftarrow q_m^A(\bar{Y}_m),\\
R_j^B &= r_j^B(\bar{Y}_m) \leftarrow r_{j-1}^B(\bar{Y}_m), \neg B_j && \text{for } 1 \leqslant j \leqslant n-1,\\
R_\alpha &= t(\bar{Y}_m) \leftarrow r_{n-1}^B(\bar{Y}_m),\\
R_\beta &= r_n^B(\bar{Y}_m) \leftarrow t(\bar{Y}_m), \neg B_n,\\
R'' &= p(\bar{X}) \leftarrow r_n^B(\bar{Y}_m).
\end{aligned}
$$

By the induction hypothesis, the query $(t(\bar{Y}_m),(\Pi_R \cup \{R_\alpha\}) \setminus \{R_n^B,R'\})$ is equivalent to the query $(t(\bar{Y}_m),\{R_\gamma\})$ where R_γ is the rule $t(\bar{Y}_m) \leftarrow A_1,\ldots,A_m,B_1,\ldots,B_{n-1}$. Hence, the query $(p(\bar{X}),\Pi_R)$ is equivalent to the query $(p(\bar{X}),\{R_\gamma,R_\beta,R''\})$ To show that these queries are equivalent to query $(p(\bar{X}),\{R\})$, we need to show that they have the same answers, and each answer has the same multiplicities.

Assume that a substitution θ is an answer to query $(p(\bar{X}),\{R\})$. Then, program $\{R\}$ has a derivation tree whose root is labeled with the ground literal $\theta(p(\bar{X}))$, has m children labeled with the ground literals $\theta(A_i)$, and n children labeled with literals $\neg\theta(B_j)$, for $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant n$, and the edges from the root to the children are labeled with rule R, as is shown in Figure 5 (on the left). Then, for $1 \leqslant i \leqslant m$, there is a derivation three whose root is labeled with the ground literal $\theta(A_i)$, and for $1 \leqslant j \leqslant n$, there is no derivation three whose root is labeled with the ground literal $\theta(B_j)$. The existence of the ground literals $\theta(A_i)$ and the non-existence of the ground literals $\theta(B_j)$ as labels of derivation tree roots prove that the ground literal $\theta(p(\bar{X}))$ is inferred using the rules R_γ, R_β, and R'', as is shown in the Figure 5 (on the right). Then, if θ is an answer to query $(p(\bar{X}),\{R\})$ then θ is an answer to query $(p(\bar{X}),\{R_\gamma,R_\beta,R''\})$. The same argument can be used in the contrary direction to prove that if θ is an answer to query $(p(\bar{X}),\{R_\gamma,R_\beta,R''\})$ then θ is an answer to query $(p(\bar{X}),\{R\})$. Finally, the multiplicity of $\theta(p(\bar{X}))$ is, for both queries, the product of the multiplicities of $\theta(A_i)$, for $1 \leqslant i \leqslant m$. Hence, both queries are equivalent.

Hence, we have proved that the normalization method produces an equivalent NRMD$^\neg$ query. □

B.4. Simulations between query languages

Claim 4 (SPARQL to NRMD$^\neg$)**.** *The triple* $(f_{1,2},g_{1,2},h_{2,1})$ *is a simulation of SPARQL in NRMD*$^\neg$.

Proof. To prove this claim, we show that, for every SPARQL query Q and RDF graph G, it holds that $[\![Q]\!]_G = h_{2,1}([\![f_{1,2}(Q)]\!]_{g_{1,2}})$ by induction on the structure of a normalized SPARQL query Q. In this proof, we assume that

θ is a NRMD$^\neg$ substitution for the variables of the NRMD$^\neg$ query $f_{1,2}(Q)$, and μ is the SPARQL mapping $h_{2,1}(\theta)$. To show then that $[\![Q]\!]_G = h_{2,1}([\![f_{1,2}(Q)]\!]_{g_{1,2}})$, we have to prove that $\mu \in [\![Q]\!]_G$ if and only if $\theta \in h_{2,1}([\![f_{1,2}(Q)]\!]_{g_{1,2}})$, and $\mathrm{card}(\mu, [\![Q]\!]_G) = \mathrm{card}(\theta, h_{2,1}([\![f_{1,2}(Q)]\!]_{g_{1,2}}))$.

1. Let Q be a triple pattern $(?X, p, ?Y)$. In this case, there is a corresponding version of the triple pattern as a NRMD$^\neg$ literal $\mathrm{triple}(X, p, Y)$, where X and Y are the corresponding variables for $?X$ and $?Y$. The NRMD$^\neg$ query $f_{1,2}(Q)$ is then $(q(X,Y), \Pi)$ where Π is the program with a rule $q(X,Y) \leftarrow \mathrm{triple}(X, p, Y)$. Let θ be the NRMD$^\neg$ substitution $\theta = (X/s,\ Y/o)$ and μ be the SPARQL mapping $h_{2,1}(\theta) = \{?X \mapsto s,\ ?Y \mapsto o\}$.
   (a) According to the NRMD$^\neg$ semantics, $\theta \in [\![f_{1,2}(Q)]\!]_{g_{1,2}(G)}$ if and only if $\mathrm{triple}(s,p,o) \in g_{1,2}(G)$. By the definition function $g_{1,2}$, $\mathrm{triple}(s,p,o) \in g_{1,2}(G)$ if and only if $(s,p,o) \in G$. By the SPARQL semantics, the SPARQL mapping μ is in $[\![Q]\!]_G$ if and only if $(s,p,o) \in G$. Hence, $\theta \in [\![f_{1,2}(Q)]\!]_{g_{1,2}(G)}$ if and only if $\mu \in [\![Q]\!]_G$.
   (b) By construction, $\mathrm{card}(\theta, [\![f_{1,2}(Q)]\!]_{g_{1,2}(G)}) = 1$ and $\mathrm{card}(\mu, [\![Q]\!]_G) = 1$. Hence, $\mathrm{card}(\theta, [\![f_{1,2}(Q)]\!]_{g_{1,2}(G)}) = \mathrm{card}(\mu, [\![Q]\!]_G)$.

   We have shown that we can simulate triple patterns of the form (X,p,Y) with NRMD$^\neg$ queries. However, it is not difficult to apply the same argument for the other forms of triple patterns (e.g., (X,p,o) or (s,X,Y)). Hence, SPARQL triple patterns are simulable with NRMD$^\neg$.
2. Let Q be a query $(P_1 \,\mathrm{AND}\, P_2)$. Assume that $\mathrm{inScope}(P_1) = \{?X, ?Y\}$ and $\mathrm{inScope}(P_2) = \{?X, ?Z\}$. The NRMD$^\neg$ query $f_{1,2}(Q)$ is then $(q(X,Y,Z), \Pi)$ where Π is the program that consists of the rules in the programs of queries $(p_1(X,Y), \Pi_1) = f_{1,2}(P_1)$ and $(p_2(X,Z), \Pi_2) = f_{1,2}(P_2)$, the rule

   $$q(X,Y,Z) \leftarrow p_1(X_1, Y), p_2(X_2, Z), \mathrm{comp}(X_1, X_2, X)$$

   and the rules that define the compatibility between values (which may also included in Π_1 and Π_2)

   $$\mathrm{comp}(X,X,X) \leftarrow \mathrm{term}(X)$$
   $$\mathrm{comp}(X,Y,X) \leftarrow \mathrm{term}(X), \mathrm{null}(Y)$$
   $$\mathrm{comp}(Y,X,X) \leftarrow \mathrm{term}(X), \mathrm{null}(Y)$$
   $$\mathrm{comp}(Y,Y,Y) \leftarrow \mathrm{null}(Y).$$

   (a) If $\theta \in [\![f_{1,2}(Q)]\!]_{g_{1,2}(G)}$ then, by the semantics of NRMD$^\neg$, there exists the NRMD$^\neg$ solutions $\theta_1 = \{X_1/a_1, Y/b\}$, $\theta_2 = \{X_2/a_2, Z/c\}$, and $\theta_3 = \{X_1/a_1, X_2/a_2, X/a\}$ such that

   $$\{X_1/a_1, Y/b\} \in [\![(p_1(X_1,Y), \Pi_1)]\!]_{g_{1,2}(G)},$$
   $$\{X_2/a_2, Z/c\} \in [\![(p_2(X_2,Z), \Pi_2)]\!]_{g_{1,2}(G)},$$
   $$\{X_1/a_1, X_2/a_2, X/a\} \in [\![(\mathrm{comp}(X_1,X_2,X), \Pi)]\!]_{g_{1,2}(G)}.$$

   By the induction hypothesis in P_1 and P_2, $\theta_1 \in \{X_1/a_1, Y/b\} \in [\![(p_1(X_1,Y), \Pi_1)]\!]_{g_{1,2}(G)}$ and $\theta_2 \in \{X_1/a_1, Y/b\} \in [\![(p_2(X_1,Y), \Pi_2)]\!]_{g_{1,2}(G)}$ if and only if mappings $\mu_1 = h_{2,1}(\theta_1)$ and $\mu_2 = h_{2,1}(\theta_2)$ hold $\mu_1 \in [\![P_1]\!]_G$ and $\mu_2 \in [\![P_2]\!]_G$. By the rules defining comp, it holds that $\mu_1 \sim \mu_2$ and $\mu_1 \cup \mu_2 = \mu$. By the semantics of the SPARQL operator AND, this it holds that $\mu \in [\![Q]\!]_G$. Hence, $\theta \in [\![f_{1,2}(Q)]\!]_{g_{1,2}(G)}$ if and only if $\mu \in [\![Q]\!]_G$.
   (b) By definition,

   $$\mathrm{card}(\mu, [\![Q]\!]_G) = \sum_{\substack{\mu_1 \in [\![P_1]\!]_G \\ \mu_2 \in [\![P_2]\!]_G \\ \mu_1 \sim \mu_2 \\ \mu = \mu_1 \cup \mu_2}} \mathrm{card}(\mu_1, [\![P_1]\!]_G) \times \mathrm{card}(\mu_2, [\![P_2]\!]_G).$$

By the induction hypothesis,

$$\mathrm{card}(\mu, [\![Q]\!]_G) = \sum_{\substack{\{X_1/a_1, Y/b\} \in [\![(p_1(X_1,Y),\Pi_1)]\!]_{g_{1,2}(G)} \\ \{X_2/a_2, Z/c\} \in [\![(p_2(X_2,Y),\Pi_2)]\!]_{g_{1,2}(G)} \\ \{X_1/a_1, X_2/a_2, X/a\} \in [\![(\mathrm{comp}(X_1,X_2,X),\Pi)]\!]_{g_{1,2}(G)}}} \mathrm{card}(\{X_1/a_1, Y/b\}, [\![f_{1,2}(P_1)]\!]_{g_{1,2}(G)}) \times \mathrm{card}(\{X_2/a_2, Z/c\}, [\![f_{1,2}(P_1)]\!]_{g_{1,2}(G)}).$$

By the semantics of NRMD$^\neg$, we conclude that $\mathrm{card}(\mu, [\![Q]\!]_G) = \mathrm{card}(\theta, [\![f_{1,2}(Q)]\!]_{g_{1,2}(G)})$.

