Multiset semantics in SPARQL, Relational Algebra and Datalog

Response Letter to Reviewers

May 2025

We thank the reviewers for their insightful reviews and feedback. We carefully followed and incorporated all your comments. In some cases, we provide a response following the comment. In other cases, we use a check mark to indicate that an issue was solved. In case of need, the most relevant changes are highlighted (in blue) in the new version of the article.

1 Review #1

The manuscript studies the expressive power of a fragment of SPARQL under multiset semantics by identifying both a version of Datalog with multiset semantics and a fragment of the relational algebra with multiset semantics that have the exact same expressive power as the fragment of SPARQL considered by the authors.

The material is well organized and presented in an easy-to-follow way. The results are correct (with a minor exception, but that is not difficult to fix—see below) and, to the best of my knowledge, this is the first publication showing these results. For these reasons, I am generally in favor of accepting this work for publication in the journal. However, there are three main points that the authors need to work on a bit more, plus several other more minor points.

1.1 Main points

M1. The mapping from the multiset relational algebra (MRA) to SPARQL in Section 8.1 is not complete. In particular, the function g31 that translates every MRA database into a "SPARQL database" (i.e., an RDF graph) does not consider the cardinalities of the tuples in the multiset relations. More specifically, the RDF triples produced by the function β that the definition of function g31 is based do not capture the cardinalities. Due to this issue, the claim that "MRA can be simulated in SPARQL" (Lemma 13) is not fully supported. Fixing this issue may be a bit of work but I don't foresee any inherent complications in it: The idea would be to extend the β function such that the resulting RDF triples also describe the cardinalities of the represented tuples, and then to extend the query-related mapping functions (f13 and h13) accordingly.

Response: We changed the definition of functions β. Now, given a relation r and a tuple t, the function $\beta(t, r)$ returns a set of RDF triples for each copy of t.

M2. The manuscript provides very little (almost nothing) in terms of motivation of the presented work. The authors should extend the introduction (ideally even the abstract) with a few statements to tell the reader why the presented work is of relevance and what the potential impact of the presented results may be. (I see that the authors base their work on a few questions, which are listed in the 'Objectives and Contributions' part of the introduction. Yes, it is not clear what motivates these questions. Why might it be interesting to have answers to these questions?)

Response: We improved the motivation presented in the Introduction.

M3. The 'Related work and Conclusions' section (Sec.9) needs to be improved in two ways. First, there are several sentences for which it is not clear what they are supposed to mean: i) "the multiset semantics of SPARQL has not been systematically addressed" ¡- What exactly do the authors mean by "systematically address[ing]" the multiset semantics of SPARQL? ii) "the goal of characterizing the multiset algebraic and/or logical structure of the operators in SPARQL." ¡- I don't understand what this goal is meant to be. (What is a "multiset algebraic structure of operators"? What is a "logical structure of operators"? "characterizing" in terms of what?) iii) "this study shows the complexities and challenges that the introduction of multisets brings to [...] SPARQL" ¡- I don't see how the presented work shows such complexities and challenges. Second, the related works part should be improved as well. At the moment, it appears to be a bit unorganized and it is focused only on SPARQL. Related formalisms, specifically the ones that the considered variants of Datalog and MRA are based on, should be discussed here as well. In particular, I would expect the commonalities and the differences (if any) between these considered variants and the variants in the literature to be elaborated very clearly.

Response: We improved the description of the Related Work.

1.2 Additional things

A1. The formal semantics for multisets in SPARQL should not be claimed as a contribution (as done in Sec.1, line 23 on page 3) because this has already been provided by Perez et al. in 2006 and is the basis of the SPARQL specifications already since version 1.0 of SPARQL.

Response: We have changed the mentioned paragraph to avoid such a claim, giving more emphasis to our contributions.

A2. In a similar sense (and also related to my point M3 above), it is not clear to me to what extend the authors can actually claim that they ”develop a logical formalism for multisets” (NRMD$\neg$) and that they ”develop the relational counterpart of this fragment.” In fact, it is not crystal clear from the manuscript which parts of these formalisms are taken from the literature and exactly which parts have been added. The manuscript needs to be improved in this aspect!

Response: We agree with this comment. We changed the Introduction to clarify the specific contributions of the work.

A3. Why exactly does Table 1 contain the same SPARQL expression (”A AND B”) in different rows? There should be a brief explanation of this.

Response: We revised Table 1 to improve the clarity of the explanation. We also included a brief explanation about the use of the AND operator (page 3).

A4. A4. What does it mean that ”the root of ti unifies with Ai under θ''? (line 39 on page 10)

Response: We revised the definition of Derivation Tree to improve its clarity.

A5. Line 45 on page 10 considers the case that ”the root of t is a colored version of F'' where t is a derivation tree. This is not possible because, by the given definition of derivation trees, the roots of these trees are not colored (see lines 34 and 41 on page 10), as is also visible in Fig.3. So, something seems to be wrong or missing here.

Response: We revised the description to improve its clarity.

A6. The manuscript is inconsistent and sloppy in terms of how it calls the things that are translated into one another. For instance, in lines 37-40 on page 13, it should be ”SPARQL pattern” instead of ”SPARQL query”, it should be ”RDF graph” instead of ”SPARQL database”, it should be ”NRMD$\neg$ answer” instead of ”NRMD$\neg$ query solution”, and it should be ”multiset of mappings” (”multiset of solution mappings”) instead of ”SPARQL query solution”. There are many more such inconsistencies throughout the whole manuscript.

Response: We fixed the inconsistencies in the article.

A7. Similarly, the manuscript sometimes uses the word ”multiplicity” (e.g., line 43 on page 20, Def.23 on page 27) and more often the word ”cardinality”. Pick one and be consistent about it!

Response: We just use the term ”cardinality” in the new version of the article.

A8. Def.18: SPARQL is not defined for ”a multiset of RDF triples” – there is no such thing in the context of RDF and SPARQL.

Response: The error was fixed.

A9. The formula of $\sum_V$ at the end of page 25 seems incorrect (incomplete) to me. For instance, just saying “$s, p \in V$” in the first subset means that the condition “$S = P$” will be added to the selection operator if the subject and the predicate of the given triple pattern are variables, no matter whether they are the same or two different variables! This doesn't seem to make sense. Instead, in addition to “$s, p \in V$” there should also be the condition that “$s = p$”. Same issue for the other two subsets in the formula of $\sum_V$.

Response: We changed the definition of the function that translates a triple pattern into a MRA expression. It should be clearer now.

1.3 Minor things

- ✓ m1. Line 35 on page 3 mentions a union operator using a symbol that does not show up in the actual definitions in Sec.5.
- ✓ m2. The title of Sec.4 (page 9) is missing a closing parenthesis.
- ✓ m3. Line 44 on page 10 talks about ”nr-Datalog¬”, which has not been introduced. Probably this was meant to be ”NRMD¬”.
- ✓ m4. Line 8 on page 12 mentions ”relation schema R = A1,...,An” which is not entirely correct. By the definitions given in the previous paragraph, R and A1,...,An are different things.
- ✓ m5. Line 14 on page 12: ”set of relational schemas” → ”set of relation schemas”
- ✓ m6. Lines 15-16 on page 12: What does it mean that a ”relation ri satisfies the schema Ri”?
- ✓ m7. The definition in lines 35-36 on page 12 assumes that attribute A is in $\widehat{E}_i$, which should be mentioned explicitly as part of this definition.
- ✓ m8. Line 1 on page 13: How is this notion of ”is equal to” defined?
- ✓ m9. The definition of Eval(E,D) on page 13 assumes that E and D are over the same relational database schema. This assumption should be made explicit in lines 6-7 and, also, by adding ”over T” at the end of the first sentence on line 8.
- ✓ m10. Line 49 on page 13 mentions “each term $t \in G$” which is incorrect. G is a set of triples, not a set of terms.

✓ m11. The notation used in line 34 on page 17 has not been introduced. The notation for writing multisets as given in the paper is a different one (namely with a card function).

✓ m12. It is not clear whether the variables used in Datalog substitutions are of the same kind as the variables in SPARQL solution mappings. According to the formulas in Def.8 that seems to be the case, but later in Def.11, the authors seem to make a difference.

✓ m13. Def.9: the symbols p, c1, and cn (as used in the formula of the definition) are not introduced within the definition.

✓ m14. Line 40 on page 18: What is "$gp(L)^{D}_{\Pi}$" ?

✓ m15. Point 2 at the end of page 18 aims to define T(vr(Ri,L)) but the bullet points that follow introduce only T(Ri).

✓ m16. Line 15 on page 19: "(a0,a0,a0)" → "(NULL,NULL,NULL)"

✓ m17. Similar issue with a0 in line 37 on page 19.

✓ m18. Line 50 on page 19 mentions "$\theta\mu = \{X1/c1, ..., Xn/cn\}$". Why this notation? Where is it introduced? Why not the same notation as for the SPARQL solution mappings? Both are (partial) functions after all.

✓ m19. Line 41 on page 20: "under" → "over"

✓ m20. Def.14: It is inconsistent to use the symbol R here. It should be r instead (to be consistent with Sec.5.1). Same issue in Sec.7.2.1 and in Def.15.

✓ m21. Line 41 on page 23: "MRA relations" are undefined. Sec.5.1 calls them "multiset relations"!

✓ m22. Same issue in lines 24-25 on page 25.

✓ m23. Line 45 on page 23: "... to IRIs[, and tuples to IRIs]."

m24. Lines 28-29 on page 24: What are u1 and u2 in the given SPARQL pattern?

✓ m25. Line 25 on page 25: "a set of tuples" → "a multiset of tuples" !!

✓ m26. Line 42 on page 25: "a multiset of RDF triples" → "a set of RDF triples"

✓ m27. Line 1 on page 26: "Selection(s,p,o)" → "Selection(T)"

✓ m28. Def.22: "from graph patterns" → "from normalized graph patterns"

✓ m29. Def.22: "whose selection formulas" → "whose filter conditions"

✓ m30. Table 7: What is $P_\emptyset$ (in the first row of the table)? In fact, I don't think that this row is needed. The base case of the syntax are triple patterns, which are covered by the next row in the table.

✓ m31. Line 43 on page 27: "RDF mapping" → "SPARQL mapping"

2 Review #2

The paper "Multiset semantics in SPARQL, Relational Algebra and Datalog" investigates the relationship between a fragment of SPARQL (the relational core), the Multiset Relational Algebra (MRA), and Non-recursive multiset Datalog with safe negation (NRMD$^{\neg}$). The authors prove that all of these frameworks have the same expressive power.

The paper is well-structured. First, the concept of query languages and their expressive power is formally defined. Then the three different frameworks are introduced in detail. This is followed by sections which always focus on two frameworks, define simulations from the first framework to the second and back and thereby show that these have the same expressive power. These sections use the notation from the introducing section such that it is easy to follow which exact mapping is defined in which moment and why this mapping is needed. The paper then discusses related work and concludes by listing the findings.

From my really subjective view (more as an interested reader), I really enjoyed the section about the problems of translating SPARQL Filters to Datalog which was well-explained and illustrated by examples.

While I appreciate the clear structure of the paper and also its contribution, I see two main points for improvements:

2.1 Main points

M1. I could not always fully understand the details of the definitions or formalisations. This was for example the case for the section about MRA where it is important to understand how r, R, $\hat{R}$, t, T and $\hat{r}$ relate or in Table 4 where the Datalog rules for SPARQL queries are introduced but not further explained. Here I would recommend to spend more times on details and also add examples where possible

Response: We fixed the notations and introduced examples.

M2. I know that this is for space reasons, but in my opinion it is a little bit unfortunate that the main contribution of the paper, the proofs of the frameworks having the same expressive power can only be found in the appendix. I would recommend to either put longer proof sketches into the paper itself or to add examples illustrating the mappings such that the reader can better appreciate the contribution.

Response: We introduced examples to illustrate the mappings.

2.2 Additional things

A1. Definition 4: term(t) for each term $t \in G \rightarrow G$ is a set of triples, so how is that meant?

Response: Thank you for spotting it out. We should have written: "for each t such that there exists a triple $(s, p, o) \in G$ with $s = t$, $p = t$, $o = t$." Thus, the symbol t denotes all the terms in the graph, that is, elements in subject, predicate, and object position. Intuitively, atoms term(t) list all terms in the graph, and atoms eq(t, t), that each term is equal to itself. We fixed the text, added a clarifying explanation, and an example.

A2. Mapping on page 18: the definition of $T(vr(R_i, L)$ is defined using $T(R_i)$, but the latter is never defined, also $gp(L)^D_\pi$ is not introduced

Response: The function $T(R)$ is defined as a module of the function of $gp(L, \Pi)$. These two functions are mutually recursive. To clarify the notation, we indicate that both functions are defined together, and make it explicit the paramters of $T(R)$ that were implicit.

A3. Introduction of multiset relations (page 12): it could be that the reason is that I am less familiar with MRA than with SPARQL and Datalog, but that part was difficult for me to read and would benefit from examples or at least from more detailed explanations.

Response: We improved the notations.

A4. Page 17, table 4, (P_1 *AND* P_2): the need for the predicate comp becomes only clear after reading section 8.2.1 where we have a similar construct. This is far too late, the rule needs explanation.

Response: We added the definition of predicate comp in Definition 4 and illustrated it in Example 3.

A5. If the authors show that SPARQL and NRMD$^\neg$ have the same expressive power and that NRMD$^\neg$ and MRA have the same expressive power, then it directly follows that also SPARQL and NRMD$^\neg$ have the same expressive power. Why did the authors choose to still give a proof for the latter?

Response: Because our objective is to study SPARQL and this translation gives direct translation rules.

A6. page 11: In the proof for lemma 2, rules are rewritten and there is a claim that it is clear that the resulting program is normal because the original program was safe. I don't see why $var(L_2) = var(L_1)$, I can only see the inclusion. Maybe that could be clarified?

Response: Thank you for the observation. We added the missing rule 2.c.

A7. page 12: "A relational database schema is a set of relational schemas. Given a relational database schema $T = \{R1, ..., Rn\}$, a multiset relational database over T is a set of multiset relations $\{r1, ..., rn\}$ where each relation ri satisfies the schema Ri". → what does it mean to satisfy a schema?

Response: We fixed the sentence. It now says: "the relation r_i is defined over the schema R_i."

A8. Lemmas 8, 9, etc. I personally would prefer to see the direct claims instead of the lemmas, mainly because the functions f, g, h are already defined in the text, so it would make sense to directly state that these are simulations. I am furthermore aware that the proofs don't fit in the text, but I would prefer to get some more detail about the respective proofs here. An alternative could be to provide a small example how the simulation works.

Response: We preferred to leave the text as it is because we consider these statements are important to highlight.

A9. Page 18: NRMD$^{\neg}$ to SPARQL: I think an interesting detail of the mapping is that the predicate name is handled like the arguments and also is represented by a special triple (u, α_0, p). I would mention that somewhere.

Response: We added Example 6 to explain it.

A10. Page 23, Definition 18: SPARQL is defined on RDF graphs which are sets, the definition maps to multisets of RDF triples. Here I see a mismatch which needs to be resolved.

Response: Yes, the translation is involved. Essentially, each MRA tuple (e.g., with multiplicity 2) gets a unique identifier. With this identifier, we define two sets of triples to describe each individual occurrence of the tuple. We added an example for clarity.

A11. Related work and conclusions: I think that the contribution is interesting, I would however expect the authors to write a little bit more about the relevance of the findings. Maybe some practical consequences for implementers or the fact that proven results for one framework then can be transferred to others?

Response: We took the comment and expanded the conclusions accordingly.

2.3 Minor comments

✓ page 7+8: var(t), var(P) -¿ var is used but not defined

✓ - page 8, Definition 3: “Every sub-pattern… holds that …” -¿ “For every sub-pattern… it holds that …”

✓ - page 8: “..., the following equivalences are hold”-¿ “..., the following equivalences hold”

✓ - page 14: the “solution to the query on the right side” is used twice, I guess the first one should be the left side?

✓ - page 16: rules presented in Table 5 -¿ Table 4

✓ - page 18: especial -¿ special

✓ - page 18, definition 9: $\{NULL, NULL, NULL\}$ *to* $\{(NULL, NULL, NULL)\}$, as $g_{2,1}$ maps into a set of triples

✓ - page 18: “…is equivalent to R but have literal L as head.” → has

✓ - page 19, Example 4: $gp((q(X), \pi)) \rightarrow gp(q(X), \pi)$

✓ - page 19: “This is done using the additional triple $\{NULL, NULL, NULL\}$...”, most likely $(NULL, NULL, NULL)$ is meant?”

✓ - page 19, Example 5: $f_{2,1}((q(X), \Pi) \rightarrow f_{2,1}((q(X), \Pi))$

✓ - page 20, Theorem 1, page 23, Theorem 2, page 28, Theorem 3: “If follows from Lemma X and Lemma Y” → The Claim follows from…

✓ - page 20: “For each tuple t in $r, \Sigma(r)$ contains a fact f of the form $p(c1, \ldots, cn)$ where p is R,...” where p is the image of R (or however we want to use the aforementioned mapping)

Multiset semantics in SPARQL, Relational Algebra and Datalog

Renzo Angles [a], Claudio Gutierrez [b] and Daniel Hernández [c]

[a] *Department of Computer Science, Faculty of Engineering, Universidad de 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 analyzes and characterizes the algebraic and logical structure of the multiset semantics for SPARQL patterns involving AND, UNION, FILTER, EXCEPT, and SELECT. To do this, we align SPARQL with two well-established query languages: Datalog and Relational Algebra. Specifically, we study (i) a version of non-recursive Datalog with safe negation extended to support multisets, and (ii) a multiset relational algebra comprising projection, selection, natural join, arithmetic union, and except. We prove that these three formalisms are expressively equivalent under multiset semantics.