We have shown that we can simulate queries of the form $(P_1 \text{ AND } P_2)$, where $\mathrm{inScope}(P_1) = \{?X, ?Y\}$ and $\mathrm{inScope}(P_2) = \{?X, ?Z\}$, with NRMD$^\neg$ queries. However, it is not difficult to apply the same argument for queries where P_1 and P_2 have different sets of in-scope variables. Hence, SPARQL queries of the form $(P_1 \text{ AND } P_2)$ are simulable with NRMD$^\neg$.

3. Let Q be a query $(P_1 \text{ EXCEPT } P_2)$, and $?\bar{X}$ be the list of SPARQL variables in set $\mathrm{inScope}(Q)$. The NRMD$^\neg$ query $f_{1,2}(Q)$ is then $(q(\bar{X}), \Pi)$ where Π is the program that consists of the rules in programs of queries $(p_1(\bar{X}), \Pi_1) = f_{1,2}(P_1)$ and $(p_2(\bar{X}), \Pi_2) = f_{1,2}(P_2)$, and the rule $q(\bar{X}) \leftarrow p_1(\bar{X}), \neg p_2(\bar{X})$.
   (a) By the semantics of NRMD$^\neg$, $\theta \in [\![f_{1,2}(Q)]\!]_{g_{1,2}(G)}$ if and only if $\theta \in [\![f_{1,2}(P_1)]\!]_{g_{1,2}(G)}$ and $\theta \notin [\![f_{1,2}(P_2)]\!]_{g_{1,2}(G)}$. By the induction hypothesis, the last condition is equivalent to $\mu \in [\![P_1]\!]_G$ and $\mu \notin [\![P_2]\!]_G$. By the SPARQL semantics, this is equivalent to $\mu \in [\![Q]\!]_G$.
   (b) By definition, $\mathrm{card}(\mu, [\![Q]\!]_G) = \mathrm{card}(\mu, [\![P_1]\!]_G)$ and $\mathrm{card}(\theta, [\![f_{1,2}(Q)]\!]_{g_{1,2}(G)}) = \mathrm{card}(\theta, [\![f_{1,2}(P_1)]\!]_{g_{1,2}(G)})$. By the induction hypothesis, $\mathrm{card}(\mu, [\![P_1]\!]_G) = \mathrm{card}(\theta, [\![f_{1,2}(P_1)]\!]_{g_{1,2}(G)})$. Hence, $\mathrm{card}(\mu, [\![Q]\!]_G) = \mathrm{card}(\theta, [\![f_{1,2}(Q)]\!]_{g_{1,2}(G)})$.

   Hence, SPARQL queries of the form $(P_1 \text{ EXCEPT } P_2)$ are simulable with NRMD$^\neg$.
4. Let Q be a SPARQL query $(P_1 \text{ UNION } P_2)$. The NRMD$^\neg$ query $f_{1,2}(Q)$ is then $(q(\bar{X}), \Pi)$ where $\bar{X}$ is the list with the variables in set $\mathrm{inScope}(Q)$, and Π is the program that consists of the rules in program of queries $(p_1(\bar{X}), \Pi_1) = f_{1,2}(P_1)$ and $(p_2(\bar{X}), \Pi_2) = f_{1,2}(P_2)$, and the rules that correspond the operation UNION, namely $q(\bar{X}) \leftarrow p_1(\bar{X})$ and $q(\bar{X}) \leftarrow p_2(\bar{X})$.
   (a) By the NRMD$^\neg$ semantics, θ is a solution of $(q(\bar{X}), \Pi)$ if and only if $\theta \in [\![(p_1(\bar{X}), \Pi_1)]\!]_{g_{1,2}(G)}$ or $\theta \in [\![(p_2(\bar{X}), \Pi_2)]\!]_{g_{1,2}(G)}$. By the induction hypothesis, this is equivalent to that $\mu \in [\![P_1]\!]_G$ or $\mu \in [\![P_1]\!]_G$. By the SPARQL semantics, this is equivalent to $\mu \in [\![Q]\!]_G$.
   (b) By the NRMD$^\neg$ semantics, $\mathrm{card}(\theta, [\![f_{1,2}(Q)]\!]_{g_{1,2}(G)}) = \mathrm{card}(\theta, [\![f_{1,2}(P_1)]\!]_{g_{1,2}(G)}) + \mathrm{card}(\theta, [\![f_{1,2}(P_2)]\!]_{g_{1,2}(G)})$ and $\mathrm{card}(\theta, [\![f_{1,2}(Q)]\!]_{g_{1,2}(G)}) = \mathrm{card}(\mu, [\![P_1]\!]_G) + \mathrm{card}(\mu, [\![P_2]\!]_G)$.
   By the induction hypothesis, $\mathrm{card}(\theta, [\![f_{1,2}(P_1)]\!]_{g_{1,2}(G)}) = \mathrm{card}(\mu, [\![P_1]\!]_G)$ and $\mathrm{card}(\theta, [\![f_{1,2}(P_2)]\!]_{g_{1,2}(G)}) = \mathrm{card}(\mu, [\![P_2]\!]_G)$. Hence $\mathrm{card}(\theta, [\![f_{1,2}(Q)]\!]_{g_{1,2}(G)}) = \mathrm{card}(\mu, [\![Q]\!]_G)$.

   Hence, SPARQL queries of the form $(P_1 \text{ UNION } P_2)$ are simulable with NRMD$^\neg$.
5. Let Q be the SPARQL query $(P \text{ FILTER } \varphi)$ where is an atomic filter condition (i.e., a filter condition of the form $?X = c$, $?X = ?Y$, or $\mathrm{bound}(?X)$), and L_φ be a set of NRMD$^\neg$ literals defined as follows:

$$L_\varphi = \begin{cases} X = c, \mathrm{bound}(X) & \text{if } \varphi \text{ is } ?X = c, \\ X = Y, \mathrm{bound}(X), \mathrm{bound}(Y) & \text{if } \varphi \text{ is } ?X = ?Y, \\ \mathrm{bound}(X) & \text{if } \varphi \text{ is } \mathrm{bound}(?X). \end{cases}$$

   The NRMD$^\neg$ query $f_{1,2}(Q)$ is then $(q(\bar{X}), \Pi)$ where $\bar{X}$ is the list with the variables in set $\mathrm{inScope}(Q)$, and Π is the program that consists of the rules in program of query $(p(\bar{X}), \Pi') = f_{1,2}(P)$, and the rule that corresponds the operation FILTER, namely rule $q(\bar{X}) \leftarrow p(\bar{X}), L_\varphi$.
   (a) By the NRMD$^\neg$ semantics, θ is a solution of $(q(\bar{X}), \Pi)$ if and only if $\theta \in [\![(p(\bar{X}), \Pi')]\!]_{g_{1,2}(G)}$, and $\theta(L_\varphi) \subseteq g_{1,2}(G)$. By the induction hypothesis, $\theta \in [\![(p(\bar{X}), \Pi')]\!]_{g_{1,2}(G)}$ is equivalent to $\mu \in [\![P]\!]_G$. By construction, $\theta(L_\varphi) \subseteq \mathrm{atoms}(\Pi', g_{1,2}(G))$ if and only if $\mu(\varphi) = \mathit{true}$. By the SPARQL semantics, this is equivalent to $\mu \in [\![Q]\!]_G$.

(b) By construction, every fact in $\theta(L_\varphi)$ occurs once in $g_{1,2}(G)$. For each fact in $F \in \theta(L_\varphi)$ there is then only one proof that $F \in \mathrm{atoms}(\Pi, g_{1,2}(G))$. Hence, $\mathrm{card}(\theta, [\![f_{1,2}(Q)]\!]_{g_{1,2}(G)}) = \mathrm{card}(\theta, [\![f_{1,2}(P)]\!]_{g_{1,2}(G)})$. By the induction hypothesis, $\mathrm{card}(\theta, [\![f_{1,2}(P)]\!]_{g_{1,2}(G)}) = \mathrm{card}(\mu, [\![P]\!]_G)$. According to the SPARQL semantics, $\mathrm{card}(\mu, [\![P]\!]_G) = \mathrm{card}(\mu, [\![Q]\!]_G)$. Hence, $\mathrm{card}(\theta, [\![f_{1,2}(Q)]\!]_{g_{1,2}(G)}) = \mathrm{card}(\mu, [\![Q]\!]_G)$.

Hence, SPARQL queries of the form $(P \,\mathrm{FILTER}\, \varphi)$ are simulable with NRMD$^\neg$.

6. Let Q be the SPARQL query $(\mathrm{SELECT}\, \bar{X}\ P)$. The NRMD$^\neg$ query $f_{1,2}(Q)$ is then $(q(\bar{X}), \Pi)$, where Π is the program that consists of the rules in the program of query $(p(\bar{Y}, \Pi') = f_{1,2}(P)$, and the rule that corresponds to the operation projects, namely rule $q(\bar{X}) \leftarrow p(\bar{Y}), \mathrm{null}(x_1), \ldots, \mathrm{null}(x_n)$, where $x_1, \ldots, x_n$ are the variables that are in W but not in $\mathrm{inScope}(P_1)$.

(a) By the NRMD$^\neg$ semantics, θ is a solution of $(q(\bar{X}), \Pi)$ if and only if there exists a solution $\theta' \in [\![(p(\bar{Y}, \Pi')]\!]_{g_{1,2}(G)}$ such that $\theta(x) = \theta'(x)$ if $x \in \mathrm{inScope}(Q) \cap \mathrm{inScope}(P)$. Let $\mu = h_{2,1}(\theta)$ and $\mu' = h_{2,1}(\theta')$. By construction $\mu = \mu'|_{\mathrm{inScope}(Q)}$. By the induction hypothesis, $\mu' \in [\![P]\!]_G$. Hence, $\mu \in [\![Q]\!]_G$.

(b) By construction,

$$\mathrm{card}(\theta, [\![f_{1,2}(Q)]\!]_{g_{1,2}(G)}) = \sum_{\substack{\theta'|_{\mathrm{inScope}(Q)} = \theta \\ \theta' \in [\![f_{1,2}(P)]\!]_{g_{1,2}(G)}}} \mathrm{card}(\theta', [\![f_{1,2}(P)]\!]_{g_{1,2}(G)}).$$

By the induction hypothesis,

$$\mathrm{card}(\theta, [\![f_{e1,2}(Q)]\!]_{g_{1,2}(G)}) = \sum_{\substack{\theta'|_{\mathrm{inScope}(Q)} = \theta \\ \mu' = h_{2,1}(\theta') \\ \mu' \in [\![P]\!]_G}} \mathrm{card}(\mu', [\![P]\!]_G).$$

By construction,

$$\mathrm{card}(\theta, [\![f_{e1,2}(Q)]\!]_{g_{1,2}(G)}) = \sum_{\substack{\mu'|_{\mathrm{inScope}(Q)} = h_{2,1}(\theta) \\ \mu' \in [\![P]\!]_G}} \mathrm{card}(\mu', [\![P]\!]_G).$$

Hence, $\mathrm{card}(\theta, [\![f_{1,2}(Q)]\!]_{g_{1,2}(G)}) = \mathrm{card}(h_{2,1}(\theta), [\![Q]\!]_G)$.

Hence, SPARQL queries of the form $(\mathrm{SELECT}\ \bar{X}\ P)$ are simulable with NRMD$^\neg$.

Hence, the triple $(f_{1,2}, g_{1,2}, h_{2,1})$ is a simulation of SPARQL in NRMD$^\neg$. □

Claim 5 (NRMD$^\neg$ to SPARQL)**.** *The triple $(f_{2,1}, g_{2,1}, h_{1,2})$ is a simulation of NRMD$^\neg$ in SPARQL.*

Proof. To prove this claim, we consider only normalized NRMD$^\neg$ queries Q, that is, queries where rules consist of projection rules, join rules and negation rules (see Section 6.2). This proof follows from induction on the structure of query $Q = (q(\bar{X}), \Pi)$ with inductive hypothesis $\theta \in [\![Q]\!]_D$ if and only if $h_{1,2}(\theta) \in [\![f_{2,1}(Q)]\!]_{g_{2,1}(D)}$ and $\mathrm{card}(\theta, [\![Q]\!]_D) = \mathrm{card}(h_{1,2}(\theta), [\![f_{2,1}(Q)]\!]_{g_{2,1}(D)})$.

1. If q is an extensional predicate, then $f_{2,1}(Q)$ is the SPARQL query

$$(\mathrm{SELECT}\ ?X_1 \ldots ?X_n\ ((?Y, \alpha_0, p)\ \mathrm{AND}\ (?Y, \alpha_1, ?X_1)\ \mathrm{AND} \cdots \mathrm{AND}\ (?Y, \alpha_n, ?X_n)),$$

where the SPARQL variables $?X_1 \ldots ?X_n$ correspond to the n NRMD$^\neg$ variables in $\bar{X}$.

(a) By construction, $\theta \in [\![Q]\!]_D$ if and only if $h_{1,2}(\theta) \in [\![f_{2,1}(Q)]\!]_{g_{2,1}(D)}$. Indeed, each colored fact $\langle q(a_1, \ldots, a_n), i\rangle$ in $\mathrm{coloring}(D)$ corresponds to a subgraph $\{(u_i, \alpha_0, p), (u_i, \alpha_1, a_1), \cdots, (u_i, \alpha_n, a_n)\}$ where u_i is a fresh IRI to identify the colored fact, and $a_i = \theta(x_i)$, for the i-th variable $x_i \in \bar{X}$.

(b) $\mathrm{card}(\theta, [\![Q]\!]_D)$ is the multiplicity of $q(a_1, \ldots, a_n)$ in multiset D. By construction, this is the number of subsgraphs of the form $\{(u_i, \alpha_0, p), (u_i, \alpha_1, a_1), \cdots, (u_i, \alpha_n, a_n)\}$ of $g_{2,1}(D)$. Hence, $\mathrm{card}(\theta, [\![Q]\!]_D) = \mathrm{card}(h_{1,2}(\theta), [\![f_{2,1}(Q)]\!]_{g_{2,1}(D)})$.

2. If q is an intensional predicate, then there are several rules in Π with head $q(\bar{X})$, each one matching one of the following forms:

   – $q(\bar{X}) \leftarrow p(\bar{Y})$,
   – $q(\bar{X}) \leftarrow p_1(\bar{Y}_1), p_2(\bar{Y}_2)$,
   – $q(\bar{X}) \leftarrow p_3(\bar{Y}_3), \neg p_4(\bar{Y}_4)$.

   The function $f_{2,1}(Q)$ maps each of these rules to one of the following SPARQL queries:

   – $(\mathrm{SELECT}\ \bar{X}\ f_{2,1}((p(\bar{Y}), \Pi)))$,
   – $(f_{2,1}((p_1(\bar{X}), \Pi))\ \mathrm{AND}\ f_{2,1}((p_2(\bar{X}), \Pi)))$,
   – $(f_{2,1}((p_3(\bar{X}), \Pi))\ \mathrm{EXCEPT}\ f_{2,1}((p_4(\bar{X}), \Pi)))$.

   If $\{R_1, \ldots, R_n\}$ is the set rules in Π with predicate q in the head, then the SPARQL query $f_{2,1}(Q)$ has the form $(P_1\ \mathrm{UNION} \cdots \mathrm{UNION}\ P_n)$, where P_i is the corresponding SPARQL query for the rule R_i, for $1 \leqslant i \leqslant n$.

   (a) First we will prove that the SPARQL query and the NRMD$^\neg$ query have the same answers. A substitution θ is an answer of query Q if and only if at least one of the following conditions holds:

   – For a rule R_i of the form $q(\bar{X}) \leftarrow p(\bar{Y})$, there exists a solution θ' of query $(p(\bar{Y}), \Pi)$ such that $\theta(x) = \theta'(x)$ for every variable $x \in \bar{X}$. Then, by the inductive hypothesis, there exists a solution $\mu' \in [\![f_{2,1}((p(\bar{Y}), \Pi))]\!]_{g_{2,1}(D)}$ such that $h_{1,2}(\mu') = \theta'$. Let μ be the solution mapping $\mu'|_{\bar{X}}$. By construction, $\mu \in [\![f_{2,1}(Q)]\!]_{g_{2,1}(D)}$ and $h_{1,2}(\mu) = \theta$.
   – For a rule R_i of the form $q(\bar{X}) \leftarrow p_1(\bar{Y}_1), p_2(\bar{Y}_2)$, substitutions $\theta_1 = \theta|_{\bar{Y}_1}$ and $\theta_2 = \theta|_{\bar{Y}_2}$ are solutions of queries $(p_1(\bar{Y}_1), \Pi)$ and $(p_2(\bar{Y}_2), \Pi)$. By the inductive hypothesis, there exist two solutions $\mu_1 \in [\![f_{2,1}((p_1(\bar{Y}_1), \Pi))]\!]_{g_{2,1}(D)}$ and $\mu_2 \in [\![f_{2,1}((p_2(\bar{Y}_2), \Pi))]\!]_{g_{2,1}(D)}$ such that $h_{1,2}(\mu_1) = \theta_1$ and $h_{1,2}(\mu_2) = \theta_2$. Let μ be the solution mapping $\mu_1 \cup \mu_2$. By construction, $\mu \in [\![f_{2,1}(Q)]\!]_{g_{2,1}(D)}$ and $h_{1,2}(\mu) = \theta$.
   – For a rule R_i of the form $q(\bar{X}) \leftarrow p_3(\bar{Y}_3), \neg p_4(\bar{Y}_4)$, substitution θ is a solution of query $(p_3(\bar{Y}_3), \Pi)$ and θ is not a solution of query $(p_4(\bar{Y}_3), \Pi)$. By the inductive hypothesis, there exists a solution $\mu \in [\![f_{2,1}((p_3(\bar{Y}_3), \Pi))]\!]_{g_{2,1}(D)}$ such that $\mu \notin [\![f_{2,1}((p_4(\bar{Y}_4), \Pi))]\!]_{g_{2,1}(D)}$, and $h_{1,2}(\mu) = \theta$. By construction, $\mu \in [\![f_{2,1}(Q)]\!]_{g_{2,1}(D)}$.

   Hence, $\theta \in [\![Q]\!]_D$ if and only if there exists μ such that $f_{1,2}(\mu) = \theta$ and $\mu \in [\![f_{2,1}(Q)]\!]_{g_{2,1}(D)}$.

   (b) We next prove that the answers have the same cardinality in SPARQL and NRMD$^\neg$. By definition,

$$\begin{aligned}\mathrm{card}(\theta, [\![Q]\!]_D) = &\sum_{\theta'|_{\bar{X}}=\theta} \mathrm{card}(\theta', [\![(p(\bar{Y}), \Pi)]\!]_D) + \\ &\mathrm{card}(\theta_1, [\![(p_1(\bar{Y}_1), \Pi)]\!]_D) \times \mathrm{card}(\theta_2, [\![(p_2(\bar{Y}_2), \Pi)]\!]_D) + \\ &\mathrm{card}(\theta, [\![(p_3(\bar{Y}_3), \Pi)]\!]_D).\end{aligned}$$

   By the inductive hypothesis,

$$\begin{aligned}\mathrm{card}(\theta, [\![Q]\!]_D) = &\sum_{\substack{\theta'|_{\bar{X}}=\theta \\ h_{1,2}(\mu')=\theta'}} \mathrm{card}(\mu', [\![f_{2,1}((p(\bar{Y}), \Pi))]\!]_{g_{2,1}(D)}) + \\ &\mathrm{card}(\mu_1, [\![f_{2,1}((p_1(\bar{Y}_1), \Pi))]\!]_{g_{2,1}(D)}) \times \mathrm{card}(\mu_2, [\![f_{2,1}((p_2(\bar{Y}_2), \Pi))]\!]_{g_{2,1}(D)}) + \\ &\mathrm{card}(\mu, [\![f_{2,1}((p_3(\bar{Y}_3), \Pi))]\!]_{g_{2,1}(D)}) \\ = &\ \mathrm{card}(\mu, [\![f_{2,1}(Q)]\!]_{g_{2,1}(D)}).\end{aligned}$$

Hence, the triple $(f_{2,1}, g_{2,1}, h_{1,2})$ is a simulation of NRMD$^\neg$ in SPARQL. □

Claim 6 (MRA to NRMD$^\neg$)**.** *The triple* $(f_{3,2}, g_{3,2}, h_{2,3})$ *is a simulation of MRA in NRMD*$^\neg$.