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

1. Introduction

Informally speaking, multisets are sets in which 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 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 includes 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 for their extensions to multisets, whose theory is complex (particularly the 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, we will show the different extensions to multisets of set operators found in the literature. Consider the following multiset relations: $R(W,X) = \{\!\{(a,b),(a,b),(a,d)\}\!\}$, $S(W,X) = \{\!\{(a,b)\}\!\}$ and $T(Y,Z) = \{\!\{(b,c),(b,c)\}\!\}$. For the first relation, R is the name of the relation, W and X are the attributes that conform the schema of R, R contains three tuples, and the tuple (a,b) is duplicated (i.e., its cardinality is 2). A

[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. Let R, S and T be multiset relations satisfying that R and S have the same attributes, and T does not have attributes in common with R. The cardinality of an element x in a relation R is represented as $R(x)$. Note that SPARQL works with multisets of bindings, whose corresponding schema is a set of variables.

Operation	**Operator**	**Cardinality for** x	**SQL**	**SPARQL**
Selection	$\sigma_\varphi(R)$	$\begin{cases} R(x) & \text{if } x \text{ satisfies } \varphi, \\ 0 & \text{otherwise.} \end{cases}$	`SELECT * FROM` R `WHERE` φ	R `FILTER (`φ`)`
Cartesian product	$R \times T$	$R(x) \times T(x)$	R `CROSS JOIN` T	R `AND` T
Join	$R \bowtie_\varphi T$	$R(x) \times T(x)$	`(`R `CROSS JOIN` T`) WHERE` φ	`(`R `AND` T`) FILTER (`φ`)`

Max-union	$R \sqcup S$	$\max(R(x), S(x))$	`(`R `UNION ALL` S`) EXCEPT ALL`
			`(`R `INTERSECT ALL` S`)`
				–

Arithmetic union	$R \cup S$	$R(x) + S(x)$	R `UNION ALL` S	R `UNION` S
Min-intersection	$R \cap S$	$\min(R(x), S(x))$	R `INTERSECT ALL` S	–
Max-intersection	$R \sqcap S$	$S(x) \times S(x)$	R `NATURAL JOIN` S	R `AND` S
Arithmetic difference	$R - S$	$\max(0, R(x) - S(x))$	R `EXCEPT ALL` S	–
Existential negation	$R \setminus S$	$\begin{cases} R(x) & \text{if } S(x) = 0, \\ 0 & \text{otherwise.} \end{cases}$	`SELECT * FROM` R `WHERE` x `NOT IN (`S`)`	R `MINUS` S
Projection	$\pi_{Atts}(R)$	$\sum_{t \in R,\, t[Atts]=x} R(x)$	`SELECT` $Atts$ `FROM` R	`SELECT` $Atts$

similar description can be given for the relations S and T. Note that R and S have the same attributes, while T does not have attributes in common with R and S.

- The *selection* returns the tuples satisfying a given condition but keeping cardinalities. For example, $\sigma_{X='b'}(R)$ returns the multiset $\{\!\!\{ (a,b), (a,b) \}\!\!\}$ with schema (W, X).
- The *cartesian product* results in the multiplication of the cardinalities. For example, $R \times T$ returns the multiset $\{\!\!\{ (a,b,b,c), (a,b,b,c), (a,b,b,c), (a,b,b,c), (a,d,b,c), (a,d,b,c) \}\!\!\}$ with schema (W, X, Y, Z).
- The *join* results in the multiplication of the cardinalities, as it is expressed as a cartesian product followed by a selection. For example, $R \bowtie_{X=Y} T$ returns the multiset $\{\!\!\{ (a,b,b,c), (a,b,b,c), (a,b,b,c), (a,b,b,c) \}\!\!\}$ with schema (W, X, Y, Z).
- The *max-union* takes the maximum number of occurrences of an element. For example, $R \sqcup S$ returns the multiset $\{\!\!\{ (a,b), (a,b), (a,d) \}\!\!\}$ with schema (W, X).
- The *arithmetic union* adds up cardinalities. For example, $R \cup S$ returns the multiset $\{\!\!\{ (a,b), (a,b), (a,d), (a,b) \}\!\!\}$ with schema (W, X).
- The *min-intersection* takes the minimum number of occurrences of each element in the intersection. For example, $R \cap S$ returns the multiset $\{\!\!\{ (a,b) \}\!\!\}$ with schema (W, X).
- The *max-intersection* returns the product of the cardinalities of each element in the intersection. For example, $R \sqcap S$ returns the multiset $\{\!\!\{ (a,b), (a,b) \}\!\!\}$ with schema (W, X).
- The *arithmetic difference* subtracts the cardinalities of the elements up to zero. For example $R - S$ returns the multiset $\{\!\!\{ (a,b), (a,d) \}\!\!\}$ with schema (W, X).
- The *existential negation* returns the elements in the first multiset that do not occur in the second one, but preserving the cardinalities. For example $R \setminus S$ returns the multiset $\{\!\!\{ (a,d) \}\!\!\}$ with schema (W, X).
- The *projection* reduces the number of attributes in each tuple, and gives rise to new cardinalities for the resulting tuples. For example $\pi_W(R)$ returns the multiset $\{\!\!\{ (a), (a), (a) \}\!\!\}$ with schema (W).

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. Also note that SPARQL uses the AND operator to implement cartesian product and max-intersection. The first case occurs because R and T do not have variables in common, and the second case occurs because R and S have the same set of variables.

The landscape of operators over multisets poses important challenges for integrating multisets in query languages. First, as shown in Table 1, some operators exhibit different semantics when applied to multisets. Second, while relational algebra and SQL support all the semantics listed, SPARQL and Datalog only support a subset. Third – and this is the main motivation for our research – it remains unclear whether there exists an optimal set of multiset operators for SPARQL, and if so, which one it is. To tackle these questions, it is essential to understand how formalisms that are "closed" with respect to SPARQL behave, and how their design and behavior can inform or be translated into SPARQL. In technical terms, this means analyzing the expressive power of SPARQL regarding multisets. To this end, we focus on two natural and well-studied reference points: relational algebra and Datalog. That is the aim of this article. Next, we review the existing literature about multisets.

Related Work. First, we consider the research works that define general algebras for manipulating bags. Albert [4] extended typical set operations (union, intersection, difference, and boolean selection) to bags, and demonstrated that some of the algebraic properties for sets fail for multisets. Grumbach et al. [6] introduced a bag algebra, called BALG, that extends relational operations to handle duplicates. This paper shows that BALG is more expressive than standard relational algebra because it can count duplicates, but it still has low data complexity (LOGSPACE). Grumbach and Milo [7] focused on designing bag algebras that are both expressive and computationally tractable. They introduce restricted forms of projection and join to maintain tractable data complexity. Libkin and Wong [5, 9] introduced BQL, a query language for handling bags and aggregate functions (sum, count, avg.). They show that BQL is more expressive than traditional set-based languages, and shows that after incorporating structural recursion to BQL, it is able to express all primitive recursive functions, significantly increasing its computational power. Ricciotti and Cheney [19] explored how to mix set and bag semantics in query languages, addressing practical needs found in SQL (e.g., SELECT versus SELECT DISTINCT). They propose a formal model that supports both semantics and allows translation between them.

The first attempt to extend the relational algebra to include multisets was made by Dayal et al. [10]. In this work, the authors introduced a multiset relational algebra (formed by the operators of projection, selection, join, max-union, arithmetic union, min-intersection and arithmetic difference) and studied their algebraic properties. This work laid the groundwork for formalizing bag semantics in relational query languages. Klauser and Goodman [11] provided a semantic framework for understanding the role of multirelations (relations with duplicates) at the conceptual level. The authors explain how any query language can be extended consistently to have full multirelational expressiveness. Afrati et al. [16] studied query containment in relational databases under bag semantics and bag-set semantics (duplicates allowed in intermediate steps but not in final output). The authors identify conditions under which containment is decidable and provide complexity results. Console et al. [12] investigated fragments of bag relational algebra, focusing on their expressive power. The authors also study query answering over bags with nulls (i.e. under incomplete data).

Multisets have also been the subject of study in the context of Datalog, with various extensions proposed to support bag semantics. Mumick et al. [14] defined the Magic Sets transformation for optimizing recursive queries, and described how to adapt this technique to support duplicates. They also showed how to efficiently evaluate recursive queries under multiset semantics. In a subsequent work [13], Mumick et al. extended the Magic Sets technique to support duplicates and aggregate functions in recursive queries. The authors also studied the challenges of preserving correct bag semantics when applying recursion and aggregation. Cohen [15] studied the problem of query equivalence under bag semantics. This work includes complexity results and demonstrates that equivalence checking is significantly harder under bag semantics. Bertossi et al. [14] developed a translation of Datalog under bag semantics into warded Datalog$^{\pm}$, a well-behaved extension under set semantics. The authors investigated the properties of the resulting Datalog$^{\pm}$ programs, the problem of deciding multiplicities, and expressibility of some bag operations.

For SPARQL – the standard query language for RDF databases – Pérez et al. [29] provided the first formal treatment of its multiset semantics. This work influenced the definition of SPARQL 1.0 [30] and SPARQL 1.1 [31], whose semantics are based on operations over multisets of mappings (although a database is a set of RDF triples).

Schmidt et al. [32] presented a formal framework for SPARQL query optimization, addressing both set and bag semantics. The authors analyzed the algebraic properties of SPARQL operations like OPTIONAL, UNION, and FILTER under multisets, and introduced equivalence rules and normal forms for optimizing queries. Kaminski et al. [33] presented a formal investigation of subqueries and aggregate functions in SPARQL 1.1, focusing on their semantics under multisets. The authors analyzed the expressive power of these constructs, showing that SPARQL 1.1 is strictly more expressive than SPARQL 1.0 due to these features.

Finally, we review research articles that present comparisons and translations among SPARQL, relational algebra, and Datalog. Cyganiak [34] was among the first to translate a core fragment of SPARQL into relational algebra. Polleres [35] proved the inclusion of the fragment of SPARQL patterns with safe filters into Datalog by providing a precise and correct set of rules. Schenk [36] proposed a formal semantics for SPARQL based on Datalog, but concentrated on complexity more than expressiveness issues. Both Polleres and Schenk did not consider the multiset semantics of SPARQL in their translations. Angles and Gutierrez [37] studied the expressive power of SPARQL by providing a translation to non-recursive safe Datalog with negation. Chebotko et al. [38] addressed the problem of translating SPARQL queries into SQL while preserving bag semantics. The authors proposed a formal translation framework that captures the subtleties of OPTIONAL, UNION, and FILTER, and ensures that duplicates in the result sets are handled correctly when mapped to relational databases. Angles and Gutierrez [23] studied the multiset semantics of SPARQL patterns by translating its patterns into two languages: a version of multiset relational algebra and multiset non-recursive Datalog with safe negation. Angles et al. [39] implemented the translation from SPARQL to Datalog within the Vadalog system [40].

Objectives and Contributions. The main objective of this article is to examine the theoretical foundations of SPARQL's multiset semantics. To do so, we compare it with classical algebraic and logical frameworks – specifically, Relational Algebra and Datalog. We focus on the SPARQL fragment built from AND, UNION, FILTER, EXCEPT, and SELECT, characterizing its structure and proving its expressive equivalence with corresponding fragments of Relational Algebra and Datalog.

The specific contributions of our research are as follows:

(1) Based on the work of Mumick et al [13], who defined the multiset semantics for Datalog without negation, we defined a version called *Non-Recursive Multiset Datalog with Safe Negation* (NRMD$^\neg$). The definition of NRMD$^\neg$includes negation and follows a proof-theoretic semantics.

(2) Based on the work of Dayal et al. [10], who extended the relational algebra to include multiset relations, we defined a *Multiset Relational Algebra* (MRA). The definition of MRA includes the operators of projection (π), selection (σ), natural join ($\bowtie$), arithmetic union ($\cup$) and filter difference ($\setminus$), all of them working under multiset semantics.

(3) We show the equivalence among the aforementioned SPARQL fragment, MRA and NRMD$^\neg$ by providing translations for databases, queries, and answers. Table 2 shows a glimpse of these translations, whose details are developed in this paper.

This paper extends a previously published conference paper [23]. Herein, we provide extended discussion throughout, we extend the study to 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 also consider bag semantics. Some of the additional contributions of this paper come from Hernandez's Ph.D. thesis [41].

The rest of the article is organized as follows. Section 2 presents basic concepts and notations. The SPARQL query language is defined in Section 3. Non-recursive Multiset Datalog with Safe Negation (NRMD$^\neg$) is defined in Section 4. The Multiset Relational Algebra (MRA) is defined in Section 5. The equivalence between SPARQL and NRMD$^\neg$ is presented in Section 6. The equivalence between MRA and NRMD$^\neg$ is presented in Section 7. The equivalence between MRA and SPARQL is presented in Section 8. Conclusions are presented in Section 9.

Table 2

SCHEMA OF CORRESPONDENCES AMONG: SPARQL graph patterns; 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 $card : S \rightarrow \mathbb{N}^+$ is a function that defines the cardinality 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, $\text{card}(a, M) = n$ denotes that $a \in \text{set}(M)$ and the cardinality of a in M is n, and usually write it as $(a, n) \in M$. Abusing notation, we write $\text{card}(a, M) = 0$ if $a \notin \text{set}(M)$ and $a \in M$ when $\text{card}(a, M) \geqslant 1$. In what follows we will prefer these formal notions instead of the informal and intuitive $\{\!\{a, a, a, b\}\!\}$.

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 query answers in $\mathcal{L}$, and* $\text{Eval} : \mathcal{Q} \times \mathcal{D} \rightarrow \mathcal{S}$ *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 query answer 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 $q : \mathcal{D} \rightarrow \mathcal{S}$ 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 [42], Kontchakov et al. [43], and Angles and Gutierrez [44] use this notion to compare different fragments of SPARQL.

$$\begin{tikzcd}
\mathcal{D}_1 \arrow[r, "g_{12}"] \arrow[d, "\mathcal{Q}_1"'] & \mathcal{D}_2 \arrow[r, "g_{23}"] \arrow[d, "f_{12}(\mathcal{Q}_1)"'] & \mathcal{D}_3 \arrow[d, "f_{12} \circ f_{23}(\mathcal{Q}_1)"'] \\
\mathcal{S}_1 & \mathcal{S}_2 \arrow[l, "h_{12}"'] & \mathcal{S}_3 \arrow[l, "h_{23}"']
\end{tikzcd}$$

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 $(f_{12} \circ f_{23},\ g_{12} \circ g_{23},\ h_{23} \circ h_{12})$ where $\circ$ denotes the composition of functions (e.g., $g_{12} \circ g_{23}$ denotes the function from D_1 to D_3 that results from composing g_{12} and g_{23}).

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

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* query answer 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 $\mathcal{L}_1$.*

The above definition of generalized expressive power is implicit in the translations by Polleres [35], Angles and Gutierrez [23, 37], and Polleres and Wallner [20].

Observe that the extended notion defined above defines a partial order: the containment relation on the equivalence classes over the relation $\cong$. In fact, reflexivity and antisymmetry follow directly from the definition, while 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.

3. Multiset SPARQL

SPARQL [30, 31] is the standard query language for RDF. In this paper we study a fragment of SPARQL, the "relational core", described by Angles and Gutierrez [23], 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 [23, 43] 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 **I** and **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*

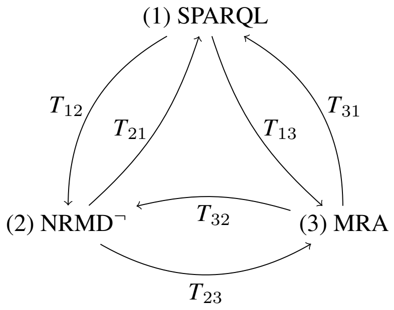

Fig. 2. The triangle of simulations among SPARQL, Non-Recursive Multiset Datalog with Safe Negation (NRMD$^\neg$), and Multiset Relational Algebra (MRA) described in this paper. The query languages are identified by numbers, and T_{ij} denotes the simulation of language i using language j.

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.

A SPARQL database will be a set of RDF triples.

Note: In addition to $\mathbf{I}$ and $\mathbf{L}$, SPARQL admits 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 [45].

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. We denote by $\mathrm{var}(\varphi)$ the set of variables occurring in φ.

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*. We will assume that a triple pattern has at least one variable.
- 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 the variable $?X$ to the term c. Given a finite set of variables W, the restriction of a mapping μ to W, denoted $\mu_{|W}$, is a mapping μ' that satisfies $\mathrm{dom}(\mu') = W \cap \mathrm{dom}(\mu)$ and $\mu'(?X) = \mu(?X)$ when $?X \in \mathrm{dom}(\mu')$. 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 $\mu_1(?X) = \mu_2(?X)$, 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 that occur 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 $\mathrm{card}(\mu, \Omega) = 0$.