Proof. We prove this claim for normalized MRA expressions where the condition of a selection formula is always an equality atom (e.g., $\sigma_{A=B}(R)$). This proof follows by induction on the structure of a MRA expression E, assuming that given a MRA database D, for every subquery E' of E it holds that $t' \in [\![E']\!]_D$ if and only if there exists a NRMD$^\neg$ solution $\theta' \in [\![f_{3,2}(E)]\!]_{g_{3,2}(D)}$ such that $h_{2,3}(\theta') = t'$.

1. If E is a relation name R then $f_{3,2}(E)$ is the NRMD$^\neg$ query $(r(\widehat{E}), \emptyset)$ where r is an extensional predicate.
   (a) By definition, $t \in [\![E]\!]_D$ if and only if t belongs to the multiset relation corresponding to the relation name R in the database D. By construction, $t \in [\![E]\!]_D$ is thus equivalent to $\theta \in [\![f_{3,2}(E)]\!]_{g_{3,2}(D)}$, where $h_{2,3}(\theta) = t$. Indeed, $t \in R^I$ if and only if $r(a_1, \ldots, a_n) \in g_{3,2}(D)$ and $t = (a_1, \ldots, a_n)$.
   (b) The fact that $\mathrm{card}(t, [\![E]\!]_D) = \mathrm{card}(\theta, [\![f_{3,2}(E)]\!]_{g_{3,2}(D)})$ follows by construction; the multiplicity of t in the multiset relation corresponding to the relation name R is the same as the multiplicity of fact $r(a_1, \ldots, a_n)$ in multiset $g_{3,2}(D)$.
2. If E is a query $E_1 \cup E_2$, then $\widehat{E}_1 = \widehat{E}$ and $\widehat{E}_2 = \widehat{E}$, and $f_{3,2}(E)$ is a NRMD$^\neg$ query $(q(\widehat{E}), \Pi)$ such that $f_{3,2}(E_1) = (q_1(\widehat{E}), \Pi)$ and $f_{3,2}(E_2) = (q_2(\widehat{E}), \Pi)$, and program Π includes the rules $q(\widehat{E}) \leftarrow q_1(\widehat{E})$ and $q(\widehat{E}) \leftarrow q_2(\widehat{E})$.
   (a) By definition, $t \in [\![E]\!]_D$ if and only if $t \in [\![E_1]\!]_D$ or $t \in [\![E_2]\!]_D$. By the induction hypothesis, $t \in [\![E_2]\!]_D$ is equivalent to say that there exists θ such that $h_{2,3}(\theta) = t$ and $\theta \in [\![f_{3,2}(E_1)]\!]_{g_{3,2}(D)}$ or $\theta \in [\![f_{3,2}(E_2)]\!]_{g_{3,2}(D)}$. That is, $\theta \in [\![f_{3,2}(E)]\!]_{g_{3,2}(D)}$.
   (b) Assume the respective answers t and θ described in (a). By definition,
   $$\mathrm{card}(t, [\![E]\!]_D) = \mathrm{card}(t, [\![E_1]\!]_D) + \mathrm{card}(t, [\![E_2]\!]_D),$$
   $$\mathrm{card}(\theta, [\![f_{3,2}(E)]\!]_{g_{3,2}(D)}) = \mathrm{card}(\theta, [\![f_{3,2}(E_1)]\!]_{g_{3,2}(D)}) + \mathrm{card}(\theta, [\![f_{3,2}(E_2)]\!]_{g_{3,2}(D)}).$$
   By the inductive hypothesis, these two multiplicities are equal.
3. If E is a query $E_1 \bowtie E_2$ then $\widehat{E}_1 \cup \widehat{E}_2 = \widehat{E}$, $f_{3,2}(E) = (q(\widehat{E}), \Pi)$, $f_{3,2}(E_1) = (q_1(\widehat{E}_1), \Pi)$, $f_{3,2}(E_2) = (q_2(\widehat{E}_2), \Pi)$, and program Π includes the rule $q(\widehat{E}) \leftarrow q_1(\widehat{E}_1), q_2(\widehat{E}_2)$.
   (a) By definition, $t \in [\![E]\!]_D$ if and only if there exists two tuples t_1 and t_2 such that $t_1 \sim t_2$, $t = t_1 \cup t_2$, $t_1 \in [\![E_1]\!]_D$ and $t_2 \in [\![E_2]\!]_D$. By the induction hypothesis, $t \in [\![E]\!]_D$ if and only if there exists two NRMD$^\neg$ solutions $\theta_1 \in [\![f_{3,2}(E_1)]\!]_{g_{3,2}(D)}$ and $\theta_2 \in [\![f_{3,2}(E_2)]\!]_{g_{3,2}(D)}$ where $h_{2,3}(\theta_1) = t_1$ and $h_{2,3}(\theta_2) = t_2$. Let θ be $\theta_1 \cup \theta_2$. By construction, $\theta \in [\![f_{3,2}(E)]\!]_{g_{3,2}(D)}$ and $h_{2,3}(\theta) = t$.
   (b) Assume the respective answers t, t_1, t_2, θ, θ_1, and θ_2 described in (a). By definition,
   $$\mathrm{card}(t, [\![E]\!]_D) = \mathrm{card}(t_1, [\![E_1]\!]_D) \times \mathrm{card}(t_2, [\![E_1]\!]_D),$$
   $$\mathrm{card}(\theta, [\![f_{3,2}(E)]\!]_{g_{3,2}(D)}) = \mathrm{card}(\theta_1, [\![f_{3,2}(E_1)]\!]_{g_{3,2}(D)}) \times \mathrm{card}(\theta_2, [\![f_{3,2}(E_1)]\!]_{g_{3,2}(D)}).$$
   By the inductive hypothesis, these two multiplicities are equal.
4. If E is a query $E_1 \setminus E_2$ then $\widehat{E}_1 = \widehat{E}$, $\widehat{E}_1 = \widehat{E}$, $f_{3,2}(E) = (q(\widehat{E}), \Pi)$, $f_{3,2}(E_1) = (q_1(\widehat{E}), \Pi)$, $f_{3,2}(E_2) = (q_2(\widehat{E}), \Pi)$, and program Π includes the rule $q(\widehat{E}) \leftarrow q_1(\widehat{E}), \neg q_2(\widehat{E})$.
   (a) By definition, $t \in [\![E]\!]_D$ if and only if $t \in [\![E_1]\!]_D$ and $t \notin [\![E_2]\!]_D$. By the induction hypothesis, $t \in [\![E]\!]_D$ if and only if there exists a NRMD$^\neg$ solution θ such that $\theta \in [\![f_{3,2}(E_1)]\!]_{g_{3,2}(D)}$, $\theta \notin [\![f_{3,2}(E_2)]\!]_{g_{3,2}(D)}$, and $h_{2,3}(\theta) = t$. By construction, $\theta \in [\![f_{3,2}(E)]\!]_{g_{3,2}(D)}$.
   (b) Assume the respective answers t and θ described in (a). By definition, $\mathrm{card}(t, [\![E]\!]_D) = \mathrm{card}(t, [\![E_1]\!]_D)$ and $\mathrm{card}(\theta, [\![f_{3,2}(E)]\!]_{g_{3,2}(D)}) = \mathrm{card}(\theta, [\![f_{3,2}(E_1)]\!]_{g_{3,2}(D)})$. By the induction hypothesis, these two multiplicities are equal.

5. If E is a query $\pi_S(E_1)$ then $\widehat{E} = S$ and $S \subseteq \widehat{E_1}$, and $f_{3,2}(E)$ is a NRMD$^\neg$ query $(q(\bar{X}), \Pi)$ such that $f_{3,2}(E_1) = (q_1(\widehat{E}), \Pi)$, and program Π includes the rule $q(\widehat{E}) \leftarrow q_1(\widehat{E_1})$.

   (a) By definition, $t \in [\![E]\!]_D$ if and only if there exists a tuple $t_1 \in [\![E_1]\!]_D$ such that $t_1|_{\widehat{E}} = t$. By the induction hypothesis, $t_1 \in [\![E]\!]_D$ if and only if there exists $\theta_1 \in [\![f_{3,2}(E_1)]\!]_{g_{3,2}(D)}$ such that $h_{2,3}(\theta_1) = t_1$. Let θ be $\theta_1|_{\widehat{E}}$. By construction, $t \in [\![E]\!]_D$ if and only if $\theta \in [\![f_{3,2}(E)]\!]_{g_{3,2}(D)}$ and $h_{2,3}(\theta) = t$.

   (b) Assume the respective answers t and θ described in (a). By definition,

$$\mathrm{card}(t, [\![E]\!]_D) = \sum_{\substack{t_1 \in [\![E_1]\!]_D \\ t_1|_{\widehat{E}} = t}} \mathrm{card}(t_1, [\![E_1]\!]_D),$$

$$\mathrm{card}(\theta, [\![f_{3,2}(E)]\!]_{g_{3,2}(D)}) = \sum_{\substack{\theta_1 \in [\![f_{3,2}(E_1)]\!]_{g_{3,2}(D)} \\ \theta_1|_{\widehat{E}} = \theta}} \mathrm{card}(\theta_1, [\![f_{3,2}(E_1)]\!]_{g_{3,2}(D)}).$$

   By the induction hypothesis, these two multiplicities are equal.

6. If E is a query $\rho_{A/B}(E_1)$ then $\widehat{E} = (\widehat{E_1} \setminus \{A\}) \cup \{B\}$, $f_{3,2}(E) = (q(\widehat{E}), \Pi)$, and $f_{3,2}(E_1) = (q(\widehat{E_1}), \Pi)$.

   (a) By definition, $t \in [\![E]\!]_D$ if and only if there exists a tuple $t_1 \in [\![E_1]\!]_D$ where $t(C) = t_1(C)$ for every attribute $C \in \widehat{E} \setminus \{A\}$, and $t(A) = t_1(B)$. By the induction hypothesis, $t_1 \in [\![E_1]\!]_D$ if and only if there exists a solution $\theta_1 \in [\![f_{3,2}(E_1)]\!]_{g_{3,2}(D)}$ such that $h_{2,3}(\theta_1) = t_1$. Let θ be the tuple with domain $\widehat{E}$ such that $\theta(C) = \theta_1(C)$ for every attribute $C \in \widehat{E} \setminus \{A\}$, and $\theta(A) = \theta_1(B)$. By construction, $\theta \in [\![f_{3,2}(E)]\!]_{g_{3,2}(D)}$ if and only if $\theta_1 \in [\![f_{3,2}(E_1)]\!]_{g_{3,2}(D)}$ and $h_{2,3}(\theta) = t$.

   (b) Assume the respective query answers t, t_1, θ, and θ_1 described in (a). By definition,

$$\mathrm{card}(t, [\![E]\!]_D) = \mathrm{card}(t_1, [\![E_1]\!]_D),$$

$$\mathrm{card}(\theta, [\![f_{3,2}(E)]\!]_{g_{3,2}(D)}) = \mathrm{card}(\theta_1, [\![f_{3,2}(E_1)]\!]_{g_{3,2}(D)}).$$

   By the induction hypothesis, these two multiplicities are equal.

7. If E is a query $\sigma_{A=B}(E_1)$ then $\widehat{E} = \widehat{E_1}$, $f_{3,2}(E) = (q(\widehat{E}), \Pi)$, and $f_{3,2}(E_1) = (q_1(\widehat{E_1}), \Pi)$ and program Π includes the rule $q(\widehat{E}) \leftarrow q_1(\widehat{E_1}), A = B$.

   (a) By definition, $t \in [\![E]\!]_D$ if and only if $t \in [\![E_1]\!]_D$ and $t(A) = t(B)$. By the induction hypothesis, there is an answer $\theta \in [\![f_{3,2}(E_1)]\!]_{g_{3,2}(D)}$ such that $h_{2,3}(\theta) = t$. By construction, $\theta(A) = \theta(B)$. Then, $\theta \in [\![f_{3,2}(E)]\!]_{g_{3,2}(D)}$.

   (b) Assume the respective query answers t and θ described in (a). By definition,

$$\mathrm{card}(t, [\![E]\!]_D) = \mathrm{card}(t_1, [\![E_1]\!]_D),$$

$$\mathrm{card}(\theta, [\![f_{3,2}(E)]\!]_{g_{3,2}(D)}) = \mathrm{card}(\theta_1, [\![f_{3,2}(E_1)]\!]_{g_{3,2}(D)}).$$

   By the induction hypothesis, these two multiplicities are equal.

Hence, the triple $(f_{3,2}, g_{3,2}, h_{2,3})$ is a simulation of MRA in NRMD$^\neg$. □

Claim 7 (NRMD$^\neg$ to MRA)**.** *The triple $(f_{2,3}, g_{2,3}, h_{3,2})$ is a simulation of NRMD$^\neg$ in SPARQL.*

Proof. To prove this claim we consider only normalized NRMD$^\neg$ queries Q, that is, queries where rules consist of projection rules, join rules and negation rules (see Section 6.2). This prove follows from induction on the structure of query $Q = (q(\bar{X}), \Pi)$ with inductive hypothesis $\theta \in [\![Q]\!]_D$ if and only if $h_{3,2}(\theta) \in [\![f_{2,3}(Q)]\!]_{g_{2,3}(D)}$ and $\mathrm{card}(\theta, [\![Q]\!]_D) = \mathrm{card}(h_{3,2}(\theta), [\![f_{2,3}(Q)]\!]_{g_{2,3}(D)})$.

1. If q is a extensional predicate, then $f_{2,1}(Q)$ is the MRA query $\rho_{A_1/X_1}(\cdots\rho_{A_n/X_n}(R))$, where the MRA attributes $X_1,\ldots,X_n$ correspond to the n NRMD$^\neg$ variables in $\bar{X}$, and R is the relation name corresponding to predicate q.

   (a) Let r be the MRA relation associated to relation name R in the MRA database $g_{2,3}(D)$. Let θ be a NRMD$^\neg$ answer with domain $\{X_1,\ldots,X_1\}$, and t be the MRA tuple where $t(A_i) = \theta(X_i)$ for $1 \leqslant i \leqslant n$. By definition, $\theta \in [\![Q]\!]_D$ if and only if $p(\theta(X_1),\ldots,\theta(X_n)) \in D$. Because, by definition, each fact $q(a_1,\ldots,a_n)$ in D corresponds to a tuple $t \in r$ where $t(A_i) = a_i$ for $1 \leqslant i \leqslant n$, then $\theta \in [\![Q]\!]_D$ if and only if $t \in r$. Let s be a MRA tuple with $\hat{s} = \{X_1,\ldots,X_n\}$ where $s(X_i) = t(A_i)$, for $1 \leqslant i \leqslant n$. By definition, $t \in r$ if and only if $s \in [\![\rho_{A_1/X_1}(\cdots\rho_{A_n/X_n}(R))]\!]_{g_{2,3}(D)}$. By construction, $s = h_{3,2}(\theta)$. Hence, $\theta \in [\![Q]\!]_D$ if and only if $h_{3,2}(\theta) \in [\![f_{2,3}(Q)]\!]_{g_{2,3}(D)}$.

   (b) The identity $\mathrm{card}(\theta, [\![Q]\!]_D) = \mathrm{card}(h_{3,2}(\theta), [\![f_{2,3}Q]\!]_{f_{2,3}(D)})$ follows from the next identities:

$$\begin{aligned}\mathrm{card}(\theta, [\![Q]\!]_D) &= \mathrm{card}(q(\theta(X_1),\ldots,\theta(X_n)), D)\\ &= \mathrm{card}(t, r)\\ &= \mathrm{card}(s, [\![f_{2,3}(Q)]\!]_{g_{2,3}(D)})\\ &= \mathrm{card}(h_{3,2}(\theta), [\![f_{2,3}Q]\!]_{g_{2,3}(D)}).\end{aligned}$$

2. If q is an intensional predicate then there are several rules in Π with head $q(\bar{X})$, each one has matches of the following forms:

   – $q(\bar{X}) \leftarrow p(\bar{Y})$,
   – $q(\bar{X}) \leftarrow p_1(\bar{Y}_1), p_2(\bar{Y}_2)$,
   – $q(\bar{X}) \leftarrow p_3(\bar{Y}_3), \neg p_4(\bar{Y}_4)$.

   where $\bar{X} \subseteq \bar{Y}$, $\bar{Y}_1 \cup \bar{Y}_2 = \bar{X}$, $\bar{Y}_3 = \bar{X}$, and $\bar{Y}_4 = \bar{X}$. The function $f_{2,3}(Q)$ maps each of these rules to one of the following MRA queries:

   – $\pi_{\bar{X}}(f_{2,3}((p(\bar{Y}), \Pi)))$,
   – $(f_{2,3}((p_1(\bar{X}), \Pi)) \bowtie f_{2,3}((p_2(\bar{X}), \Pi)))$,
   – $(f_{2,3}((p_3(\bar{X}), \Pi)) \setminus f_{2,3}((p_4(\bar{X}), \Pi)))$.

   If $\{R_1,\ldots,R_n\}$ is the set rules in Π with predicate q in the head, then the MRA query $f_{2,3}(Q)$ has the form $(E_1 \cup \cdots \cup E_n)$, where E_i is the corresponding MRA expression for the rule R_i, for $1 \leqslant i \leqslant n$.

   (a) First, we will prove that the MRA expression and the NRMD$^\neg$ query have the same answers. A substitution θ is an answer of query Q if and only if one of the following conditions holds:

   – There exists a solution θ' of query $(p(\bar{Y}), \Pi)$ such that $\theta(x) = \theta'(x)$ for every variable $x \in \bar{X}$. By the induction hypothesis, there exists a solution $t' \in [\![f_{2,3}((p(\bar{Y}), \Pi))]\!]_{g_{2,3}(D)}$ such that $h_{3,2}(t') = \theta'$. Let t be the solution mapping $t'|_{\bar{X}}$. By construction, $t \in [\![f_{2,3}(Q)]\!]_{g_{2,3}(D)}$ and $h_{3,2}(t) = \theta$.
   – Substitutions $\theta_1 = \theta|_{\bar{Y}_1}$ and $\theta_2 = \theta|_{\bar{Y}_2}$ are solutions of queries $(p_1(\bar{Y}_1), \Pi)$ and $(p_2(\bar{Y}_2), \Pi)$. By the induction hypothesis, there exists two solutions $t_1 \in [\![f_{2,3}((p_1(\bar{Y}_1), \Pi))]\!]_{g_{2,3}(D)}$ and $t_2 \in [\![f_{2,3}((p_2(\bar{Y}_2), \Pi))]\!]_{g_{2,3}(D)}$ such that $h_{3,2}(t_1) = \theta_1$ and $h_{3,2}(t_2) = \theta_2$. Let t be the MRA solution $t_1 \cup t_2$. By construction, $t \in [\![f_{2,3}(Q)]\!]_{g_{2,3}(D)}$ and $h_{3,2}(t) = \theta$.
   – θ is a solution of query $(p_3(\bar{Y}_3), \Pi)$ and θ is not a solution of query $(p_4(\bar{Y}_3), \Pi)$. By the induction hypothesis, there exists a solution $t \in [\![f_{2,3}((p_3(\bar{Y}_3), \Pi))]\!]_{g_{2,3}(D)}$ such that $t \notin [\![f_{2,3}((p_4(\bar{Y}_4), \Pi))]\!]_{g_{2,3}(D)}$, and $h_{3,2}(t) = \theta$. By construction, $t \in [\![f_{2,3}(Q)]\!]_{g_{2,3}(D)}$.