Table 3

Evaluation of complex filter conditions [30, §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

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:

- If φ is $?X = c$ and $c \in \mathbf{T}$, then: (a) If $?X \in \mathrm{dom}(\mu)$ then $\mu(\varphi) = \mathit{true}$ when $\mu(?X) = c$ and $\mu(\varphi) = \mathit{false}$ otherwise; (b) If $?X \notin \mathrm{dom}(\mu)$ then $\mu(\varphi) = \mathit{error}$.
- If φ is $?X = ?Y$ and $?X, ?Y \in \mathrm{dom}(\mu)$, then $\mu(\varphi) = \mathit{true}$ when $\mu(?X) = \mu(?Y)$, and $\mu(\varphi) = \mathit{false}$ otherwise. If $?X \notin \mathrm{dom}(\mu)$ or $?Y \notin \mathrm{dom}(\mu)$ then $\mu(\varphi) = \mathit{error}$.
- If φ is $\mathrm{bound}(?X)$ and $?X \in \mathrm{dom}(\mu)$ then $\mu(\varphi) = \mathit{true}$; otherwise $\mu(\varphi) = \mathit{false}$.
- 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 $\mathrm{set}(M) = \{\mu \mid \mu \in M_1,\ \mu(\varphi) = \mathit{true}\}$ 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. [43]. Kontchakov et al. [43] 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 SPARQL 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 [31] defines a finite set of variables, called *in-scope*, that include all variables of a SPARQL pattern P that can occur in the solution mappings 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 $?Z$ that 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):*

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

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 $\text{inScope}(P)$, is defined recursively as follows:

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

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

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 $?X \notin \text{inScope}(P_1)$, and $\text{inScope}(P_2) \neq \text{inScope}(P_3)$. We next present a method to normalize these patterns.

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

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

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

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

$$\begin{aligned}
(P_1 \text{ FILTER } \varphi) &\equiv (P'_1 \text{ FILTER } \varphi),\\
(P_2 \text{ UNION } P_3) &\equiv (P'_2 \text{ UNION } P'_3),
\end{aligned}$$

$(P_2 \text{ EXCEPT } P_3) \equiv (P'_2 \text{ EXCEPT } P'_3)$.

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, $\text{var}(\varphi) \subseteq \text{inScope}(P'_1)$ and $\text{inScope}(P'_2) = \text{inScope}(P'_3)$. Hence, these equivalences can be used to normalize SPARQL patterns. □

Example 2. *Let P 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 $(a, \text{email}, \text{a@ex.org})$. The pattern P 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 the pattern P is a pattern P' that results from replacing P_1 by the pattern $P'_1 = (\text{SELECT } ?X\,?Y\,(?X, \text{is}, \text{person}))$. The patterns P and P' are equivalent because the patterns P_1 and P'_1 return the same multiset of solution mappings $\Omega_1 = \{\!\{\{?X \mapsto a\}\}\!\}$. Note that variable $?Y$ is not in the solutions 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 $\begin{bmatrix} ?X \\ \hline a \end{bmatrix}$ and $\begin{bmatrix} ?X & ?Y \\ \hline a & \bot \end{bmatrix}$. 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. [13], a database is defined to allow duplicate facts, and the evaluation of a 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$^\neg$. Furthermore, we follow the work of Bertossi et al. [17] 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, named facts, rules and goals.

A *fact* is a positive literal that 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 its variables occur positively. 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.

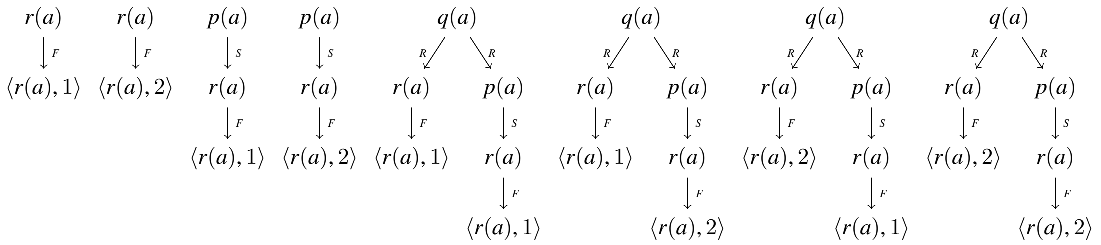
r(a) r(a) p(a) p(a) q(a) q(a) q(a) q(a)
F F S S R R R R R R R R
⟨r(a),1⟩ ⟨r(a),2⟩ r(a) r(a) r(a) p(a) r(a) p(a) r(a) p(a) r(a) p(a)
F F F S F S F S F S
⟨r(a),1⟩ ⟨r(a),2⟩ ⟨r(a),1⟩ r(a) ⟨r(a),1⟩ r(a) ⟨r(a),2⟩ r(a) ⟨r(a),2⟩ r(a)
F F F F
⟨r(a),1⟩ ⟨r(a),2⟩ ⟨r(a),1⟩ ⟨r(a),2⟩

Fig. 3. Example of derivation trees. Let D be a NRMD$^{\neg}$ database, $F = r(a)$ be a fact in D with $\mathrm{card}(F, D) = 2$, $\Pi = \{R, S\}$ be a NRMD$^{\neg}$ program 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.

A *goal clause* is an atom without 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.

4.2. NRMD$^{\neg}$ Semantics

We follow the formalisms by Mumick et al. [13] and Bertossi et al. [17] that use a proof-theoretic semantics for NRMD$^{\neg}$ programs. The semantics is based on the notions of "substitution" and "derivation tree".

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, the answer for 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 obtained from one proof showing that this substitution works.

The notion of "colored set" [13] will be used to identify the different copies of an element. The *colored set* of a multiset M, denoted $\mathrm{coloring}(M)$, is the set $C = \{\langle a, i\rangle \mid a \in \mathrm{set}(M) \text{ and } 1 \leqslant i \leqslant \mathrm{card}(a, M)\}$. Each element $\langle a, i\rangle \in C$ is called a colored copy 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 $\mathrm{coloring}^{-1}(\langle a, i\rangle) = a$.

The notion of "derivation tree" [13] will be used to count the number of proofs for an atom. Formally, a *Derivation Tree* is a connected, undirected graph, with no cycles, represented as a tuple $\mathcal{T} = (\mathcal{N}, \mathcal{E}, \mathcal{L}, \epsilon, \lambda)$ where $\mathcal{N}$ is a set of nodes, $\mathcal{E}$ is a set of edges, $\mathcal{L}$ is a set of labels (for nodes and edges), $\epsilon : \mathcal{E} \to \mathcal{N} \times \mathcal{N}$ is a total function that assigns a pair of nodes to each edge, and $\lambda : (\mathcal{N} \cup \mathcal{E}) \to \mathcal{L}$ is a total function that assigns a label to each node and edge. The function $root(\mathcal{T})$ will be used to obtain the root node of $\mathcal{T}$.

Let R be a rule of the form $L_{n+1} \leftarrow L_1, \ldots, L_m, L_{m+1}, \ldots, L_n$ where $L_1, \ldots, L_m$ are positive literals, and $L_{m+1}, \ldots, L_n$ are negative literals, DT be a set of derivation trees, and $ST = (\mathcal{T}_1, \ldots, \mathcal{T}_m)$ be a sequence of derivation trees that satisfy that every derivation tree in ST is also in DT. We say that DT matches R with ST, denoted $DT \models^{ST} R$, if there is a substitution θ satisfying: (i) for every positive literal $L_i \in R$ it applies that $\theta(L_i) = root(\mathcal{T}_i)$ where $\mathcal{T}_i \in ST$; and (ii) for every negative literal $L_j \in$ it applies that DT does not contain a derivation tree whose root node has the label $\theta(L_j)$.

Assume that $DT \models^{ST} R$ where $ST = (\mathcal{T}_1, \ldots, \mathcal{T}_m)$, $\mathcal{T}_1 = (\mathcal{N}_1, \mathcal{E}_1, \mathcal{L}_1, \epsilon_1, \lambda_1), \ldots,$ and $\mathcal{T}_m = (\mathcal{N}_m, \mathcal{E}_m, \mathcal{L}_m, \epsilon_m, \lambda_m)$. The derivation tree $\mathcal{T}_R = (\mathcal{N}_R, \mathcal{E}_R, \mathcal{L}_R, \epsilon_R, \lambda_R)$ for the rule R is defined as follows: $\mathcal{N}_R = \{n_r\} \cup \mathcal{N}_1 \cup \cdots \cup \mathcal{N}_m$, $\mathcal{E}_R = \{e_1, \ldots, e_m\} \cup \mathcal{E}_1 \cup \cdots \cup \mathcal{E}_m$, $\mathcal{L}_R = \{R, \theta(L_{n+1})\} \cup \mathcal{L}_1 \cup \cdots \cup \mathcal{L}_m$, every assignment in ϵ_i is also in ϵ_R, $\epsilon_R(e_1) = (n_r, root(\mathcal{T}_1)), \ldots, \epsilon_R(e_m) = (n_r, root(\mathcal{T}_m))$, every assignment in λ_i is also in λ_R, $\lambda_R(n_r) = \theta(L_{n+1})$, and $\lambda_R(e_1) = R$.

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

1. For every fact $F \in D$ of the form $p(t_i, \ldots, t_n)$, and for every colored copy $\langle p(t_i, \ldots, t_n), i\rangle$, it applies that $\mathrm{dt}(\Pi, D)$ contains a derivation tree $\mathcal{T}_F^i = (\mathcal{N}_F^i, \mathcal{E}_F^i, \mathcal{L}_F^i, \epsilon_F^i, \lambda_F^i)$ where $\mathcal{N}_F^i = \{n_1, n_2\}$, $\mathcal{E}_F^i = \{e_1\}$, $\mathcal{L}_F^i = \{p(t_i, \ldots, t_n), \langle p(t_i, \ldots, t_n), i\rangle, F\}$, $\epsilon_F^i(e_1) = (n_1, n_2)$, $\lambda_F^i(n_1) = p(t_i, \ldots, t_n)$, $\lambda_F^i(n_2) = \langle p(t_i, \ldots, t_n), i\rangle$, and $\lambda_F^i(e_1) = F$.

2. Assume that DT is the set of derivation trees obtained for the facts in D as defined above. Given a rule R in Π and a sequence of derivation trees ST satisfying $DT \models^{ST} R$, the derivation tree for R is added to DT. This process is repeated for every rule R in Π, until no more derivation trees are generated. Finally, $\mathrm{dt}(\Pi, D) = DT$.

Let Π be a NRMD$^\neg$ program, D be a NRMD$^\neg$ database and F be a fact. A derivation tree $\mathcal{T} \in \mathrm{dt}(\Pi, D)$ is said to be a *proof* for the fact F if the label of the root node is 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 cardinality of F in $\mathrm{atoms}(\Pi, D)$ is the number of proofs of F. Figure 3 shows the derivation trees that are proofs of the facts derived from an example NRMD$^\neg$ program. The facts $r(a)$, $p(a)$ and $q(a)$ belong to $\mathrm{atoms}(\Pi, D)$ with cardinalities 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$^\neg$ query 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$^\neg$ query 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 query answer for 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. [17] 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 translations 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:

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

Next, we 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

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

where $A_1, \dots, A_m$ are positive literals, and $\neg B_1, \dots, \neg B_n$ 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 $A_1, \dots, A_i$. 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 and $R_j^{B'}$ 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) $R_j^B = r_j^B(\bar{Y}_m) \leftarrow r_{j-1}^B(\bar{Y}_m), \neg B_j'(\bar{Y}_m)$,

(c) $R_j^{B'} = B'_j(\bar{Y}_m) \leftarrow r_{j-1}^{B}(\bar{Y}_m), B_j$.

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 solutions of query $(p(\bar{X}), \Pi)$ after the replacement are the same and have the same cardinalities. 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 with attributes $\widehat{R} = \{A_1, \ldots, A_n\}$ 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 relation 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 is defined over the schema R_i. Sometimes we will write MRA database, emphasizing that the multiset relational database is in the context of MRA.

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, $A \in \widehat{E}_1$ and $B \in \mathbf{A}$ 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 the same constant as 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 (of the same schema as E) is defined as a function $\text{Eval}(E, D)$ that returns a multiset relation r with the same schema as E.

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

- 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 $\text{set}(r) = \{t \mid t \in r_1 \text{ and } t \models \psi\}$ and $\text{card}(t, r) = \text{card}(t, r_1)$.
- If E is $\pi_S(E_1)$ then $\text{set}(r) = \{t' \mid t' = t[S] \text{ and } t \in r_1\}$ and $\text{card}(t', r) = \sum_{t \text{ with } t[S]=t'} \text{card}(t, r_1)$.
- 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 $\text{set}(r) = \{t_1 \cup t_2 \mid t_1 \in r_1,\ t_2 \in r_2, \text{ and } t_1 \sim t_2\}$ and $\text{card}(t_1 \cup t_2, r) = \text{card}(t_1, r_1) \times \text{card}(t_2, r_2)$.
- If E is $(E_1 \cup E_2)$ then $\text{set}(r) = \{t \mid t \in r_1 \text{ or } t \in r_2\}$ and $\text{card}(t, r) = \text{card}(t, r_1) + \text{card}(t, r_2)$.
- If E is $(E_1 \setminus E_2)$ then $\text{set}(r) = \{t \mid t \in r_1 \text{ and } t \notin r_2\}$ and $\text{card}(t, r) = \text{card}(t, r_1)$.

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. Equivalence between SPARQL and NRMD$^\neg$

This section presents the simulations that prove that SPARQL and Non-Recursive Multiset Datalog with Safe Negation (NRMD$^\neg$) have the same expressive power. Specifically, we show that SPARQL can be simulated by NRMD$^\neg$ (Section 6.1), and NRMD$^\neg$ can be simulated by SPARQL (Section 6.2).

6.1. From SPARQL to NRMD$^\neg$

This section shows that SPARQL can be simulated by Non-Recursive Multiset Datalog with Safe Negation (NRMD$^\neg$). To support this, we describe the following translation functions:

- function f_{12}, that translates SPARQL queries into NRMD$^\neg$ queries;
- function g_{12}, that translates SPARQL databases into NRMD$^\neg$ databases; and
- function h_{12}, that translates NRMD$^\neg$ query answers into SPARQL query answers.

6.1.1. Translating databases from SPARQL to NRMD$^\neg$

Recall that a SPARQL database is a set of RDF triples and a NRMD$^\neg$ database is a multiset of facts.

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_{12})**.** *Let τ be the vocabulary with predicate names* term, eq, triple, *and* null *with arities $\alpha(\text{term}) = 1$, $\alpha(\text{eq}) = 2$, $\alpha(\text{triple}) = 3$, and $\alpha(\text{null}) = 1$. Given an RDF graph G, the function $g_{12}(G)$ returns a NRMD$^\neg$ database D wich consists of the facts over the vocabulary τ defined according to the following rules:*

- $\text{term}(t) \in D$ *and* $\text{eq}(t,t) \in D$*, for each element t such that there is a triple $(s,p,o) \in G$ with $t = s$, $t = p$, or $t = o$;*
- $\text{triple}(v_1, v_2, v_3) \in D$ *for each triple $(v_1, v_2, v_3) \in G$;*
- $\text{null}(\perp) \in D$*, where $\perp$ is the constant reserved in RDF to encode unbounded values;*
- $\text{comp}(\perp, \perp, \perp) \in D$*;*
- $\text{comp}(a, a, a)$*,* $\text{comp}(a, \perp, a)$*,* $\text{comp}(\perp, a, a)$ *for each term a in D.*

Example 3. *Let G be the RDF graph defined as follows*

$$\begin{aligned}G = \{&(\mathsf{Alice}, \mathsf{livesIn}, \mathsf{Santiago}), (\mathsf{Alice}, \mathsf{knows}, \mathsf{Bob}),\\ &(\mathsf{Bob}, \mathsf{livesIn}, \mathsf{Santiago}), (\mathsf{Bob}, \mathsf{knows}, \mathsf{Carol}),\\ &(\mathsf{Carol}, \mathsf{livesIn}, \mathsf{Lima})\}.\end{aligned}$$

Then the data is translated for Datalog as follows:

$$\begin{aligned}g_{12}(G) = \{\!\!\{\ &\text{term}(\mathsf{Alice}), \text{term}(\mathsf{Bob}), \text{term}(\mathsf{Carol}), \text{term}(\mathsf{Santiago}), \text{term}(\mathsf{livesIn}), \text{term}(\mathsf{knows}),\\ &\text{eq}(\mathsf{Alice}, \mathsf{Alice}), \text{eq}(\mathsf{Bob}, \mathsf{Bob}), \text{eq}(\mathsf{Carol}, \mathsf{Carol}), \text{eq}(\mathsf{Santiago}, \mathsf{Santiago}),\\ &\text{eq}(\mathsf{livesIn}, \mathsf{livesIn}), \text{eq}(\mathsf{knows}, \mathsf{knows}),\\ &\text{triple}(\mathsf{Alice}, \mathsf{livesIn}, \mathsf{Santiago}), \text{triple}(\mathsf{Alice}, \mathsf{knows}, \mathsf{Bob}),\\ &\text{triple}(\mathsf{Bob}, \mathsf{livesIn}, \mathsf{Santiago}), \text{triple}(\mathsf{Bob}, \mathsf{knows}, \mathsf{Carol}),\\ &\text{triple}(\mathsf{Carol}, \mathsf{livesIn}, \mathsf{Lima}),\\ &\text{null}(\perp)\\ &\text{comp}(\perp, \perp, \perp),\\ &\text{comp}(\mathsf{Alice}, \mathsf{Alice}, \mathsf{Alice}), \text{comp}(\mathsf{Alice}, \perp, \mathsf{Alice}), \text{comp}(\perp, \mathsf{Alice}, \mathsf{Alice}),\\ &\text{comp}(\mathsf{livesIn}, \mathsf{livesIn}, \mathsf{livesIn}), \text{comp}(\mathsf{livesIn}, \perp, \mathsf{livesIn}), \text{comp}(\perp, \mathsf{livesIn}, \mathsf{livesIn}),\\ &\quad\vdots\\ &\text{comp}(\mathsf{Lima}, \mathsf{Lima}, \mathsf{Lima}), \text{comp}(\mathsf{Lima}, \perp, \mathsf{Lima}), \text{comp}(\perp, \mathsf{Lima}, \mathsf{Lima})\,\}\!\!\}.\end{aligned}$$

Intuitively, atoms $\text{term}(t)$ list all terms in the graph, atoms $\text{eq}(t,t)$, that each term is equal to itself, and atoms $\text{comp}(a, b, c)$ state the compatibility between a and b. The atoms $\text{triple}(v_1, v_2, v_3)$ encode the triples of the graph, and the atom $\text{null}(\perp)$ states that $\perp$ is the null value.

6.1.2. Translating queries from SPARQL to NRMD$^\neg$

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:

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

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

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

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 cardinality differs. Indeed, the cardinality of μ for the query on the right side is twice the cardinality 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 mapping μ of the pattern P_1, formula φ_1 produces an error. Then, formula $\neg\varphi_1$ also produces an error, and hence μ is not a solution to the query on the left side. On the other hand, since μ is a solution mapping for P_1 but not a solution to the pattern $(P_1 \text{ FILTER } \varphi_1)$, μ is a solution mapping to the query on the right side. Hence, this equivalency is not valid because the queries do not have the same solution mappings.

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 $\mu(\varphi_1 \vee \varphi_2) = \textit{true}$ if and only if $\mu(\varphi_1) = \textit{true}$ or $\mu(\varphi_2) = \textit{true}$, and the cardinality of μ on the left hand side is the sum of the cardinalities of μ in each of the terms of the right hand side. □

Now, consider the following equivalence:

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

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:*

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

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

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

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

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

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

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

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

Example 4. *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 $\mu(\mathrm{Error}(\varphi)) = \text{true}$, 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 solutions of P those solutions μ such that $\mu(\varphi)$ is *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 NRMD$^\neg$ queries. The translation essentially follows the idea presented by Polleres [35], adapted to multisets by Angles and Gutierrez [23], and improved by Hernández [41]. 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 4, transforms a SPARQL graph pattern P into a set of NRMD$^\neg$ 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 [35] 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_{12})**.** *Given a SPARQL query P, the function $f_{12}(P)$ 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 $\delta(P)$.*

Example 5. *Let Q be the following SPARQL query asking for all people, the place where they live, and optionally the people their know:*

$$\begin{array}{l}
(((?person, \textsf{livesIn}, ?somewhere) \;\mathrm{AND}\; (?person, \textsf{knows}, ?somebody)) \\
\;\;\mathrm{UNION} \\
\;((?person, \textsf{livesIn}, ?somewhere) \;\mathrm{EXCEPT} \\
\;\;(\,\mathrm{SELECT}\; ?person\; ?somewhere \\
\;\;\;\mathrm{WHERE}\; ((?person, \textsf{livesIn}, ?somewhere) \;\mathrm{AND}\; (?person, \textsf{knows}, ?somebody))))).
\end{array}$$

Table 4

Definition of function δ which allows to translate a SPARQL graph pattern into a set of NRMD$^{\neg}$ rules. Given a pattern P, $\bar{P}$ returns the variables of P 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)	$p(\bar{P}) \leftarrow triple(x_1, x_2, x_3)$	$\bar{P}$ contains the variables in $\{x_1, x_2, x_3\}$
$(P_1 \text{ AND } P_2)$	$p(\bar{P}) \leftarrow v_1(p_1(\bar{P}_1)), v_2(p_2(\bar{P}_2)), \{\mathrm{comp}(v_1(X), v_2(X), X) \mid X \in \bar{P}_1 \cap \bar{P}_2\}$; $\mathrm{comp}(X,X,X) \leftarrow \mathrm{term}(X)$; $\mathrm{comp}(X,Y,X) \leftarrow \mathrm{term}(X), \mathrm{null}(Y)$; $\mathrm{comp}(X,Y,Y) \leftarrow \mathrm{null}(X), \mathrm{term}(Y)$; $\mathrm{comp}(X,X,X) \leftarrow \mathrm{null}(X)$; $\delta(P_1)$; $\delta(P_2)$	v_1 and v_2 are functions whose domain is $\bar{P}_1 \cap \bar{P}_2$, have disjoint range, and $v_i(L)$ denotes a copy of a literal L where its variables have been renamed according to function v_i.
$(P_1 \text{ UNION } P_2)$	$p(\bar{P}) \leftarrow p_1(\bar{P}_1)$; $p(\bar{P}) \leftarrow p_2(\bar{P}_2)$; $\delta(P_1)$; $\delta(P_2)$	$\bar{P} = \bar{P}_1 = \bar{P}_2$
$(P_1 \text{ EXCEPT } P_2)$	$p(\bar{P}) \leftarrow p_1(\bar{P}_1), \neg p_2(\bar{P}_2)$; $\delta(P_1)$; $\delta(P_2)$	$\bar{P} = \bar{P}_1$
$(P_1 \text{ FILTER } x_1 = x_2)$	$p(\bar{P}) \leftarrow p_1(\bar{P}_1), \mathrm{eq}(x_1, x_2)$; $\delta(P_1)$	$\bar{P} = \bar{P}_1$
$(P_1 \text{ FILTER bound}(?X))$	$p(\bar{P}) \leftarrow p_1(\bar{P}_1), \mathrm{term}(?X)$; $\delta(P_1)$;	$\bar{P} = \bar{P}_1$
$(\text{SELECT } W\ P_1)$	$p(\bar{P}) \leftarrow p_1(\bar{P}_1), \mathrm{null}(x_1), \ldots, \mathrm{null}(x_n)$; $\delta(P_1)$	$\bar{P} = W$, and $x_1, \ldots, x_n$ are the variables that are in W but not in $\mathrm{inScope}(P_1)$.

This SPARQL query is not normalized because both sides of the UNION *operator have different variables. To normalize it we introduce a variable* $?somebody$ *with a* SELECT *clause:*

$$
\begin{array}{l}
(((?person, \textsf{livesIn}, ?somewhere) \text{ AND } (?person, \textsf{knows}, ?somebody)) \\
\quad \text{UNION} \\
\quad (\text{SELECT } ?person\ ?somewhere\ ?somebody \\
\quad\quad \text{WHERE } ((?person, \textsf{livesIn}, ?somewhere) \text{ EXCEPT} \\
\qquad\qquad\qquad (\text{SELECT } ?person\ ?somewhere \\
\qquad\qquad\qquad\quad \text{WHERE } ((?person, \textsf{livesIn}, ?somewhere) \text{ AND } (?person, \textsf{knows}, ?somebody)))))).
\end{array}
$$

Then, the corresponding NRMD$^{\neg}$ *query* $f_{12}(Q)$ *is the query* $(q(X), \Pi)$ *where* Π *is defined as follows:*

$$p_1(X, Y, Z) \leftarrow (X_1, \textsf{livesIn}, Y), \mathrm{triple}(X_2, \textsf{knows}, Z), \mathrm{comp}(X_1, X_2, X)$$

$$p_1(X, Y, Z) \leftarrow p_2(X, Y), \mathrm{null}(Z),$$

$$p_2(X, Y) \leftarrow p_3(X, Y), \neg p_4(X, Y),$$

$$P_3(X, Y) \leftarrow \mathrm{triple}(X, \textsf{livesIn}, Y)$$

$$P_4(X, Y) \leftarrow p_5(X, Y, Z)$$

$$P_5(X, Y, Z) \leftarrow (X_1, \textsf{livesIn}, Y), \mathrm{triple}(X_2, \textsf{knows}, Z), \mathrm{comp}(X_1, X_2, X)\,,$$

where the NRMD$^{\neg}$ *variables* X, Y, *and* Z *correspond to the SPARQL variables* $?person$, $?somewhere$, *and* $?somebody$.

6.1.3. Translating query answers from NRMD$^{\neg}$ *to SPARQL*

Recall that a NRMD$^{\neg}$ query answer is a pair (V, M) (where V is a set of variables and M is a multiset of substitutions) and a SPARQL query answer is a multiset of solution mappings.

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

Definition 8 (Function h_{12})**.** *Given a multiset of Datalog substitutions Θ, the function $h_{12}(\Theta)$ returns a multiset of SPARQL solution mappings defined as*

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

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

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

Proof. We need to show that, using the functions defined above, (f_{12}, g_{12}, h_{12}) is a simulation of SPARQL in NRMD$^\neg$. The proof is in the Claim 4 of the Appendix. □

6.2. From NRMD$^\neg$ to SPARQL

This section shows that Non-Recursive Multiset Datalog with Safe Negation (NRMD$^\neg$) can be simulated by SPARQL. To support this, we describe the following translation functions:

- function f_{21}, that translates NRMD$^\neg$ queries into SPARQL queries;
- function g_{21}, that translates NRMD$^\neg$ databases into SPARQL databases; and
- function h_{21}, that translates SPARQL query answers into NRMD$^\neg$ query answers.

6.2.1. Translating databases from NRMD$^\neg$ to SPARQL

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]. Also 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. This idea is formalized next.

In what follows, we will 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 special IRIs) that we will denote with the same symbol c.

Definition 9 (Function g_{21})**.** *Assume that the NRMD$^\neg$ database D contains n copies of a fact F (namely $p(c_1, \ldots, c_n)$), and the $\mathrm{coloring}(D)$ 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.*

Then the function g_{21} applied to the multiset of NRMD$^\neg$ facts D, returns a set of RDF triples (i.e. an RDF graph) defined as

$$g_{21}(D) = \{(\mathtt{NULL}, \mathtt{NULL}, \mathtt{NULL})\} \bigcup_{\langle F,i\rangle \in \mathrm{coloring}(D)} \{(u_{\langle F,i\rangle}, \alpha_0, p), (u_{\langle F,i\rangle}, \alpha_1, c_1), \ldots, (u_{\langle F,i\rangle}, \alpha_n, c_n)\}.$$

Example 6. *Let D be the following NRMD$^\neg$ database:*

$$D = \{\!\!\{ p(a,b), p(a,b), p(a,c), q(b,d,a), q(b,e,a) \}\!\!\}.$$

In this dataset, the fact $p(a,b)$ contains two copies, so we need to generate the colored copies for this fact, namely $\langle p(a,b), 1\rangle$ and $\langle p(a,b), 2\rangle$. Similarly, for fact $p(a,b)$ we have a single copy, and thus we generate a simple colored

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

copy, $\langle p(a,c),1\rangle$*. Then the data is translated for SPARQL as follows:*

$$\begin{aligned} g_{21}(D) = \{&(\mathtt{NULL},\mathtt{NULL},\mathtt{NULL}),\\ &(u_{\langle p(a,b),1\rangle},\alpha_0,p),(u_{\langle p(a,b),1\rangle},\alpha_1,a),(u_{\langle p(a,b),1\rangle},\alpha_2,b),\\ &(u_{\langle p(a,b),2\rangle},\alpha_0,p),(u_{\langle p(a,b),2\rangle},\alpha_1,a),(u_{\langle p(a,b),2\rangle},\alpha_2,b),\\ &(u_{\langle p(a,c),1\rangle},\alpha_0,p),(u_{\langle p(a,c),1\rangle},\alpha_1,a),(u_{\langle p(a,c),1\rangle},\alpha_2,c),\\ &(u_{\langle p(b,d,a),1\rangle},\alpha_0,q),(u_{\langle p(b,d,a),1\rangle},\alpha_1,b),(u_{\langle p(b,d,a),1\rangle},\alpha_2,d),(u_{\langle p(b,d,a),1\rangle},\alpha_3,a),\\ &(u_{\langle p(b,e,a),1\rangle},\alpha_0,q),(u_{\langle p(b,e,a),1\rangle},\alpha_1,b),(u_{\langle p(b,e,a),1\rangle},\alpha_2,d),(u_{\langle p(b,e,a),1\rangle},\alpha_3,a)\}. \end{aligned}$$

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. *Translating queries from NRMD$^\neg$ to SPARQL*

A notable difference between NRMD$^\neg$ and SPARQL is the way both languages define the scope of variables. In NRMD$^\neg$, all variables in a rule 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 the 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 $L = q(X_1,\dots,X_n)$ and a rule R whose head is $q(Y_1,\dots,Y_n)$, the *renamed rule of* R *with respect to* L, denoted $\mathrm{vr}(R,L)$, is the rule R' that results from R by consistently renaming each variable Y_i as X_i, for $1 \leqslant i \leqslant n$.

Let L be a positive literal $p(X_1,\dots,X_n)$ 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 $\{R_1,\dots,R_n\}$ in Π (i.e., p is intensional), then $\mathrm{gp}(L,\Pi)$ returns:

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

   where the operator $T(R)$ is defined as follows:

   - If R is a projection rule $L \leftarrow L_1$ then $T(R)$ is $(\mathrm{SELECT}\,\overline{X}\ P_1)$ where $\overline{X} = \mathrm{var}(L)$ and $P_1 = \mathrm{gp}(L_1,\Pi)$;
   - If R is a join rule $L \leftarrow L_1, L_2$ then $T(R)$ is $(P_1\ \mathrm{AND}\ P_2)$ where $P_1 = \mathrm{gp}(L_1,\Pi)$ and $P_2 = \mathrm{gp}(L_2,\Pi)$;
   - If R is a negation rule $L \leftarrow L_1, \neg L_2$ then $T(R)$ 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 there is just one rule R_1 then $\mathrm{gp}(L,\Pi)$ can be reduced to $T(R_1)$ (no need to rename variables).

Example 7. *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 queries. Recall that a NRMD$^\neg$ query 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 query answer is a multiset Ω of SPARQL mappings. To conclude the translation, we need to define a function that, given a SPARQL query answer Ω, returns a NRMD$^\neg$ query answer (V, M). The issue is that we cannot compute the set V when the multiset Ω is empty. For example, an empty NRMD$^\neg$ database D is translated as the SPARQL database consisting of the set of triples $\{(\texttt{NULL}, \texttt{NULL}, \texttt{NULL})\}$ (see Subsection 6.2.1). The evaluation of the query $\mathrm{gp}(q(X), \Pi)$ in Example 7 returns an empty multiset of mappings, Ω, where the query 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 query answer Ω does not contain the information needed 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 query answer, we can extend the function gp with a query that introduces the variables of the query. This is done using the additional triple $(\texttt{NULL}, \texttt{NULL}, \texttt{NULL})$ 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 $(\texttt{NULL}, \texttt{NULL}, ?X_1)$ AND $\cdots$ AND $(\texttt{NULL}, \texttt{NULL}, ?X_n)$. The translation of a NRMD$^\neg$ query is then the union of the graph patterns computed by the functions gp and VarQuery.

Definition 10 (Function f_{21}). *Given a NRMD$^\neg$ query $Q = (L, \Pi)$, the function $f_{21}(Q)$ returns a SPARQL graph pattern* $(\mathrm{gp}(L, \Pi)\ \mathrm{UNION}\ \mathrm{VarQuery}(\mathrm{var}(L)))$.

Example 8. *Consider the NRMD$^\neg$ query $(q(X), \Pi)$ in Example 7. Then, $f_{21}((q(X), \Pi))$ is the following SPARQL graph pattern:*

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

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 $f_{21}((q(X), \Pi))$ on the SPARQL database $g_{21}(D) = \{(\texttt{NULL}, \texttt{NULL}, \texttt{NULL})\}$ is the SPARQL query answer $\Omega = \{\{?X \mapsto \texttt{NULL}\}\}$. Intuitively, the mapping $\{?X \mapsto \texttt{NULL}\}$ does not codify a NRMD$^\neg$ substitution, but the variables in the domain of NRMD$^\neg$ substitutions.

6.2.3. *Translating query answers from SPARQL to NRMD$^\neg$*

Recall that a SPARQL query answer is a multiset of solution mappings, and a NRMD$^\neg$ query answer is a pair (V, M) (where V is a set of variables and M is a multiset of substitutions). Since a SPARQL solution mapping can be seen as a NRMD$^\neg$ substitution, the translation from SPARQL mappings to NRMD$^\neg$ substitutions does not require modifications, except for the mapping $\{?X_1 \mapsto \texttt{NULL}, \ldots, ?X_n \mapsto \texttt{NULL}\}$ which is used to codify the solution variables.

Definition 11 (Function h_{21}). *Let Ω be a multiset of SPARQL solution mappings that includes a mapping $\mu_{V \mapsto \texttt{NULL}}$ with cardinality 1 where $\mathrm{dom}(\mu_{V \mapsto \texttt{NULL}}) = V$ and $\mu(?X) = \texttt{NULL}$ for every variable $?X \in \mathrm{dom}(\mu_{V \mapsto \texttt{NULL}})$, and for every mapping $\mu' \in \Omega$ it holds that $\mathrm{dom}(\mu') = V$. The NRMD$^\neg$ solution for Ω, denoted $h_{21}(\Omega)$, 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 $\theta_\mu = \{X_1 \mapsto c_1, \ldots, X_n \mapsto c_n\}$ where the NRMD$^\neg$ variable X_i corresponds to the SPARQL variable $?X_i$.*
2. $\mathrm{set}(M) = \{\theta_\mu \mid \mu \in \Omega \setminus \{\mu_{V \mapsto \texttt{NULL}}\}\}$.
3. $\mathrm{card}(\theta_\mu, M) = \mathrm{card}(\mu, \Omega)$.

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

Proof. This is a long but straightforward induction on Datalog queries using as hypothesis that (f_{21}, g_{21}, h_{21}) 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 among SPARQL and NRMD$^\neg$ stated in this section, we get the following theorem:

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

Proof. The claim is based on the simulation of SPARQL with NRMD$^\neg$ (Lemma 8) and the simulation of NRMD$^\neg$ with MRA (Lemma 9). □

7. Equivalence between MRA and 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. Specifically, we show that MRA can be simulated by NRMD$^\neg$ (Section 7.1), and NRMD$^\neg$ can be simulated by MRA (Section 7.2).