   Hence, $\theta \in [\![Q]\!]_D$ if and only if there exists μ such that $f_{1,2}(\mu) = \theta$ and $\mu \in [\![f_{2,1}(Q)]\!]_{g_{2,1}(D)}$.

(b) We next prove that the answers have the same cardinality in MRA and NRMD$^{\neg}$. By definition,

$$\begin{aligned}\mathrm{card}(\theta, [\![Q]\!]_D) = &\sum_{\theta'|_{\bar{X}}=\theta} \mathrm{card}(\theta', [\![(p(\bar{Y}), \Pi)]\!]_D) + \\ &\mathrm{card}(\theta_1, [\![(p_1(\bar{Y}_1), \Pi)]\!]_D) \times \mathrm{card}(\theta_2, [\![(p_2(\bar{Y}_2), \Pi)]\!]_D) + \\ &\mathrm{card}(\theta, [\![(p_3(\bar{Y}_3), \Pi)]\!]_D).\end{aligned}$$

By the induction hypothesis,

$$\begin{aligned}\mathrm{card}(\theta, [\![Q]\!]_D) = &\sum_{\substack{\theta'|_{\bar{X}}=\theta \\ h_{1,2}(t')=\theta'}} \mathrm{card}(t', [\![f_{2,3}((p(\bar{Y}), \Pi))]\!]_{g_{2,3}(D)}) + \\ &\mathrm{card}(t_1, [\![f_{2,3}((p_1(\bar{Y}_1), \Pi))]\!]_{g_{2,3}(D)}) \times \mathrm{card}(t_2, [\![f_{2,3}((p_2(\bar{Y}_2), \Pi))]\!]_{g_{2,3}(D)}) + \\ &\mathrm{card}(t, [\![f_{2,3}((p_3(\bar{Y}_3), \Pi))]\!]_{g_{2,3}(D)}) \\ = &\,\mathrm{card}(t, [\![f_{2,3}(Q)]\!]_{g_{2,3}(D)}).\end{aligned}$$

Hence, the triple $(f_{2,3}, g_{2,3}, h_{3,2})$ is a simulation of NRMD$^{\neg}$ in MRA. □

Claim 8 (MRA to SPARQL)**.** *The triple $(f_{3,1}, g_{3,1}, h_{1,3})$ is a simulation of NRMD$^{\neg}$ in SPARQL.*

Proof. We proof this claim for normalized MRA expressions where the condition of a selection formula is always an equality atom (e.g., $\sigma_{A=B}(R)$). We proof this claim by induction on the structure of a MRA expression E, assuming that given an MRA database D, for every subquery E' of E it holds that $t' \in [\![E']\!]_D$ if and only if there exists a SPARQL solution $\mu' \in [\![f_{3,1}(E)]\!]_{g_{3,1}(D)}$ such that $h_{1,3}(\mu') = t'$.

1. If E is a relation name R then $f_{3,1}(E)$ is the SPARQL query $(\mathrm{SELECT}\ \ ?A_1 \cdots ?A_n\ \ P)$ where P is the basic graph pattern $((?X, u_b, u_r)\,\mathrm{AND}(?X, u_1, ?A_1)\,\mathrm{AND} \cdots \mathrm{AND}(?X, u_n, ?A_n)))$, and $?A_1, \ldots, ?A_n$ are the variables corresponding to the attributes associated to relation name R.
   (a) Let t be an MRA tuple with $\hat{t} = \widehat{R}$, and μ be an SPARQL mapping with $h_{1,3}(\mu) = t$. By definition, $t \in [\![E]\!]_D$ if and only if tuple t belongs to multiset relation R^D. By construction, $t \in [\![E]\!]_D$ is thus equivalent to the existence of an an IRI u such that the triples (u, u_b, u_r), $(u, u_1, t(A_1)), \ldots, (u, u_n, t(A_n))$ belong to the RDF graph $g_{3,1}(D)$. By definition, there exists such an IRI u if and only if there exists a SPARQL mapping $\mu' \in [\![P]\!]_{g_{3,1}(D)}$ where $\mu'(?X) = u$ and $\mu(?A_i) = t(a_i)$, for $1 \leqslant i \leqslant n$. By construction, $\mu'|_{?A_1,\ldots,?A_n} = \mu$. Then, $\mu' \in [\![P]\!]_{g_{3,1}(D)}$ if and only if $\mu \in [\![f_{3,1}(Q)]\!]_{g_{3,1}(D)}$.
   (b) The fact that $\mathrm{card}(t, [\![E]\!]_D) = \mathrm{card}(\mu, [\![f_{3,1}(E)]\!]_{g_{3,1}(D)})$ follows by construction; the multiplicity of t in the multiset relation R^D is the same as the number of IRIs u such that such that the triples (u, u_b, u_r), $(u, u_1, t(A_1)), \ldots, (u, u_n, t(A_n))$ belong to the RDF graph $g_{3,1}(D)$.
2. If E is a query $E_1 \cup E_2$, then $\widehat{E}_1 = \widehat{E}$, $\widehat{E}_2 = \widehat{E}$, and $f_{3,1}(E)$ is the SPARQL query $(f_{3,1}(E_1)\,\mathrm{UNION}\,f_{3,1}(E_2))$.
   (a) Let t be an MRA tuple with $\hat{t} = \widehat{E}$, and μ be an SPARQL mapping such that $h_{1,3}(\mu) = t$. By definition, $t \in [\![E]\!]_D$ if and only if $t \in [\![E_1]\!]_D$ or $t \in [\![E_2]\!]_D$. By the induction hypothesis, $t \in [\![E]\!]_D$ if and only if $\mu \in [\![f_{3,1}(E_1)]\!]_{g_{3,1}(D)}$ or $\mu \in [\![f_{3,1}(E_2)]\!]_{g_{3,1}(D)}$. By definition, $t \in [\![E]\!]_D$ if and only if $\mu \in [\![f_{3,1}(E)]\!]_{g_{3,1}(D)}$.
   (b) Assume the respective answers t and μ described in (a). By definition,

$$\begin{aligned}&\mathrm{card}(t, [\![E]\!]_D) = \mathrm{card}(t, [\![E_1]\!]_D) + \mathrm{card}(t, [\![E_2]\!]_D), \\ &\mathrm{card}(\mu, [\![f_{3,1}(E)]\!]_{g_{3,1}(D)}) = \mathrm{card}(\mu, [\![f_{3,1}(E_1)]\!]_{g_{3,1}(D)}) + \mathrm{card}(\mu, [\![f_{3,1}(E_2)]\!]_{g_{3,1}(D)}).\end{aligned}$$

   By the induction hypothesis, these two multiplicities are equal.

3. If E is a query $E_1 \ltimes E_2$ then $\widehat{E}_1 \cup \widehat{E}_2 = \widehat{E}$, and $f_{3,1}(E)$ is the SPARQL query $(f_{3,1}(E_1)\,\mathrm{AND}\,f_{3,1}(E_2))$.
   (a) Let t be an MRA tuple with $\hat{t} = \widehat{E}$, and μ be an SPARQL mapping such that $h_{1,3}(\mu) = t$. By definition, $t \in [\![E]\!]_D$ if and only if there exists two tuples t_1 and t_2 such that $t_1 \sim t_2$, $t = t_1 \cup t_2$, $t_1 \in [\![E_1]\!]_D$ and $t_2 \in [\![E_2]\!]_D$. By the induction hypothesis, $t_1 \in [\![E_1]\!]_D$ and $t_2 \in [\![E_2]\!]_D$ if and only there exist two SPARQL mappings μ_1 and μ_2 such that $h_{1,3}(\mu_1) = t_1$, $h_{1,3}(\mu_2) = t_2$, $\mu_1 \in [\![f_{3,1}(E_1)]\!]_{g_{3,1}(D)}$, and $\mu_1 \in [\![f_{3,1}(E_1)]\!]_{g_{3,1}(D)}$. By construction, $\mu_1 \sim \mu_2$, $\mu_1 \cup \mu_2 = \mu$, and $\mu \in [\![f_{3,1}(E)]\!]_{g_{3,1}(D)}$. Hence, $t \in [\![E]\!]_D$ if and only if $\mu \in [\![f_{3,1}(E)]\!]_{g_{3,1}(D)}$.
   (b) Assume the respective answers t, t_1, t_2, μ, μ_1, and μ_2 described in (a). By definition,

$$\mathrm{card}(t, [\![E]\!]_D) = \mathrm{card}(t_1, [\![E_1]\!]_D) \times \mathrm{card}(t_2, [\![E_1]\!]_D),$$
$$\mathrm{card}(\mu, [\![f_{3,1}(E)]\!]_{g_{3,1}(D)}) = \mathrm{card}(\mu_1, [\![f_{3,1}(E_1)]\!]_{g_{3,1}(D)}) \times \mathrm{card}(\mu_2, [\![f_{3,1}(E_1)]\!]_{g_{3,1}(D)}).$$

   By the induction hypothesis, these two multiplicities are equal.

4. If E is a query $E_1 \setminus E_2$ then $\widehat{E}_1 = \widehat{E}$, $\widehat{E}_1 = \widehat{E}$, and $f_{3,1}(E)$ is the SPARQL query $(f_{3,1}(E_1)\,\mathrm{EXCEPT}\,f_{3,1}(E_2))$.
   (a) Let t be an MRA tuple with $\hat{t} = \widehat{E}$, and μ be an SPARQL mapping such that $h_{1,3}(\mu) = t$. By definition, $t \in [\![E]\!]_D$ if and only if $t \in [\![E_1]\!]_D$ and $t \notin [\![E_2]\!]_D$. By the induction hypothesis, $t \in [\![E]\!]_D$ if and only if $\mu \in [\![f_{3,1}(E_1)]\!]_{g_{3,1}(D)}$ and $\mu \notin [\![f_{3,1}(E_2)]\!]_{g_{3,1}(D)}$. Hence, $t \in [\![E]\!]_D$ if and only if $\mu \in [\![f_{3,1}(E)]\!]_{g_{3,1}(D)}$.
   (b) Assume the respective answers t and μ described in (a). By definition, $\mathrm{card}(t, [\![E]\!]_D) = \mathrm{card}(t, [\![E_1]\!]_D)$ and $\mathrm{card}(\mu, [\![f_{3,1}(E)]\!]_{g_{3,1}(D)}) = \mathrm{card}(\mu, [\![f_{3,1}(E_1)]\!]_{g_{3,1}(D)})$. By the induction hypothesis, these two multiplicities are equal.