7.1. From MRA to NRMD$^\neg$

This section shows that Multiset Relational Algebra (MRA) can be simulated by Non-Recursive Multiset Datalog with Safe Negation (NRMD$^\neg$). To support this, we describe the following translation functions:

- function f_{32}, that translates MRA queries into NRMD$^\neg$ queries;
- function g_{32}, that translates MRA databases into NRMD$^\neg$ databases; and
- function h_{32}, that translates NRMD$^\neg$ query answers into MRA query answers.

7.1.1. Translating databases from MRA to NRMD$^\neg$

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 multiset 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 over a relation schema R, with $\vec{R} = (A_1, \dots, A_n)$, 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 $p(c_1, \dots, c_n)$ where p is the image of R, every c_i is $t(A_i)$, and the cardinality of f in $\Sigma(r)$ is given by the cardinality 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:*

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

Example 9. *Let D be an MRA dataset consisting in two relations r and s with respective relation schemas R and S with $\vec{R} = (A_1, A_2)$ and $\vec{S} = (A_1, A_3)$, and defined as follows:*

$$r = \{\!\{ \{A_1 \mapsto a_1, A_2 \mapsto a_2\}, \{A_1 \mapsto a_1, A_2 \mapsto a_2\}, \{A_1 \mapsto a_1, A_2 \mapsto a_3\} \}\!\},$$

$$s = \{\!\{ \{A_1 \mapsto a_1, A_3 \mapsto a_4\} \}\!\}.$$

Then, the corresponding NRMD$^\neg$ dataset is the following:

$$g_{32}(D) = \{\!\{ p_R(a_1, a_2), p_R(a_1, a_2), p_R(a_1, a_3), p_S(a_1, a_4), \mathrm{eq}(a_1, a_1), \mathrm{eq}(a_2, a_2), \mathrm{eq}(a_3, a_3), \mathrm{eq}(a_4, a_4) \}\!\},$$

where predicates p_R and p_S are the corresponding images for the relation schemas R and S.

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. Translating queries from MRA to NRMD$^\neg$

Recall that a MRA query is a relational algebra expression, and a NRMD$^\neg$ query is a set of rules.

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 cardinality 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$^\neg$), we omit in how intensional predicate names are assigned.

7.1.3. Translating query answers fro NRMD$^\neg$ to MRA

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

Definition 14 (Function h_{32})**.** *Given a NRMD$^\neg$ query 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 $\operatorname{card}(t, r) = \operatorname{card}(\theta, M)$.*

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

Proof. Let f_{32}, g_{32}, h_{32} be the functions described in Definition 12, Definition 14 and Definition 13 respectively. 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). □

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

7.2. *From NRMD$^\neg$ to MRA*

This section shows that Non-Recursive Multiset Datalog with Safe Negation (NRMD$^\neg$) can be simulated by Multiset Relational Algebra (MRA). To support this, we describe the following translation functions:

- f_{23}, that translates NRMD$^\neg$ queries into MRA queries;
- g_{23}, that translates NRMD$^\neg$ databases into MRA databases; and
- h_{23}, that translates MRA query answers into NRMD$^\neg$ query answers.

7.2.1. *Translating databases from NRMD$^\neg$ to MRA*

Recall that a database in NRMD$^\neg$ is a multiset of facts, and a database in MRA 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.*

Example 10. *Let M be the multiset of NRMD$^\neg$ facts defined as follows:*

$$M = \{\!\{p_1(c_1,c_2), p_1(c_1,c_2), p_1(c_1,c_3), p_2(c_1,c_4)\}\!\},$$

Then, the corresponding MRA dataset $g_{23}(M)$ consists of the following relations r_1 and r_2 with relation schemas R_1 and R_2, and $\vec{R}_1 = (A_1, A_2)$ and $\vec{R}_2 = (B_1, B_2)$:

$$r_1 = \{\!\{\{A_1 \mapsto c_1, A_2 \mapsto c_2\}, \{A_1 \mapsto c_1, A_2 \mapsto c_2\}, \{A_1 \mapsto c_1, A_2 \mapsto c_3\}\}\!\},$$
$$r_2 = \{\!\{\{B_1 \mapsto c_1, B_3 \mapsto c_4\}\}\!\}.$$

7.2.2. Translating queries from NRMD$^\neg$ to MRA

Recall that a NRMD$^\neg$ query is a set of rules, and a MRA query is a relational algebra expression.

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. Translating query answers from MRA to NRMD$^\neg$

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

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 a NRMD$^\neg$ query 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 by 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. MRA and NRMD$^\neg$ have the same expressive power

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

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

Proof. The claim is based on the simulation of MRA with NRMD$^\neg$ (Lemma 11) and the simulation of NRMD$^\neg$ with MRA (Lemma 12). □

8. Equivalence between MRA and SPARQL

This section presents the simulations that prove that Multiset Relational Algebra (MRA) and SPARQL have the same expressive power. Specifically, we show that MRA can be simulated by SPARQL (Section 8.1), and SPARQL can be simulated by MRA (Section 8.2).

8.1. From MRA to SPARQL

This section shows that Multiset Relational Algebra (MRA) can be simulated by SPARQL. To support this, we describe the following translation functions:

- function f_{31}, that translates MRA queries into SPARQL queries;
- function g_{31}, that translates MRA databases into SPARQL databases; and
- function h_{31}, that translates SPARQL query answers into MRA query answers.

8.1.1. Translating databases from MRA to SPARQL

Recall that a MRA database is a set of relations (where each relation is a multiset of tuples), and a SPARQL database is a set of triples.

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

Let r be a multiset relation, t be a tuple in r and $\{t^1, \ldots, t^n\}$ be the set of copies of t where $n = \operatorname{card}(t, r)$. The function $\beta(t, r)$ returns a set of RDF triples defined as follows: for each copy t^i of t, $\beta(t, r)$ contains a triple (iri_t^i, iri_b, iri_r) where iri_t^i is an IRI which identifies the tuple t^i, iri_r is an IRI which identifies the relation r, and iri_b is an IRI which describes that iri_t^i is a tuple of iri_r; and, for each attribute A in $\widehat{r}$, $\beta(t, r)$ contains a triple of the form (iri_t^i, iri_A, lit_A) where iri_A is an IRI which identifies the attribute A, and lit_A is a literal equivalent to the value $t(A)$. Hence, for each copy of a tuple t we create a set of RDF triples.

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

- *For each multiset relation r in D, and for each tuple t in r, D' contains the RDF triples returned by $\beta(t, r)$;*
- *D' contains a triple $(\mathtt{NULL}, \mathtt{NULL}, \mathtt{NULL})$ where $\mathtt{NULL}$ is a special IRI. Like in the simulation of NRMD$^{\neg}$ with SPARQL, the simulation of MRA with SPARQL uses this special triple to retrieve the variables that are attributes of the MRA query answer.*

Example 11. *Let D be an MRA dataset consisting in two relations r and s with respective relation schemas R and S with $\vec{R} = (A_1, A_2)$ and $\vec{S} = (A_1, A_3)$, and defined as follows:*

$$
\begin{aligned}
r &= \{\!\!\{ \{A_1 \mapsto a_1, A_2 \mapsto a_2\}, \{A_1 \mapsto a_1, A_2 \mapsto a_2\}, \{A_1 \mapsto a_1, A_2 \mapsto a_3\} \}\!\!\}, \\
s &= \{\!\!\{ \{A_1 \mapsto a_1, A_3 \mapsto a_4\} \}\!\!\}.
\end{aligned}
$$

Then, the corresponding SPARQL dataset is the following:

$$
\begin{aligned}
g_{31} = \{ & (u_1^1, iri\_b, iri\_r), (u_1^1, iri\_A_1, lit\_A_{a_1}), (u_1^1, iri\_A_2, lit\_A_{a_2}), \\
& (u_1^2, iri\_b, iri\_r), (u_1^2, iri\_A_1, lit\_A_{a_1}), (u_1^2, iri\_A_2, lit\_A_{a_2}), \\
& (u_2^1, iri\_b, iri\_r), (u_2^1, iri\_A_1, lit\_A_{a_1}), (u_2^1, iri\_A_2, lit\_A_{a_3}), \\
& (v_1^1, iri\_b, iri\_s), (v_1^1, iri\_A_1, lit\_A_{a_1}), (v_1^1, iri\_A_3, lit\_A_{a_4}), \\
& (\mathtt{NULL}, \mathtt{NULL}, \mathtt{NULL}) \},
\end{aligned}
$$

where u_1^1, u_1^2, and u_2^1 correspond to the IRIs for the tuples in the multiset relation r and v_1^1 correspond to the IRI of the tuple in the multiset relation s.

8.1.2. Translating queries from MRA to SPARQL

Recall that a MRA query is a relational algebra expression and a SPARQL query is a graph pattern.

First, consider the following issue. A query answer in MRA is a multiset relation r over a set of attributes $\widehat{r}$, whereas a query answer in SPARQL does 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 $(\mathtt{NULL}, \mathtt{NULL}, ?X_1)\ \mathrm{AND}\ \cdots\ \mathrm{AND}\ (\mathtt{NULL}, \mathtt{NULL}, ?X_n)$, where, for $1 \leqslant i \leqslant n$, variable $?X_i$ is the corresponding SPARQL variable for the MRA attribute X_i.

Example 12. *Consider the MRA expression $r \bowtie s$ where $\widehat{r} = \{X, Y\}$ and $\widehat{s} = \{Y, Z\}$. Then,* $\mathrm{AttrQuery}(E) = (\mathtt{NULL}, \mathtt{NULL}, ?X)\ \mathrm{AND}\ (\mathtt{NULL}, \mathtt{NULL}, ?Y)\ \mathrm{AND}\ (\mathtt{NULL}, \mathtt{NULL}, ?Z)$*, 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* $(\Upsilon(E)\ \mathrm{UNION}\ \mathrm{AttrQuery(E)})$.

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 $?X_1 \cdots ?X_n$ $((?Y, iri\_b, iri\_r)$ AND $((?Y, \mathrm{iri}\_A_1, ?X_1)$ AND $(\cdots$ AND $(?Y, \mathrm{iri}\_A_2, ?X_n) \cdots)$	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 $\widehat{R}$.
$(E_1 \bowtie E_2)$	$(P_1$ 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.

Example 13. *Consider the MRA expression $E = R \bowtie S$ where $\widehat{R} = \{A, B\}$ and $\widehat{S} = \{B, C\}$. Then the corresponding SPARQL query $f_{31}(E)$ is the following:*

$$\begin{aligned} f_{31}(E) = (((&\mathrm{SELECT}\ \{?X_A, ?X_B\}\ \mathrm{WHERE}\ (?Y_1, \mathrm{iri}\_A, ?X_A)\ \mathrm{AND}(?Y_1, \mathrm{iri}\_B, ?X_B))\ \ \mathrm{AND} \\ &(\mathrm{SELECT}\ \{?X_B, ?X_C\}\ \mathrm{WHERE}\ (?Y_1, \mathrm{iri}\_B, ?X_B)\ \mathrm{AND}(?Y_1, \mathrm{iri}\_C, ?X_C)))\ \ \mathrm{UNION} \\ &((\mathtt{NULL}, \mathtt{NULL}, ?X_A)\ \mathrm{AND}\ (\mathtt{NULL}, \mathtt{NULL}, ?X_A)\ \mathrm{AND}\ (\mathtt{NULL}, \mathtt{NULL}, ?X_A))). \end{aligned}$$

If $R = \{\!\{\{A \mapsto a, B \mapsto b\}, \{A \mapsto a, B \mapsto b\}\}\!\}$ and $S = \{\!\{\{B \mapsto b, C \mapsto c\}\}\!\}$, the answer to the SPARQL query over the corresponding translation of the MRA database D to an RDF graph is the following multiset:

$$\begin{aligned} [\![f_{31} E]\!]_{g_{31}(D)} = \{\!\{ &\{?X_A \mapsto a, ?X_B \mapsto b, ?X_C \mapsto c\}, \\ &\{?X_A \mapsto a, ?X_B \mapsto b, ?X_C \mapsto c\}, \\ &\{?X_A \mapsto \mathtt{NULL}, ?X_B \mapsto \mathtt{NULL}, ?X_C \mapsto \mathtt{NULL}\}\}\!\}. \end{aligned}$$

Otherwise, if R is empty then:

$$[\![f_{31}E]\!]_{g_{31}(D)} = \{\!\{\{?X_A \mapsto \texttt{NULL}, ?X_B \mapsto \texttt{NULL}, ?X_C \mapsto \texttt{NULL}\}\}\!\}.$$

Whereas the first two SPARQL mappings $\{?X_A \mapsto a, ?X_B \mapsto b, ?X_C \mapsto c\}$ correspond are duplicates of the same MRA answer, $\{A \mapsto a, B \mapsto b, C \mapsto c\}$, the last mapping does not correspond to an answer, but encodes the attributes of the MRA query. By encoding the attributes of the MRA query, we can reconstruct the result MRA relation even in the case it is empty.

8.1.3. Translating query answers from SPARQL to MRA

Recall that a query answer in SPARQL is a multiset of mappings, and a query answer in MRA is a multiset relation (i.e. a multiset of tuples). 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 $\text{dom}(\mu) = V$ for every mapping $\mu \in \Omega$, and that includes the mapping $\mu_{V\mapsto\texttt{NULL}}$ with $\text{dom}(\mu_{V\mapsto\texttt{NULL}}) = V$, $\mu_{V\mapsto\texttt{NULL}}(?X) = \texttt{NULL}$ 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 $\text{dom}(\mu) = V$ 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 $?X \in V$.*
- $\text{set}(r) = \{t_\mu \mid \mu \in \text{set}(\Omega) \setminus \{\mu_{V\mapsto\texttt{NULL}}\}\}$.
- $\text{card}(t_\mu, r) = \text{card}(\mu, \Omega)$

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. From SPARQL to MRA

This section shows that SPARQL can be simulated by Multiset Relational Algebra (MRA). To support this, we describe the following translation functions:

- function f_{13} which translates SPARQL queries into MRA queries;
- function g_{13} which translates SPARQL databases into MRA databases; and
- function h_{13} which translates MRA query answers into SPARQL query answers.

The translation presented here is inspired by the one presented by Cyganiak [34]. However, unlike Cyganiak, we do not use null values with the SQL semantics. Instead, we use a special constant, denoted $\bot$, used to codify unbound values.

8.2.1. Translating databases from SPARQL to MRA

Recall that a SPARQL database is a set of RDF triples, and a MRA database is a set of multiset relations. 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.

Definition 21 (Function g_{13}). *Let $\bot$ be a special constant. Given a set of RDF triples G, the function $g_{13}(G)$ returns a multiset relational database D' containing the multiset relations* Trip, Null, *and* Comp *defined as follows:*

1. $\widehat{\text{Trip}} = \{S, P, O\}$, $\text{set}(\text{Trip}) = \{\{S \mapsto s, P \mapsto p, O \mapsto o\} \mid (s, p, o) \in G\}$, *and* $\text{card}(t, \text{Trip}) = 1$ *for every tuple $t \in \text{set}(\text{Trip})$.*
2. $\widehat{\text{Null}} = \{N\}$, $\text{set}(\text{Null}) = \{\{N \mapsto \bot\}\}$, *and* $\text{card}(\{N \mapsto \bot\}, \text{Null}) = 1$.

3. $\widehat{\mathrm{Comp}} = \{A_1, A_2, A\}$, $\mathrm{set}(\mathrm{Comp})$ *includes the tuple* $\{A_1 \mapsto \bot, A_2 \mapsto \bot, A \mapsto \bot\}$ *and all tuples of the form* $\{A_1 \mapsto a, A_2 \mapsto a, A \mapsto a\}$, $\{A_1 \mapsto \bot, A_2 \mapsto a, A \mapsto a\}$, *and* $\{A_1 \mapsto a, A_2 \mapsto \bot, A \mapsto a\}$ *where a is an RDF term in G, and* $\mathrm{card}(t, \mathrm{Comp}) = 1$ *for every tuple* $t \in \mathrm{set}(\mathrm{Comp})$.

Example 14. *Let G be the RDF graph defined as follows*

$$\begin{aligned} G = \{&(\mathsf{Alice}, \mathsf{livesIn}, \mathsf{Santiago}), (\mathsf{Alice}, \mathsf{knows}, \mathsf{Bob}),\\ &(\mathsf{Bob}, \mathsf{livesIn}, \mathsf{Santiago}), (\mathsf{Bob}, \mathsf{knows}, \mathsf{Carol}),\\ &(\mathsf{Carol}, \mathsf{livesIn}, \mathsf{Lima})\}. \end{aligned}$$

Then the data is translated for MRA as the database $g_{13}(G)$ with the multiset relations Trip, Null, *and* Comp *defined as follows:*

$$\begin{aligned} \mathrm{Trip} = \{\!\{&\{S \mapsto \mathsf{Alice}, P \mapsto \mathsf{livesIn}, O \mapsto \mathsf{Santiago}\}, \{S \mapsto \mathsf{Alice}, P \mapsto \mathsf{knows}, O \mapsto \mathsf{Bob}\}\\ &\{S \mapsto \mathsf{Bob}, P \mapsto \mathsf{livesIn}, O \mapsto \mathsf{Santiago}\}, \{S \mapsto \mathsf{Bob}, P \mapsto \mathsf{knows}, O \mapsto \mathsf{Carol}\},\\ &\{S \mapsto \mathsf{Carol}, P \mapsto \mathsf{livesIn}, O \mapsto \mathsf{Lima}\}\}\!\} \end{aligned}$$

$$\mathrm{Null} = \{\!\{\{N \mapsto \bot\}\}\!\}$$

$$\begin{aligned} \mathrm{Comp} = \{\!\{&\{A_1 \mapsto \bot, A_2 \mapsto \bot, A_3 \mapsto \bot\},\\ &\{A_1 \mapsto \mathsf{Alice}, A_2 \mapsto \mathsf{Alice}, A_3 \mapsto \mathsf{Alice}\},\\ &\{A_1 \mapsto \mathsf{Alice}, A_2 \mapsto \bot, A_3 \mapsto \mathsf{Alice}\},\\ &\{A_1 \mapsto \bot, A_2 \mapsto \mathsf{Alice}, A_3 \mapsto \mathsf{Alice}\},\\ &\{A_1 \mapsto \mathsf{livesIn}, A_2 \mapsto \mathsf{livesIn}, A_3 \mapsto \mathsf{livesIn}\},\\ &\{A_1 \mapsto \mathsf{livesIn}, A_2 \mapsto \bot, A_3 \mapsto \mathsf{livesIn}\},\\ &\{A_1 \mapsto \bot, A_2 \mapsto \mathsf{livesIn}, A_3 \mapsto \mathsf{livesIn}\},\\ &\quad\vdots\\ &\{A_1 \mapsto \mathsf{Lima}, A_2 \mapsto \mathsf{Lima}, A_3 \mapsto \mathsf{Lima}\},\\ &\{A_1 \mapsto \mathsf{Lima}, A_2 \mapsto \bot, A_3 \mapsto \mathsf{Lima}\},\\ &\{A_1 \mapsto \bot, A_2 \mapsto \mathsf{Lima}, A_3 \mapsto \mathsf{Lima}\}\}\!\}. \end{aligned}$$

8.2.2. Translating queries from SPARQL to MRA

Recall that a SPARQL query is a graph pattern, and a MRA query is a relational algebra expression. First, we define the function Λ which allows translating an RDF triple pattern into a MRA expression.

Assume that a, b, c are RDF terms, and $?X$, $?Y$, $?Z$ are variables. Recall that Trip is a multiset relation that is obtained from a set of RDF triples, where $\widehat{\mathrm{Trip}} = \{S, P, O\}$ is the schema of Trip. Given a triple pattern T, the function $\Lambda(T)$ returns a MRA expression defined as follows[3]:

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

[3] These rules are based on Cyganiak's translation [34].

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

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 $\text{bound}(X)$ 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 $Q = (P_1 \text{ AND } P_2)$ 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 $\text{inScope}(P_1) \cap \text{inScope}(P_2) = \{?X\}$ 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 $\rho_{A_1/X_1}(\rho_{A_2/X_2}(\text{Comp}))$ 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 $\rho_{A/X}(\rho_{A_1/X_1}(\rho_{A_2/X_2}(\text{Comp})))$ 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 $\text{Comp}(\nu_1, \nu_2, \mathcal{X})$ to denote the join of MRA expressions of the form $\rho_{A/X}(\rho_{A_1/\nu_1(X)}(\rho_{A_2/\nu_2(X)}(\text{Comp})))$, 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 $\text{range}(\nu_1) \cap \text{range}(\nu_2) = \emptyset$. Then, we define the MRA operation $E_1 * E_2$ in terms of existing MRA operators as follows:

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

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 $(\text{SELECT } \mathcal{X}\ P)$ where the set of variables $\mathcal{X}$ include a variable that is not in the scope of P, we need to generate values $\bot$ to fill the tuples returned by the translated query. For example, if $\text{inScope}(P) = \{?X\}$ 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 $\rho_{N/Y}(\text{Null})$. Given a set $\mathcal{Y} = \{Y_1, \ldots, Y_n\}$ of attribute names, we define the MRA expression $\Delta(\mathcal{Y})$ as $\rho_{N/Y_1}(\text{Null}) \bowtie \cdots \bowtie \rho_{N/Y_n}(\text{Null})$.

Next, we present the translation of SPARQL queries to MRA queries.

Definition 22 (Function f_{13}). *The translation rules in Table 7 define the function f_{13} from normalized graph patterns whose filter conditions have no Boolean connectives to MRA queries.*

Table 7

Definition of the function f_{13}, which takes a normalized SPARQL pattern P as input (without logical connectives in filter conditions) and returns an MRA query.

SPARQL pattern P	MRA query $f_{13}(P)$	where...
(s, p, o)	$\Lambda(s, p, o)$	
$(P_1 \text{ AND } P_2)$	$(f_{13}(P_1) * f_{13}(P_2))$	
$(P_1 \text{ UNION } P_2)$	$(f_{13}(P_1) \cup f_{13}(P_2))$	
$(P_1 \text{ EXCEPT } P_2)$	$(f_{13}(P_1) \setminus f_{13}(P_2))$	
$(\text{SELECT inScope}(P)\ P_1)$	$\pi_{\mathcal{A}}(f_{13}(P_1) \bowtie \Delta_{\mathcal{B}})$	$\mathcal{A}$ is the set of attribute names corresponding to the variables in set $\text{inScope}(P)$ and $\mathcal{B}$ is the set of attribute names that correspond to variables that are in set $\text{inScope}(P) \setminus \text{inScope}(P_1)$.
$(P_1 \text{ FILTER } \varphi)$	$\sigma_{\gamma(\varphi)}(f_{13}(P_1))$	

Example 15. *Let Q be the following SPARQL query asking for all people, the place where they live, and optionally the people their know (notice that this query is already normalized as we discussed in Example 5).*

$$
\begin{array}{l}
(((?person, \mathsf{livesIn}, ?somewhere) \text{ AND } (?person, \mathsf{knows}, ?somebody)) \\
\quad \text{UNION} \\
\quad (\,\text{SELECT } ?person\ ?somewhere\ ?somebody \\
\quad\quad \text{WHERE } ((?person, \mathsf{livesIn}, ?somewhere) \text{ EXCEPT} \\
\quad\quad\quad\quad (\,\text{SELECT } ?person\ ?somewhere \\
\quad\quad\quad\quad\quad \text{WHERE } ((?person, \mathsf{livesIn}, ?somewhere) \text{ AND } (?person, \mathsf{knows}, ?somebody)))))).
\end{array}
$$

Then, the corresponding query $f_{13}(Q)$ is the query $(q(X), \Pi)$ where Π is defined as follows:

$$
\begin{array}{l}
(\Lambda(?person, \mathsf{livesIn}, ?somewhere) * \Lambda(?person, \mathsf{knows}, ?somebody)) \cup \\
((\pi_{Person}(\Lambda(?person, \mathsf{livesIn}, ?somewhere)) \setminus \\
\quad \pi_{Person}(\Lambda(?person, \mathsf{livesIn}, ?somewhere) * \Lambda(?person, \mathsf{knows}, ?somebody))) \ \bowtie \\
\quad \rho_{N/Somebody}(\text{Null}))\,,
\end{array}
$$

where the MRA attributes Person and Somebody correspond to the SPARQL variables $?person$ and $?somebody$.

8.2.3. Translating query answers from MRA to SPARQL

Recall that a MRA query answer is a multiset of tuples, and a SPARQL query answer is a multiset of solution mappings. Next, we define the function h_{31} that transforms MRA query answers into NRMD$^{\neg}$ query answers.

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 SPARQL mapping $\mu = \{?X \mapsto a\}$. Recall that we write $?X$ to denote the corresponding SPARQL variable for a MRA attribute X.

Definition 23 (Function h_{31})**.** *Given a MRA tuple t, we write $f_{31}(t)$ 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 $\text{dom}(\mu)$ if $Y \notin \hat{t}$ or $t(Y) = \perp$. Abusing notation, $f_{31}(r)$ is also the function that receives a MRA relation r and returns the multiset Ω of SPARQL mappings where $\text{set}(\Omega) = \{\mu \mid$ there exist $t \in r$ such that $f_{31}(t) = \mu\}$ and the cardinality of mapping $f_{31}(t)$ in Ω is the cardinality of tuple t in r.*

Lemma 14. *SPARQL can be simulated by 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 by 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. The claim is based on the simulation of MRA with SPARQL (Lemma 13) and the simulation of SPARQL with MRA (Lemma 14). □

9. Conclusions

We studied the algebraic and logic structure of the multiset semantics of the core SPARQL patterns, and compared it to the classical and well-studied formalisms of multiset relational algebra and multiset Datalog. Our motivation was to shed light on the underlying theoretical structure of the multiset features of SPARQL that could help improve future designs and implementations. In this regard, the main discoveries of this research are: (1) the core fragment of SPARQL patterns matches precisely the multiset semantics of Datalog as defined by Mumick et al. [13]; and (2) this logical structure corresponds to a simple multiset algebra, namely the Multiset Relational Algebra (MRA). These correspondences, besides showing a nice parallel to the one exhibited by classical set relational algebra and relational calculus, and thus transferring theoretical guarantees from these well-studied formalisms, could help to give new insights on possible optimizations and future extensions of SPARQL.

We think there are a couple of lessons learnt in the investigation of the multiset features of SPARQL. First, contrary to the rather chaotic variety of multiset operators in SQL, it is interesting to observe that the SPARQL design comprises a more coherent body of multiset operators. We suggest that this asset should be considered and curated by designers in order to try to keep this clean design in future extensions of SPARQL. Second, there is a challenging goal for query language designers that work with multisets: existing a diversity of multiset extensions for each of the classical set operators, it is not evident at all from a theoretical perspective how to develop a logically coherent formalism that could integrate all or most of them.

Our study shows that there are fragments that behave coherently, but that operators that do not fit in this schema, when available (not always), have to be accessed in a very ad-hoc manner. Last, but not least, this study shows (and adds evidence of) the complexities and challenges that the introduction of multisets brings to query languages, exemplified here in the case of SPARQL. Much more use cases are needed in order to match the theoretical restrictions and recommendations (e.g. as studied in this paper), and real-life use cases that to the best of our knowledge do not have yet a good systematization.

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*, 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.

[8] L.S. Colby and L. Libkin, Tractable iteration mechanisms for bag languages, in: *International Conferencia on Database Theory (ICDT)*, 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 Int. Conf. on Very Large Data Bases (VLDB)*, 1985.

[12] M. Console, P. Guagliardo and L. Libkin, Fragments of bag relational algebra: Expressiveness and certain answers, *Information Systems* (2022). doi:https://doi.org/10.1016/j.is.2020.101604.

[13] I.S. Mumick, H. Pirahesh and R. Ramakrishnan, The Magic of Duplicates and Aggregates, in: *Proc. of the International Conference on Very Large Data Bases*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990, pp. 264–277.

[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: *International Conference on Database Theory (ICDT)*, Vol. 127, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2019, pp. 16:1–16:19. doi:10.4230/LIPIcs.ICDT.2019.16.

[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: *Proc. 17th ACM SIGPLAN International Symposium on Database Programming Languages (DBPL)*, ACM, New York, NY, USA, 2019, pp. 70–73. 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. Gutiérrez, The Multiset Semantics of SPARQL Patterns, in: *15th International Semantic Web Conference (ISWC)*, Lecture Notes in Computer Science, Vol. 9981, Springer, 2016, pp. 20–36. doi:10.1007/978-3-319-46523-4_2.

[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] J. Pérez, M. Arenas and C. Gutierrez, Semantics of SPARQL, Technical Report, TR/DCC-2006-17, Department of Computer Science, University of Chile, 2006.

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

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

[32] 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.

[33] 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.

[34] R. Cyganiak, A relational algebra for SPARQL, Technical Report, HPL-2005-170, HP Labs, 2005.

[35] A. Polleres, From SPARQL to Rules (and back), in: *Proceedings of the 16th Int. World Wide Web Conference (WWW)*, ACM, 2007, pp. 787–796.

[36] S. Schenk, A SPARQL Semantics Based on Datalog, in: *Annual German Conference on Advances in Artificial Intelligence*, Vol. 4667, 2007, pp. 160–174.

[37] R. Angles and C. Gutiérrez, The Expressive Power of SPARQL, in: *Proc. of the International Semantic Web Conference (ISWC)*, Lecture Notes in Computer Science, Vol. 5318, Springer, 2008, pp. 114–129–. doi:10.1007/978-3-540-88564-1_8.

[38] A. Chebotko, S. Lu and F. Fotouhi, Semantics preserving SPARQL-to-SQL translation, *Data & Knowledge Engineering* **68**(10) (2009).

[39] 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.

[40] 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.

[41] 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.

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

[43] 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.

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

[45] 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).

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 $(P_1 \text{ AND } P_2)$ then $\mathrm{subs}_{?X/?Y}(P) = \mathrm{subs}_{?X/?Y}(P_1) \text{ AND } \mathrm{subs}_{?X/?Y}(P_2)$.*
3. *If P has the form $(P_1 \text{ UNION } P_2)$ then $\mathrm{subs}_{?X/?Y}(P) = \mathrm{subs}_{?X/?Y}(P_1) \text{ UNION } \mathrm{subs}_{?X/?Y}(P_2)$.*
4. *If P has the form $(P_1 \text{ EXCEPT } P_2)$ then $\mathrm{subs}_{?X/?Y}(P) = \mathrm{subs}_{?X/?Y}(P_1) \text{ EXCEPT } \mathrm{subs}_{?X/?Y}(P_2)$.*
5. *If P has the form $(P_1 \text{ FILTER } \varphi)$ then $\mathrm{subs}_{?X/?Y}(P) = (\mathrm{subs}_{?X/?Y}(P_1) \text{ FILTER } \nu_{?X/?Y}(\varphi))$ 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 $(\text{SELECT } W \text{ WHERE } P_1)$ then:*
   (a) *If $?Y \notin \mathrm{inScope}(P_1)$, then $\mathrm{subs}_{?X/?Y}(P) = (\text{SELECT } (W \setminus \{?X\} \cup \{?Y\}) \text{ WHERE } P_1)$.*
   (b) *Otherwise, $\mathrm{subs}_{?X/?Y}(P) = (\text{SELECT } (W \setminus \{?X\} \cup \{?Y\}) \text{ WHERE } \mathrm{subs}_{?Y/Z}(P_1))$, 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.*

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;*

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*

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

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, $C = \{\mathit{false}\}$ and $\mu(\mathrm{Error}(\varphi)) = \mathit{false}$ 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 $\mu(\mathrm{Error}(\varphi)) = \mathit{true}$ 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 $(P\,\mathrm{FILTER}\,\varphi) \equiv P'$, 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:

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

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

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

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 $\psi \in \text{comp}(\varphi)$, ψ and $\text{Error}(\psi)$ are reducible, then the filter conditions φ and $\text{Error}(\varphi)$ are reducible.

1. If φ is $\text{bound}(?X)$ then

$$\{\varphi\} \xrightarrow{\circ} \emptyset,$$

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

2. If φ is $?X = a$ then

$$\{\varphi\} \xrightarrow{\circ} \emptyset,$$

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

3. If φ is $?X = ?Y$ then

$$\{\varphi\} \xrightarrow{\circ} \emptyset,$$

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

4. If φ is $\neg\psi_1$ then

$$\begin{aligned}&\{\varphi\} \xrightarrow{\neg} \{\psi,\ \text{Error}(\psi)\},\\ &\{\text{Error}(\varphi)\} = \{\text{Error}(\psi)\}.\end{aligned}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

Hence, for every filter condition φ, the filter conditions φ and $\text{Error}(\varphi)$ 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) $R_j^B = r_j^B(\bar{Y}_m) \leftarrow r_{j-1}^B(\bar{Y}_m), \neg B'_j(\bar{Y}_m)$,
   (c) $R_j^{B'} = B'_j(\bar{Y}_m) \leftarrow r_{j-1}^B(\bar{Y}_m), B_j$.
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.

1. If $m = 1$ and $n = 0$, rule R is already normalized because it is the projection rule $p(\bar{X}) \leftarrow A_1$.

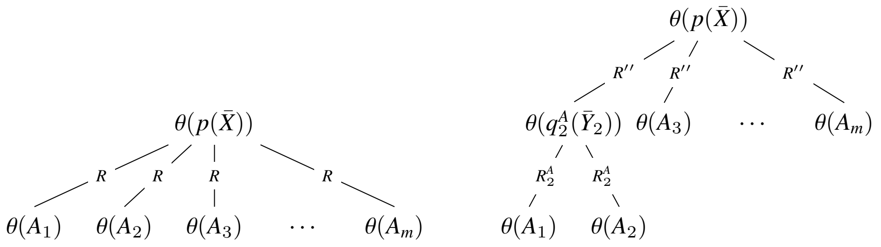

Fig. 4. Derivation trees for the ground literal $\theta(p(\bar{X}))$ regarding query $(p(\bar{X}), \{R\})$ (on the left), and query $(p(\bar{X}), \{R_2^A, R''\})$ (on the right). The children of the nodes labeled with the positive ground literals $\theta(A_i)$ are omitted.

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 cardinality.

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 cardinality of $\theta(p(\bar{X}))$ is, for both queries, the product of the cardinalities 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:

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

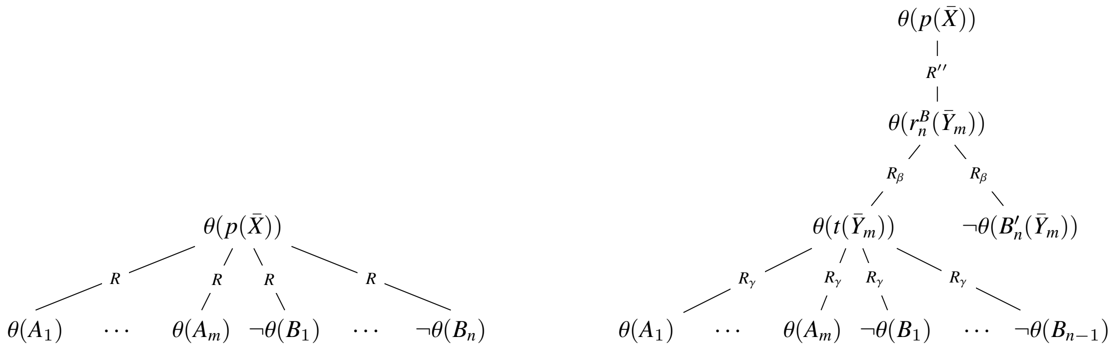
$\theta(p(\bar{X}))$
R
$\theta(A_1)$ $\cdots$ $\theta(A_m)$ $\neg\theta(B_1)$ $\cdots$ $\neg\theta(B_n)$
$\theta(p(\bar{X}))$
R''
$\theta(r_n^B(\bar{Y}_m))$
R_β
$\theta(t(\bar{Y}_m))$ $\neg\theta(B_n'(\bar{Y}_m))$
R_γ
$\theta(A_1)$ $\cdots$ $\theta(A_m)$ $\neg\theta(B_1)$ $\cdots$ $\neg\theta(B_{n-1})$

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.