5. If E is a query $\pi_S(E_1)$ then $\widehat{E} = S$ and $S \subseteq \widehat{E_1}$, and $f_{3,1}(E)$ is a SPARQL query $(\mathrm{SELECT}\ W\ f_{3,1}(E_1))$ such that W is the corresponding set of SPARQL variables for the set of attributes S.
   (a) Let t be an MRA tuple with $\hat{t} = \widehat{E}$. By definition, $t \in [\![E]\!]_D$ if and only if there exists a tuple $t_1 \in [\![E_1]\!]_D$ such that $t_1|_{\widehat{E}} = t$. By the induction hypothesis, $t_1 \in [\![E]\!]_D$ if and only if there exists $\mu_1 \in [\![f_{3,1}(E_1)]\!]_{g_{3,1}(D)}$ such that $h_{1,3}(\mu_1) = t_1$. Let μ be $\mu_1|_W$. By construction, $t \in [\![E]\!]_D$ if and only if $\mu \in [\![f_{3,1}(E)]\!]_{g_{3,1}(D)}$ and $h_{1,3}(\mu) = t$.
   (b) Assume the respective answers t and μ described in (a). By definition,

$$\mathrm{card}(t, [\![E]\!]_D) = \sum_{\substack{t_1 \in [\![E_1]\!]_D \\ t_1|_{\widehat{E}} = t}} \mathrm{card}(t_1, [\![E_1]\!]_D),$$
$$\mathrm{card}(\mu, [\![f_{3,1}(E)]\!]_{g_{3,1}(D)}) = \sum_{\substack{\mu_1 \in [\![f_{3,1}(E_1)]\!]_{g_{3,1}(D)} \\ \mu_1|_W = \mu}} \mathrm{card}(\mu_1, [\![f_{3,1}(E_1)]\!]_{g_{3,1}(D)}).$$

   By the induction hypothesis, these two multiplicities are equal.

6. If E is a query $\rho_{A/B}(E_1)$ then $\widehat{E} = (\widehat{E}_1 \setminus \{A\}) \cup \{B\}$, $f_{3,1}(E)$ is the SPARQL query that results from consistently renaming variable $?A$ as variable $?B$ in $f_{3,1}(E_1)$ (i.e., $\mathrm{subs}_{?A/?B}(A)$), and $?A$ and $?B$ are the corresponding SPARQL variables for atributes A and B.
   (a) By definition, $t \in [\![E]\!]_D$ if and only if there exists a tuple $t_1 \in [\![E_1]\!]_D$ where $t(C) = t_1(C)$ for every attribute $C \in \widehat{E} \setminus \{A\}$, and $t(A) = t_1(B)$. By the induction hypothesis, $t_1 \in [\![E_1]\!]_D$ if and only if there exists a solution $\mu_1 \in [\![f_{3,1}(E_1)]\!]_{g_{3,1}(D)}$ such that $h_{1,3}(\mu_1) = t_1$. Let μ be the SPARQL mapping with domain $(\mathrm{dom}(\mu') \setminus \{?A\}) \cup \{?B\}$ such that $\mu(?C) = \mu_1(?C)$ for every variable $?C \in \mathrm{dom}(\mu') \setminus \{?A\}$, and $\mu(?A) = \mu_1(?B)$. By construction, $h_{1,3}(\mu) = t$. Hence, $t \in [\![E]\!]_D$ if and only if $\mu \in [\![f_{3,1}(E)]\!]_{g_{3,1}(D)}$.
   (b) Assume the respective query answers t, t_1, μ, and μ_1 described in (a). By definition,

$$\mathrm{card}(t, [\![E]\!]_D) = \mathrm{card}(t_1, [\![E_1]\!]_D),$$

$$\mathrm{card}(\mu, [\![f_{3,1}(E)]\!]_{g_{3,1}(D)}) = \mathrm{card}(\mu_1, [\![f_{3,1}(E_1)]\!]_{g_{3,1}(D)}).$$

By the induction hypothesis, these two multiplicities are equal.

7. If E is a query $\sigma_{A=B}(E_1)$ then $\widehat{E} = \widehat{E}_1$, $f_{3,1}(E) = (q(\widehat{E}), \Pi)$, and $f_{3,1}(E_1)$ is the SPARQ query $(P_1 \text{ FILTER } ?A = ?B)$ where $?A$ and $?B$ are the corresponding SPARQL variables for the MRA attributes A and B.
   (a) By definition, $t \in [\![E]\!]_D$ if and only if $t \in [\![E_1]\!]_D$ and $t(A) = t(B)$. By the induction hypothesis, there is an answer $\mu \in [\![f_{3,1}(E_1)]\!]_{g_{3,1}(D)}$ such that $h_{1,3}(\mu) = t$. By construction, $\mu(A) = \mu(B)$. Then, $t \in [\![E]\!]_D$ if and only if $\mu \in [\![f_{3,1}(E)]\!]_{g_{3,1}(D)}$.
   (b) Assume the respective query answers t and θ described in (a). By definition,

$$\mathrm{card}(t, [\![E]\!]_D) = \mathrm{card}(t_1, [\![E_1]\!]_D),$$
$$\mathrm{card}(\mu, [\![f_{3,1}(E)]\!]_{g_{3,1}(D)}) = \mathrm{card}(\mu_1, [\![f_{3,1}(E_1)]\!]_{g_{3,1}(D)}).$$

By the induction hypothesis, these two multiplicities are equal.

Hence, the triple $(f_{3,1}, g_{3,1}, h_{1,3})$ is a simulation of MRA in NRMD$^\neg$. □

Claim 9 (SPARQL to MRA)**.** *The triple $(f_{1,3}, g_{1,3}, h_{3,1})$ is a simulation of NRMD$^\neg$ in SPARQL.*

Proof. To prove this claim we show that, for every SPARQL query Q and RDF graph G, it holds that $[\![Q]\!]_G = h_{3,1}([\![f_{1,3}(Q)]\!]_{g_{1,3}})$. For simplicity, we write D instead of D. We next show this identity by induction on the structure of a normalized SPARQL query Q. In this proof we assume that t is a MRA tuple with the attributes of the MRA expression $f_{1,3}(Q)$, and μ is the SPARQL mapping $h_{3,1}(t)$. To show that $[\![Q]\!]_G = h_{3,1}([\![f_{1,3}(Q)]\!]_{g_{1,3}})$, we prove that $\mu \in [\![Q]\!]_G$ if and only if $t \in [\![f_{1,3}(Q)]\!]_D$ and $\mathrm{card}(\mu, [\![Q]\!]_G) = \mathrm{card}(t, [\![f_{1,3}(Q)]\!]_D)$.

1. Case Q is a triple pattern.
   (a) By definition, every triple pattern is translated to a MRA expression consisting of operations σ, ρ, and π over the relation name Trip. For example, if Q is the triple pattern $(?X, p, ?X)$, then $f_{1,2}(Q)$ is the expression $\Pi_X(\rho_{S/X}(\sigma_{P=p \wedge S=O}(\mathrm{Trip})))$. It can be shown that the triple pattern $Q = (?X, p, ?X)$ is equivalent to the SPARQL query $Q' = (\text{SELECT } ?X\ ((?X, ?P, ?O)\ \text{FILTER}(?P = p \wedge ?X = ?O))$. Then, $\mu \in [\![Q]\!]_G$ if and only if there exists a solution $\mu' \in [\![(?X, ?P, ?O)]\!]_G$ such that $\mu = \mu'|_{\{X\}}$, $\mu'(?P) = p$ and $\mu'(?X) = \mu'(?O)$. Without loss of generality, let $\mu'(?X) = a$. Such mapping μ' is a solution of the triple pattern $(?X, ?P, ?O)$ if and only if $(a, p, a) \in G$. By construction, $(a, p, a) \in G$ if and only if $(a, p, a) \in \mathrm{Trip}^D$, where D is the MRA database D. If $(a, p, a) \in \mathrm{Trip}^D$ then $t \in [\![f_{1,3}(Q)]\!]_D$. Hence, $\mu \in [\![Q]\!]_G$ if and only if $t \in [\![f_{1,3}(Q)]\!]_D$. So far, we showed that the claim follows for a particular triple pattern. This result can be extended for all the triple patterns following the same procedure.
   (b) By construction, $\mathrm{card}(\mu, [\![f_{1,1}(Q)]\!]_{g_{1,1}(G)}) = 1$ and $\mathrm{card}(\mu, [\![Q]\!]_G) = 1$. Hence, $\mathrm{card}(t, [\![f_{1,1}(Q)]\!]_{g_{1,1}(G)}) = \mathrm{card}(\mu, [\![Q]\!]_G)$.
2. Case Q is a query $(P_1 \text{ AND } P_2)$. Without loss of generality assume that $\mathrm{inScope}(P_1) = \{?X, ?Y\}$ and $\mathrm{inScope}(P_2) = \{?X, ?Z\}$. By definition, the MRA expression for query Q is:

$$f_{1,3}(Q) = f_{1,3}(P_1) * f_{1,3}(P_2)$$
$$= \pi_{X,Y,Z}(\rho_{A_1/X_1}(\rho_{A_2/X_2}(\rho_{A/X}(\mathrm{Comp}))) \bowtie \rho_{X/X_1}(f_{1,3}(P_1)) \bowtie \rho_{X/X_2}(f_{1,3}(P_2))).$$

   (a) If $t \in [\![f_{1,3}(Q)]\!]_D$ then, there are MRA tuples $t_1 \in [\![f_{1,3}(P_1)]\!]_D$, $t_2 \in [\![f_{1,3}(P_2)]\!]_D$, and $t_3 \in [\![\mathrm{Comp}]\!]_D$ such that $t(X) = t_3(A)$, $t(Y) = t_2(Y)$, $t(Z) = t_3(Z)$, and $t_1(X) = t_3(A_1)$, $t_2(X) = t_3(A_2)$. Let $\mu_1 = h_{3,1}(t_1)$, $\mu_2 = h_{3,1}(t_2)$, and $\mu_3 = h_{3,1}(t_3)$. By the induction hypothesis in P_1 and P_2, $t_1 \in [\![f_{1,3}(P_1)]\!]_D$ and $t_2 \in [\![f_{1,3}(P_2)]\!]_D$ if and only if $\mu_1 \in [\![P_1]\!]_G$ and $\mu_2 \in [\![P_2]\!]_G$. By the definition of Comp^D, it holds that $\mu_1 \sim \mu_2$ and $\mu_1 \cup \mu_2 = \mu$. By the semantics of the SPARQL operator AND, it holds then that $\mu \in [\![Q]\!]_G$. Hence, $t \in [\![f_{1,3}(Q)]\!]_D$ if and only if $\mu \in [\![Q]\!]_G$.

(b) By definition,

$$\mathrm{card}(\mu, [\![Q]\!]_G) = \sum_{\substack{\mu_1 \in [\![P_1]\!]_G \\ \mu_2 \in [\![P_2]\!]_G \\ \mu_1 \sim \mu_2 \\ \mu = \mu_1 \cup \mu_2}} \mathrm{card}(\mu_1, [\![P_1]\!]_G) \times \mathrm{card}(\mu_2, [\![P_2]\!]_G),$$

$$\mathrm{card}(t, [\![f_{1,3}(Q)]\!]_D) = \sum_{\substack{t_1 \in [\![f_{1,3}(P_1)]\!]_D \\ t_2 \in [\![f_{1,3}(P_2)]\!]_D \\ t_3 \in [\![\mathrm{Comp}]\!]_D \\ \varphi(t_1,t_2,t_3)}} \mathrm{card}(t_3, [\![\mathrm{Comp}]\!]_D) \times \mathrm{card}(t_1, [\![P_1]\!]_D) \times \mathrm{card}(t_2, [\![P_2]\!]_D),$$

where $\varphi(t_1, t_2, t_3)$ is a condition coresponding to the compatibility, that is true if and only if the following statements hold:

i. $t_1(Y) = t(Y)$,
ii. $t_2(Z) = t(Z)$, and
iii. either

A. $(t_1(X) = t(X)$ and $t_2(X) = t(X)$,
B. $(t_1(X) = t(X)$ and $t_2(X) = t(X)$, or
C. $(t_1(X) = t(X)$ and $t_2(X) = t(X)$.

By the induction hypothesis, $\mathrm{card}(\mu_1, [\![P_1]\!]_G) = \mathrm{card}(t_1, [\![f_{1,3}(P_1)]\!]_D)$ and $\mathrm{card}(\mu_2, [\![P_2]\!]_G) = \mathrm{card}(t_2, [\![f_{1,3}(P_2)]\!]_D)$. By construction, $\mathrm{card}(t_3, [\![\mathrm{Comp}]\!]_D) = 1$. Hence, $\mathrm{card}(\mu, [\![Q]\!]_G) = \mathrm{card}(t, [\![f_{1,3}(Q)]\!]_D)$.

3. Case Q is a query $(P_1 \text{ EXCEPT } P_2)$. Let $?\bar{X}$ be the list of SPARQL variables in set $\mathrm{inScope}(Q)$. The MRA query $f_{1,3}(Q)$ is then $f_{1,3}(P_1) \setminus f_{1,3}(P_2)$.
   (a) By definition, $t \in [\![f_{1,3}(Q)]\!]_D$ if and only if $t \in [\![f_{1,3}(P_1)]\!]_D$ and $t \notin [\![f_{1,3}(P_2)]\!]_D$. By the induction hypothesis, the last condition is equivalent to $\mu \in [\![P_1]\!]_G$ and $\mu \notin [\![P_2]\!]_G$. By the SPARQL semantics, $t \in [\![f_{1,3}(Q)]\!]_G$ if and only if $\mu \in [\![Q]\!]_G$.
   (b) By definition, $\mathrm{card}(\mu, [\![Q]\!]_G) = \mathrm{card}(\mu, [\![P_1]\!]_G)$ and $\mathrm{card}(t, [\![f_{1,3}(Q)]\!]_D) = \mathrm{card}(t, [\![f_{1,3}(P_1)]\!]_D)$. By the induction hypothesis, $\mathrm{card}(\mu, [\![P_1]\!]_G) = \mathrm{card}(t, [\![f_{1,3}(P_1)]\!]_D)$. Hence, $\mathrm{card}(\mu, [\![Q]\!]_G) = \mathrm{card}(t, [\![f_{1,3}(Q)]\!]_D)$.
4. Case Q is a SPARQL query $(P_1 \text{ UNION } P_2)$. The MRA expression $f_{1,3}(Q)$ is then $f_{1,3}(P_1) \cup f_{1,3}(P_2)$.
   (a) By definition, $t \in [\![f_{1,3}(Q)]\!]_G$ if and only if $t \in [\![f_{1,3}(P_1)]\!]_D$ or $t \in [\![f_{1,3}(P_2)]\!]_D$. By the induction hypothesis, $t \in [\![f_{1,3}(Q)]\!]_G$ if and only if $\mu \in [\![P_1]\!]_G$ or $\mu \in [\![P_1]\!]_G$. By the SPARQL semantics, $t \in [\![f_{1,3}(Q)]\!]_G$ if and only if $\mu \in [\![Q]\!]_G$.
   (b) By definition,

$$\mathrm{card}(t, [\![f_{1,3}(Q)]\!]_D) = \mathrm{card}(t, [\![f_{1,3}(P_1)]\!]_D) + \mathrm{card}(t, [\![f_{1,3}(P_2)]\!]_D),$$
$$\mathrm{card}(\mu, [\![Q]\!]_G) = \mathrm{card}(\mu, [\![P_1]\!]_G) + \mathrm{card}(\mu, [\![P_2]\!]_G).$$

   By the induction hypothesis, $\mathrm{card}(t, [\![f_{1,3}(P_1)]\!]_D) = \mathrm{card}(\mu, [\![P_1]\!]_G)$ and $\mathrm{card}(t, [\![f_{1,3}(P_2)]\!]_D) = \mathrm{card}(\mu, [\![P_2]\!]_G)$. Hence $\mathrm{card}(t, [\![f_{1,3}(Q)]\!]_D) = \mathrm{card}(\mu, [\![Q]\!]_G)$.
5. Case Q is a SPARQL query $(P \text{ FILTER } \varphi)$ where φ is an atomic filter condition (i.e., a filter condition of the form $?X = c$, $?X = ?Y$, or $\mathrm{bound}(?X)$). The MRA expression $f_{1,3}(Q)$ is then $\sigma_\psi(f_{1,3}(P))$ where ψ is the MRA selection condition defined as follows:

$$\psi = \begin{cases} X = c \wedge \neg(X = \bot) & \text{if } \varphi \text{ is } ?X = c, \\ X = Y \wedge \neg(X = \bot) \wedge \neg(Y = \bot) & \text{if } \varphi \text{ is } ?X = ?Y, \\ \neg(X = \bot) & \text{if } \varphi \text{ is } \mathrm{bound}(?X). \end{cases}$$

(a) By definition, $t \in [\![f_{1,3}(Q)]\!]_D$ if and only if $t \in [\![f_{1,3}(P)]\!]_D$ and t satisfies condition ψ. It is not difficult to see that t satisfies condition ψ if and only if μ satisfies condition φ. By the induction hypothesis, $t \in [\![f_{1,3}(P)]\!]_D$ if and only if $\mu \in [\![P]\!]_G$. Hence, $\mu \in [\![f_{1,3}(Q)]\!]_D$ if and only if $\mu \in [\![Q]\!]_G$.

(b) By definition, if t and μ satisfy the respective conditions, then:

$$\mathrm{card}(t, [\![f_{1,3}(Q)]\!]_D) = \mathrm{card}(t, [\![f_{1,3}(P)]\!]_D),$$
$$\mathrm{card}(\mu, [\![Q]\!]_G) = \mathrm{card}(\mu, [\![P]\!]_G).$$

By the induction hypothesis, $\mathrm{card}(t, [\![f_{1,3}(P)]\!]_D) = \mathrm{card}(\mu, [\![P]\!]_G)$. Hence, $\mathrm{card}(t, [\![f_{1,3}(Q)]\!]_D) = \mathrm{card}(\mu, [\![Q]\!]_G)$.

6. Case Q is a SPARQL query (SELECT $?\bar{X}$ P). The MRA expression $f_{1,3}(Q)$ is then $\pi_{\bar{X}}(f_{1,3}(P) \bowtie \Delta_{\bar{Y}})$, where $\bar{X}$ is the corresponding set of attributes for the variables $?\bar{X}$ and $\bar{Y}$ is the correponding set of attributes for the variables in set $\mathrm{inScope}(P) \setminus \mathrm{inScope}(Q)$.

(a) By definition, $t \in [\![f_{1,3}(Q)]\!]_D$ if and only if $t(Y) = \perp$ for every attribute name $Y \in \bar{Y}$ and there exists a solution $t' \in [\![f_{1,3}(P)]\!]_D$ such that $t'(A) = t(A)$ for every attribute $A \in \bar{X} \setminus \bar{Y}$. Let $\mu' = h_{3,1}(t')$. By the induction hypothesis, $t' \in [\![f_{1,3}(P)]\!]_D$ if and only if $\mu' \in [\![P]\!]_G$. By construction $\mu = \mu'|_{?\bar{X}}$. Hence, $t \in [\![f_{1,3}(Q)]\!]_D$ if and only if $t \in [\![P]\!]_G$.

(b) By construction,

$$\mathrm{card}(t, [\![f_{1,3}(Q)]\!]_D) = \sum_{\substack{t'|_{\mathrm{inScope}(Q)} = t \\ t' \in [\![f_{1,3}(P)]\!]_D}} \mathrm{card}(t', [\![f_{1,3}(P)]\!]_D),$$
$$\mathrm{card}(\mu, [\![Q]\!]_G) = \sum_{\substack{\mu'|_{\mathrm{inScope}(Q)} = \mu \\ \mu' \in [\![P]\!]_G}} \mathrm{card}(\mu', [\![P]\!]_G),$$

By the induction hypothesis, $\mathrm{card}(t', [\![f_{1,3}(P)]\!]_D) = \mathrm{card}(\mu', [\![P]\!]_G)$. Hence, $\mathrm{card}(t, [\![f_{1,3}(Q)]\!]_D) = \mathrm{card}(\mu', [\![Q]\!]_G)$.

Hence, the triple $(f_{1,3}, g_{1,3}, h_{3,1})$ is a simulation of SPARQL in MRA. □