The rules above are equivalent to the following rules:

$$\begin{aligned}
R_2^A &= q_2^A(\bar{Y}_2) \leftarrow A_1, A_2, \\
R_i^A &= q_i^A(\bar{Y}_i) \leftarrow q_{i-1}^A(\bar{Y}_{i-1}), A_i && \text{for } 2 \leqslant i \leqslant m, \\
R_0^B &= r_0^B(\bar{Y}_m) \leftarrow q_m^A(\bar{Y}_m), \\
R_j^B &= r_j^B(\bar{Y}_m) \leftarrow r_{j-1}^B(\bar{Y}_m), \neg B_j'(\bar{Y}_m) && \text{for } 1 \leqslant j \leqslant n-1, \\
R_j^{B'} &= B_j'(\bar{Y}_m) \leftarrow r_{j-1}^B(\bar{Y}_m), B_j, \\
R_\alpha &= t(\bar{Y}_m) \leftarrow r_{n-1}^B(\bar{Y}_m), \\
R_\beta &= r_n^B(\bar{Y}_m) \leftarrow t(\bar{Y}_m), \neg B_n, \\
R'' &= p(\bar{X}) \leftarrow r_n^B(\bar{Y}_m).
\end{aligned}$$

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 cardinalities.

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 cardinality of $\theta(p(\bar{X}))$ is, for both queries, the product of the cardinalities 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_{12}, g_{12}, h_{21}) *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 $[\![Q]\!]_G = h_{21}([\![f_{12}(Q)]\!]_{g_{12}})$ 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 $f_{12}(Q)$, and μ is the SPARQL mapping $h_{21}(\theta)$. To show then that $[\![Q]\!]_G = h_{21}([\![f_{12}(Q)]\!]_{g_{12}})$, we have to prove that $\mu \in [\![Q]\!]_G$ if and only if $\theta \in h_{21}([\![f_{12}(Q)]\!]_{g_{12}})$, and $\mathrm{card}(\mu, [\![Q]\!]_G) = \mathrm{card}(\theta, h_{21}([\![f_{12}(Q)]\!]_{g_{12}}))$.

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 $f_{12}(Q)$ 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 $h_{21}(\theta) = \{?X \mapsto s,\ ?Y \mapsto o\}$.

   (a) According to the NRMD$^{\neg}$ semantics, $\theta \in [\![f_{12}(Q)]\!]_{g_{12}(G)}$ if and only if $\mathrm{triple}(s, p, o) \in g_{12}(G)$. By the definition function g_{12}, $\mathrm{triple}(s, p, o) \in g_{12}(G)$ 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, $\theta \in [\![f_{12}(Q)]\!]_{g_{12}(G)}$ if and only if $\mu \in [\![Q]\!]_G$.

   (b) By construction, $\mathrm{card}(\theta, [\![f_{12}(Q)]\!]_{g_{12}(G)}) = 1$ and $\mathrm{card}(\mu, [\![Q]\!]_G) = 1$. Hence, $\mathrm{card}(\theta, [\![f_{12}(Q)]\!]_{g_{12}(G)}) = \mathrm{card}(\mu, [\![Q]\!]_G)$.

   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 $f_{12}(Q)$ is then $(q(X, Y, Z), \Pi)$ where Π is the program that consists of the rules in the programs of queries $(p_1(X, Y), \Pi_1) = f_{12}(P_1)$ and $(p_2(X, Z), \Pi_2) = f_{12}(P_2)$, 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)

   $$\begin{aligned} &\mathrm{comp}(X, X, X) \leftarrow \mathrm{term}(X) \\ &\mathrm{comp}(X, Y, X) \leftarrow \mathrm{term}(X), \mathrm{null}(Y) \\ &\mathrm{comp}(Y, X, X) \leftarrow \mathrm{term}(X), \mathrm{null}(Y) \\ &\mathrm{comp}(Y, Y, Y) \leftarrow \mathrm{null}(Y). \end{aligned}$$

   (a) If $\theta \in [\![f_{12}(Q)]\!]_{g_{12}(G)}$ 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

   $$\begin{aligned} &\{X_1/a_1, Y/b\} \in [\![(p_1(X_1, Y), \Pi_1)]\!]_{g_{12}(G)}, \\ &\{X_2/a_2, Z/c\} \in [\![(p_2(X_2, Z), \Pi_2)]\!]_{g_{12}(G)}, \\ &\{X_1/a_1, X_2/a_2, X/a\} \in [\![(\mathrm{comp}(X_1, X_2, X), \Pi)]\!]_{g_{12}(G)}. \end{aligned}$$

   By the induction hypothesis in P_1 and P_2, $\theta_1 \in \{X_1/a_1, Y/b\} \in [\![(p_1(X_1, Y), \Pi_1)]\!]_{g_{12}(G)}$ and $\theta_2 \in \{X_1/a_1, Y/b\} \in [\![(p_2(X_1, Y), \Pi_2)]\!]_{g_{12}(G)}$ if and only if mappings $\mu_1 = h_{21}(\theta_1)$ and $\mu_2 = h_{21}(\theta_2)$ 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, $\theta \in [\![f_{12}(Q)]\!]_{g_{12}(G)}$ if and only if $\mu \in [\![Q]\!]_G$.

(b) By definition,

$$\mathrm{card}(\mu, \llbracket Q \rrbracket_G) = \sum_{\substack{\mu_1 \in \llbracket P_1 \rrbracket_G \\ \mu_2 \in \llbracket P_2 \rrbracket_G \\ \mu_1 \sim \mu_2 \\ \mu = \mu_1 \cup \mu_2}} \mathrm{card}(\mu_1, \llbracket P_1 \rrbracket_G) \times \mathrm{card}(\mu_2, \llbracket P_2 \rrbracket_G).$$

By the induction hypothesis,

$$\mathrm{card}(\mu, \llbracket Q \rrbracket_G) = \sum_{\substack{\{X_1/a_1, Y/b\} \in \llbracket (p_1(X_1,Y), \Pi_1) \rrbracket_{g_{12}(G)} \\ \{X_2/a_2, Z/c\} \in \llbracket (p_2(X_2,Y), \Pi_2) \rrbracket_{g_{12}(G)} \\ \{X_1/a_1, X_2/a_2, X/a\} \in \llbracket (\mathrm{comp}(X_1,X_2,X), \Pi) \rrbracket_{g_{12}(G)}}} \mathrm{card}(\{X_1/a_1, Y/b\}, \llbracket f_{12}(P_1) \rrbracket_{g_{12}(G)}) \times \mathrm{card}(\{X_2/a_2, Z/c\}, \llbracket f_{12}(P_1) \rrbracket_{g_{12}(G)}).$$

By the semantics of NRMD$^\neg$, we conclude that $\mathrm{card}(\mu, \llbracket Q \rrbracket_G) = \mathrm{card}(\theta, \llbracket f_{12}(Q) \rrbracket_{g_{12}(G)})$.

We have shown that we can simulate queries of the form $(P_1 \,\mathrm{AND}\, P_2)$, 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 $(P_1 \,\mathrm{AND}\, P_2)$ are simulable with NRMD$^\neg$.

3. Let Q be a query $(P_1 \,\mathrm{EXCEPT}\, P_2)$, and $?\bar{X}$ be the list of SPARQL variables in set $\mathrm{inScope}(Q)$. The NRMD$^\neg$ query $f_{12}(Q)$ is then $(q(\bar{X}), \Pi)$ where Π is the program that consists of the rules in programs of queries $(p_1(\bar{X}), \Pi_1) = f_{12}(P_1)$ and $(p_2(\bar{X}), \Pi_2) = f_{12}(P_2)$, and the rule $q(\bar{X}) \leftarrow p_1(\bar{X}), \neg p_2(\bar{X})$.

   (a) By the semantics of NRMD$^\neg$, $\theta \in \llbracket f_{12}(Q) \rrbracket_{g_{12}(G)}$ if and only if $\theta \in \llbracket f_{12}(P_1) \rrbracket_{g_{12}(G)}$ and $\theta \notin \llbracket f_{12}(P_2) \rrbracket_{g_{12}(G)}$. By the induction hypothesis, the last condition is equivalent to $\mu \in \llbracket P_1 \rrbracket_G$ and $\mu \notin \llbracket P_2 \rrbracket_G$. By the SPARQL semantics, this is equivalent to $\mu \in \llbracket Q \rrbracket_G$.

   (b) By definition, $\mathrm{card}(\mu, \llbracket Q \rrbracket_G) = \mathrm{card}(\mu, \llbracket P_1 \rrbracket_G)$ and $\mathrm{card}(\theta, \llbracket f_{12}(Q) \rrbracket_{g_{12}(G)}) = \mathrm{card}(\theta, \llbracket f_{12}(P_1) \rrbracket_{g_{12}(G)})$. By the induction hypothesis, $\mathrm{card}(\mu, \llbracket P_1 \rrbracket_G) = \mathrm{card}(\theta, \llbracket f_{12}(P_1) \rrbracket_{g_{12}(G)})$. Hence, $\mathrm{card}(\mu, \llbracket Q \rrbracket_G) = \mathrm{card}(\theta, \llbracket f_{12}(Q) \rrbracket_{g_{12}(G)})$.

   Hence, SPARQL queries of the form $(P_1 \,\mathrm{EXCEPT}\, P_2)$ are simulable with NRMD$^\neg$.

4. Let Q be a SPARQL query $(P_1 \,\mathrm{UNION}\, P_2)$. The NRMD$^\neg$ query $f_{12}(Q)$ 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 $(p_1(\bar{X}), \Pi_1) = f_{12}(P_1)$ and $(p_2(\bar{X}), \Pi_2) = f_{12}(P_2)$, 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 $\theta \in \llbracket (p_1(\bar{X}), \Pi_1) \rrbracket_{g_{12}(G)}$ or $\theta \in \llbracket (p_2(\bar{X}), \Pi_2) \rrbracket_{g_{12}(G)}$. By the induction hypothesis, this is equivalent to that $\mu \in \llbracket P_1 \rrbracket_G$ or $\mu \in \llbracket P_1 \rrbracket_G$. By the SPARQL semantics, this is equivalent to $\mu \in \llbracket Q \rrbracket_G$.

   (b) By the NRMD$^\neg$ semantics, $\mathrm{card}(\theta, \llbracket f_{12}(Q) \rrbracket_{g_{12}(G)}) = \mathrm{card}(\theta, \llbracket f_{12}(P_1) \rrbracket_{g_{12}(G)}) + \mathrm{card}(\theta, \llbracket f_{12}(P_2) \rrbracket_{g_{12}(G)})$ and $\mathrm{card}(\theta, \llbracket f_{12}(Q) \rrbracket_{g_{12}(G)}) = \mathrm{card}(\mu, \llbracket P_1 \rrbracket_G) + \mathrm{card}(\mu, \llbracket P_2 \rrbracket_G)$.
   By the induction hypothesis, $\mathrm{card}(\theta, \llbracket f_{12}(P_1) \rrbracket_{g_{12}(G)}) = \mathrm{card}(\mu, \llbracket P_1 \rrbracket_G)$ and $\mathrm{card}(\theta, \llbracket f_{12}(P_2) \rrbracket_{g_{12}(G)}) = \mathrm{card}(\mu, \llbracket P_2 \rrbracket_G)$. Hence $\mathrm{card}(\theta, \llbracket f_{12}(Q) \rrbracket_{g_{12}(G)}) = \mathrm{card}(\mu, \llbracket Q \rrbracket_G)$.

   Hence, SPARQL queries of the form $(P_1 \,\mathrm{UNION}\, P_2)$ are simulable with NRMD$^\neg$.

5. Let Q be the SPARQL query $(P \,\mathrm{FILTER}\, \varphi)$ 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 $f_{12}(Q)$ 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 $(p(\bar{X}), \Pi') = f_{12}(P)$, 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 $\theta \in [\![(p(\bar{X}), \Pi')]\!]_{g_{12}(G)}$, and $\theta(L_\varphi) \subseteq g_{12}(G)$. By the induction hypothesis, $\theta \in [\![(p(\bar{X}), \Pi')]\!]_{g_{12}(G)}$ is equivalent to $\mu \in [\![P]\!]_G$. By construction, $\theta(L_\varphi) \subseteq \mathrm{atoms}(\Pi', g_{12}(G))$ 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 $g_{12}(G)$. For each fact in $F \in \theta(L_\varphi)$ there is then only one proof that $F \in \mathrm{atoms}(\Pi, g_{12}(G))$. Hence, $\mathrm{card}(\theta, [\![f_{12}(Q)]\!]_{g_{12}(G)}) = \mathrm{card}(\theta, [\![f_{12}(P)]\!]_{g_{12}(G)})$. By the induction hypothesis, $\mathrm{card}(\theta, [\![f_{12}(P)]\!]_{g_{12}(G)}) = \mathrm{card}(\mu, [\![P]\!]_G)$. According to the SPARQL semantics, $\mathrm{card}(\mu, [\![P]\!]_G) = \mathrm{card}(\mu, [\![Q]\!]_G)$. Hence, $\mathrm{card}(\theta, [\![f_{12}(Q)]\!]_{g_{12}(G)}) = \mathrm{card}(\mu, [\![Q]\!]_G)$.

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 $f_{12}(Q)$ is then $(q(\bar{X}), \Pi)$, where Π is the program that consists of the rules in the program of query $(p(\bar{Y}, \Pi') = f_{12}(P)$, 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 $\theta' \in [\![(p(\bar{Y}, \Pi')]\!]_{g_{12}(G)}$ such that $\theta(x) = \theta'(x)$ if $x \in \mathrm{inScope}(Q) \cap \mathrm{inScope}(P)$. Let $\mu = h_{21}(\theta)$ and $\mu' = h_{21}(\theta')$. By construction $\mu = \mu'|_{\mathrm{inScope}(Q)}$. By the induction hypothesis, $\mu' \in [\![P]\!]_G$. Hence, $\mu \in [\![Q]\!]_G$.

(b) By construction,

$$\mathrm{card}(\theta, [\![f_{12}(Q)]\!]_{g_{12}(G)}) = \sum_{\substack{\theta'|_{\mathrm{inScope}(Q)} = \theta \\ \theta' \in [\![f_{12}(P)]\!]_{g_{12}(G)}}} \mathrm{card}(\theta', [\![f_{12}(P)]\!]_{g_{12}(G)}).$$

By the induction hypothesis,

$$\mathrm{card}(\theta, [\![f_{e1,2}(Q)]\!]_{g_{12}(G)}) = \sum_{\substack{\theta'|_{\mathrm{inScope}(Q)} = \theta \\ \mu' = h_{21}(\theta') \\ \mu' \in [\![P]\!]_G}} \mathrm{card}(\mu', [\![P]\!]_G).$$

By construction,

$$\mathrm{card}(\theta, [\![f_{e1,2}(Q)]\!]_{g_{12}(G)}) = \sum_{\substack{\mu'|_{\mathrm{inScope}(Q)} = h_{21}(\theta) \\ \mu' \in [\![P]\!]_G}} \mathrm{card}(\mu', [\![P]\!]_G).$$

Hence, $\mathrm{card}(\theta, [\![f_{12}(Q)]\!]_{g_{12}(G)}) = \mathrm{card}(h_{21}(\theta), [\![Q]\!]_G)$.

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

Hence, the triple (f_{12}, g_{12}, h_{21}) is a simulation of SPARQL in NRMD$^\neg$. □

Claim 5 (NRMD$^\neg$ to SPARQL). *The triple (f_{21}, g_{21}, h_{12}) 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 $h_{12}(\theta) \in [\![f_{21}(Q)]\!]_{g_{21}(D)}$ and $\mathrm{card}(\theta, [\![Q]\!]_D) = \mathrm{card}(h_{12}(\theta), [\![f_{21}(Q)]\!]_{g_{21}(D)})$.

1. If q is an extensional predicate, then $f_{21}(Q)$ is the SPARQL query

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

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

   (a) By construction, $\theta \in [\![Q]\!]_D$ if and only if $h_{12}(\theta) \in [\![f_{21}(Q)]\!]_{g_{21}(D)}$. Indeed, each colored fact $\langle q(a_1, \dots, a_n), i\rangle$ in 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) $\text{card}(\theta, [\![Q]\!]_D)$ is the cardinality of $q(a_1, \dots, a_n)$ 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 $g_{21}(D)$. Hence, $\text{card}(\theta, [\![Q]\!]_D) = \text{card}(h_{12}(\theta), [\![f_{21}(Q)]\!]_{g_{21}(D)})$.

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 $f_{21}(Q)$ maps each of these rules to one of the following SPARQL queries:

   - $(\text{SELECT } \bar{X}\ f_{21}((p(\bar{Y}), \Pi)))$,
   - $(f_{21}((p_1(\bar{X}), \Pi)) \text{ AND } f_{21}((p_2(\bar{X}), \Pi)))$,
   - $(f_{21}((p_3(\bar{X}), \Pi)) \text{ EXCEPT } f_{21}((p_4(\bar{X}), \Pi))$.

   If $\{R_1, \dots, R_n\}$ is the set rules in Π with predicate q in the head, then the SPARQL query $f_{21}(Q)$ has the form $(P_1 \text{ UNION} \cdots \text{UNION } P_n)$, 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 $\mu' \in [\![f_{21}((p(\bar{Y}), \Pi))]\!]_{g_{21}(D)}$ such that $h_{12}(\mu') = \theta'$. Let μ be the solution mapping $\mu'|_{\bar{X}}$. By construction, $\mu \in [\![f_{21}(Q)]\!]_{g_{21}(D)}$ and $h_{12}(\mu) = \theta$.
   - 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 $\mu_1 \in [\![f_{21}((p_1(\bar{Y}_1), \Pi))]\!]_{g_{21}(D)}$ and $\mu_2 \in [\![f_{21}((p_2(\bar{Y}_2), \Pi))]\!]_{g_{21}(D)}$ such that $h_{12}(\mu_1) = \theta_1$ and $h_{12}(\mu_2) = \theta_2$. Let μ be the solution mapping $\mu_1 \cup \mu_2$. By construction, $\mu \in [\![f_{21}(Q)]\!]_{g_{21}(D)}$ and $h_{12}(\mu) = \theta$.
   - 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 $\mu \in [\![f_{21}((p_3(\bar{Y}_3), \Pi))]\!]_{g_{21}(D)}$ such that $\mu \notin [\![f_{21}((p_4(\bar{Y}_4), \Pi))]\!]_{g_{21}(D)}$, and $h_{12}(\mu) = \theta$. By construction, $\mu \in [\![f_{21}(Q)]\!]_{g_{21}(D)}$.

   Hence, $\theta \in [\![Q]\!]_D$ if and only if there exists μ such that $f_{12}(\mu) = \theta$ and $\mu \in [\![f_{21}(Q)]\!]_{g_{21}(D)}$.

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

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

By the inductive hypothesis,

$$\begin{aligned}\mathrm{card}(\theta, [\![Q]\!]_D) = & \sum_{\substack{\theta'|_{\bar{X}}=\theta \\ h_{12}(\mu')=\theta'}} \mathrm{card}(\mu', [\![f_{21}((p(\bar{Y}), \Pi))]\!]_{g_{21}(D)}) + \\ & \mathrm{card}(\mu_1, [\![f_{21}((p_1(\bar{Y}_1), \Pi))]\!]_{g_{21}(D)}) \times \mathrm{card}(\mu_2, [\![f_{21}((p_2(\bar{Y}_2), \Pi))]\!]_{g_{21}(D)}) + \\ & \mathrm{card}(\mu, [\![f_{21}((p_3(\bar{Y}_3), \Pi))]\!]_{g_{21}(D)}) \\ = & \ \mathrm{card}(\mu, [\![f_{21}(Q)]\!]_{g_{21}(D)}).\end{aligned}$$

Hence, the triple (f_{21}, g_{21}, h_{12}) is a simulation of NRMD$^\neg$ in SPARQL. □

Claim 6 (MRA to NRMD$^\neg$)**.** *The triple $(f_{32}, g_{32}, h_{2,3})$ 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 $\theta' \in [\![f_{32}(E)]\!]_{g_{32}(D)}$ such that $h_{2,3}(\theta') = t'$.

1. If E is a relation name R then $f_{32}(E)$ 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 $\theta \in [\![f_{32}(E)]\!]_{g_{32}(D)}$, where $h_{2,3}(\theta) = t$. Indeed, $t \in R^I$ if and only if $r(a_1, \dots, a_n) \in g_{32}(D)$ and $t = (a_1, \dots, a_n)$.
   (b) The fact that $\mathrm{card}(t, [\![E]\!]_D) = \mathrm{card}(\theta, [\![f_{32}(E)]\!]_{g_{32}(D)})$ follows by construction; the cardinality of t in the multiset relation corresponding to the relation name R is the same as the cardinality of fact $r(a_1, \dots, a_n)$ in multiset $g_{32}(D)$.

2. If E is a query $E_1 \cup E_2$, then $\widehat{E}_1 = \widehat{E}$ and $\widehat{E}_2 = \widehat{E}$, and $f_{32}(E)$ is a NRMD$^\neg$ query $(q(\widehat{E}), \Pi)$ such that $f_{32}(E_1) = (q_1(\widehat{E}), \Pi)$ and $f_{32}(E_2) = (q_2(\widehat{E}), \Pi)$, 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 $\theta \in [\![f_{32}(E_1)]\!]_{g_{32}(D)}$ or $\theta \in [\![f_{32}(E_2)]\!]_{g_{32}(D)}$. That is, $\theta \in [\![f_{32}(E)]\!]_{g_{32}(D)}$.
   (b) Assume the respective answers t and θ described in (a). By definition,

$$\begin{aligned}&\mathrm{card}(t, [\![E]\!]_D) = \mathrm{card}(t, [\![E_1]\!]_D) + \mathrm{card}(t, [\![E_2]\!]_D), \\ &\mathrm{card}(\theta, [\![f_{32}(E)]\!]_{g_{32}(D)}) = \mathrm{card}(\theta, [\![f_{32}(E_1)]\!]_{g_{32}(D)}) + \mathrm{card}(\theta, [\![f_{32}(E_2)]\!]_{g_{32}(D)}).\end{aligned}$$

   By the inductive hypothesis, these two cardinalities are equal.

3. If E is a query $E_1 \bowtie E_2$ then $\widehat{E}_1 \cup \widehat{E}_2 = \widehat{E}$, $f_{32}(E) = (q(\widehat{E}), \Pi)$, $f_{32}(E_1) = (q_1(\widehat{E}_1), \Pi)$, $f_{32}(E_2) = (q_2(\widehat{E}_2), \Pi)$, 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 $\theta_1 \in [\![f_{32}(E_1)]\!]_{g_{32}(D)}$ and $\theta_2 \in [\![f_{32}(E_2)]\!]_{g_{32}(D)}$ 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, $\theta \in [\![f_{32}(E)]\!]_{g_{32}(D)}$ 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,

$$\begin{aligned}&\mathrm{card}(t, [\![E]\!]_D) = \mathrm{card}(t_1, [\![E_1]\!]_D) \times \mathrm{card}(t_2, [\![E_1]\!]_D), \\ &\mathrm{card}(\theta, [\![f_{32}(E)]\!]_{g_{32}(D)}) = \mathrm{card}(\theta_1, [\![f_{32}(E_1)]\!]_{g_{32}(D)}) \times \mathrm{card}(\theta_2, [\![f_{32}(E_1)]\!]_{g_{32}(D)}).\end{aligned}$$

By the inductive hypothesis, these two cardinalities are equal.

4. If E is a query $E_1 \setminus E_2$ then $\widehat{E}_1 = \widehat{E}$, $\widehat{E}_1 = \widehat{E}$, $f_{32}(E) = (q(\widehat{E}), \Pi)$, $f_{32}(E_1) = (q_1(\widehat{E}), \Pi)$, $f_{32}(E_2) = (q_2(\widehat{E}), \Pi)$, 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 $\theta \in [\![f_{32}(E_1)]\!]_{g_{32}(D)}$, $\theta \notin [\![f_{32}(E_2)]\!]_{g_{32}(D)}$, and $h_{2,3}(\theta) = t$. By construction, $\theta \in [\![f_{32}(E)]\!]_{g_{32}(D)}$.

   (b) Assume the respective answers t and θ described in (a). By definition, $\operatorname{card}(t, [\![E]\!]_D) = \operatorname{card}(t, [\![E_1]\!]_D)$ and $\operatorname{card}(\theta, [\![f_{32}(E)]\!]_{g_{32}(D)}) = \operatorname{card}(\theta, [\![f_{32}(E_1)]\!]_{g_{32}(D)})$. By the induction hypothesis, these two cardinalities are equal.

5. If E is a query $\pi_S(E_1)$ then $\widehat{E} = S$ and $S \subseteq \widehat{E_1}$, and $f_{32}(E)$ is a NRMD$^\neg$ query $(q(\bar{X}), \Pi)$ such that $f_{32}(E_1) = (q_1(\widehat{E}), \Pi)$, 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 $\theta_1 \in [\![f_{32}(E_1)]\!]_{g_{32}(D)}$ 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 $\theta \in [\![f_{32}(E)]\!]_{g_{32}(D)}$ and $h_{2,3}(\theta) = t$.

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

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

$$\operatorname{card}(\theta, [\![f_{32}(E)]\!]_{g_{32}(D)}) = \sum_{\substack{\theta_1 \in [\![f_{32}(E_1)]\!]_{g_{32}(D)} \\ \theta_1|_{\widehat{E}} = \theta}} \operatorname{card}(\theta_1, [\![f_{32}(E_1)]\!]_{g_{32}(D)}).$$

   By the induction hypothesis, these two cardinalities are equal.

6. If E is a query $\rho_{A/B}(E_1)$ then $\widehat{E} = (\widehat{E}_1 \setminus \{A\}) \cup \{B\}$, $f_{32}(E) = (q(\widehat{E}), \Pi)$, and $f_{32}(E_1) = (q(\widehat{E}_1), \Pi)$.

   (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 $\theta_1 \in [\![f_{32}(E_1)]\!]_{g_{32}(D)}$ 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, $\theta \in [\![f_{32}(E)]\!]_{g_{32}(D)}$ if and only if $\theta_1 \in [\![f_{32}(E_1)]\!]_{g_{32}(D)}$ and $h_{2,3}(\theta) = t$.

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

$$\operatorname{card}(t, [\![E]\!]_D) = \operatorname{card}(t_1, [\![E_1]\!]_D),$$
$$\operatorname{card}(\theta, [\![f_{32}(E)]\!]_{g_{32}(D)}) = \operatorname{card}(\theta_1, [\![f_{32}(E_1)]\!]_{g_{32}(D)}).$$

   By the induction hypothesis, these two cardinalities are equal.

7. If E is a query $\sigma_{A=B}(E_1)$ then $\widehat{E} = \widehat{E}_1$, $f_{32}(E) = (q(\widehat{E}), \Pi)$, and $f_{32}(E_1) = (q_1(\widehat{E}_1), \Pi)$ 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 $\theta \in [\![f_{32}(E_1)]\!]_{g_{32}(D)}$ such that $h_{2,3}(\theta) = t$. By construction, $\theta(A) = \theta(B)$. Then, $\theta \in [\![f_{32}(E)]\!]_{g_{32}(D)}$.

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

$$\operatorname{card}(t, [\![E]\!]_D) = \operatorname{card}(t_1, [\![E_1]\!]_D),$$
$$\operatorname{card}(\theta, [\![f_{32}(E)]\!]_{g_{32}(D)}) = \operatorname{card}(\theta_1, [\![f_{32}(E_1)]\!]_{g_{32}(D)}).$$

   By the induction hypothesis, these two cardinalities are equal.

Hence, the triple $(f_{32}, g_{32}, h_{2,3})$ is a simulation of MRA in NRMD$^\neg$. □

Claim 7 (NRMD$^\neg$ to MRA)**.** *The triple $(f_{2,3}, g_{2,3}, h_{32})$ 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 $h_{32}(\theta) \in [\![f_{2,3}(Q)]\!]_{g_{2,3}(D)}$ and $\mathrm{card}(\theta, [\![Q]\!]_D) = \mathrm{card}(h_{32}(\theta), [\![f_{2,3}(Q)]\!]_{g_{2,3}(D)})$.

1. If q is a extensional predicate, then $f_{21}(Q)$ 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, $s = h_{32}(\theta)$. Hence, $\theta \in [\![Q]\!]_D$ if and only if $h_{32}(\theta) \in [\![f_{2,3}(Q)]\!]_{g_{2,3}(D)}$.

   (b) The identity $\mathrm{card}(\theta, [\![Q]\!]_D) = \mathrm{card}(h_{32}(\theta), [\![f_{2,3}Q]\!]_{f_{2,3}(D)})$ follows from the next identities:

$$\begin{aligned}
\mathrm{card}(\theta, [\![Q]\!]_D) &= \mathrm{card}(q(\theta(X_1), \ldots, \theta(X_n)), D) \\
&= \mathrm{card}(t, r) \\
&= \mathrm{card}(s, [\![f_{2,3}(Q)]\!]_{g_{2,3}(D)}) \\
&= \mathrm{card}(h_{32}(\theta), [\![f_{2,3}Q]\!]_{g_{2,3}(D)}).
\end{aligned}$$

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 $h_{32}(t') = \theta'$. Let t be the solution mapping $t'|_{\bar{X}}$. By construction, $t \in [\![f_{2,3}(Q)]\!]_{g_{2,3}(D)}$ and $h_{32}(t) = \theta$.
   - 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 $h_{32}(t_1) = \theta_1$ and $h_{32}(t_2) = \theta_2$. Let t be the MRA solution $t_1 \cup t_2$. By construction, $t \in [\![f_{2,3}(Q)]\!]_{g_{2,3}(D)}$ and $h_{32}(t) = \theta$.

- 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 $h_{32}(t) = \theta$. 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 $f_{12}(\mu) = \theta$ and $\mu \in [\![f_{21}(Q)]\!]_{g_{21}(D)}$.

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

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

By the induction hypothesis,

$$\begin{aligned}\text{card}(\theta, [\![Q]\!]_D) = \sum_{\substack{\theta'|_{\bar{X}} = \theta \\ h_{12}(t') = \theta'}} & \text{card}(t', [\![f_{2,3}((p(\bar{Y}), \Pi))]\!]_{g_{2,3}(D)}) + \\ & \text{card}(t_1, [\![f_{2,3}((p_1(\bar{Y}_1), \Pi))]\!]_{g_{2,3}(D)}) \times \text{card}(t_2, [\![f_{2,3}((p_2(\bar{Y}_2), \Pi))]\!]_{g_{2,3}(D)}) + \\ & \text{card}(t, [\![f_{2,3}((p_3(\bar{Y}_3), \Pi))]\!]_{g_{2,3}(D)}) \\ = {} & \text{card}(t, [\![f_{2,3}(Q)]\!]_{g_{2,3}(D)}).\end{aligned}$$

Hence, the triple $(f_{2,3}, g_{2,3}, h_{32})$ is a simulation of NRMD$^{\neg}$ in MRA. □

Claim 8 (MRA to SPARQL). *The triple (f_{31}, g_{31}, h_{13}) 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 $\mu' \in [\![f_{31}(E)]\!]_{g_{31}(D)}$ such that $h_{13}(\mu') = t'$.

1. If E is a relation name R then $f_{31}(E)$ is the SPARQL query $(\text{SELECT } ?A_1 \cdots ?A_n\ P)$ where P is the basic graph pattern $((?X, u_b, u_r)\,\text{AND}(?X, u_1, ?A_1)\,\text{AND} \cdots \text{AND}(?X, u_n, ?A_n)))$, 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 $h_{13}(\mu) = t$. 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 $g_{31}(D)$. By definition, there exists such an IRI u if and only if there exists a SPARQL mapping $\mu' \in [\![P]\!]_{g_{31}(D)}$ 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, $\mu' \in [\![P]\!]_{g_{31}(D)}$ if and only if $\mu \in [\![f_{31}(Q)]\!]_{g_{31}(D)}$.

   (b) The fact that $\text{card}(t, [\![E]\!]_D) = \text{card}(\mu, [\![f_{31}(E)]\!]_{g_{31}(D)})$ follows by construction; the cardinality 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 $g_{31}(D)$.

2. If E is a query $E_1 \cup E_2$, then $\widehat{E}_1 = \widehat{E}$, $\widehat{E}_2 = \widehat{E}$, and $f_{31}(E)$ is the SPARQL query $(f_{31}(E_1)\,\text{UNION}\,f_{31}(E_2))$.

   (a) Let t be an MRA tuple with $\hat{t} = \widehat{E}$, and μ be an SPARQL mapping such that $h_{13}(\mu) = t$. 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 $\mu \in [\![f_{31}(E_1)]\!]_{g_{31}(D)}$ or $\mu \in [\![f_{31}(E_2)]\!]_{g_{31}(D)}$. By definition, $t \in [\![E]\!]_D$ if and only if $\mu \in [\![f_{31}(E)]\!]_{g_{31}(D)}$.

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

$$\mathrm{card}(t, [\![E]\!]_D) = \mathrm{card}(t, [\![E_1]\!]_D) + \mathrm{card}(t, [\![E_2]\!]_D),$$
$$\mathrm{card}(\mu, [\![f_{31}(E)]\!]_{g_{31}(D)}) = \mathrm{card}(\mu, [\![f_{31}(E_1)]\!]_{g_{31}(D)}) + \mathrm{card}(\mu, [\![f_{31}(E_2)]\!]_{g_{31}(D)}).$$

By the induction hypothesis, these two cardinalities are equal.

3. If E is a query $E_1 \bowtie E_2$ then $\widehat{E_1} \cup \widehat{E_2} = \widehat{E}$, and $f_{31}(E)$ is the SPARQL query $(f_{31}(E_1)\ \mathrm{AND}\ f_{31}(E_2))$.

   (a) Let t be an MRA tuple with $\hat{t} = \widehat{E}$, and μ be an SPARQL mapping such that $h_{13}(\mu) = t$. 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 $h_{13}(\mu_1) = t_1$, $h_{13}(\mu_2) = t_2$, $\mu_1 \in [\![f_{31}(E_1)]\!]_{g_{31}(D)}$, and $\mu_1 \in [\![f_{31}(E_1)]\!]_{g_{31}(D)}$. By construction, $\mu_1 \sim \mu_2$, $\mu_1 \cup \mu_2 = \mu$, and $\mu \in [\![f_{31}(E)]\!]_{g_{31}(D)}$. Hence, $t \in [\![E]\!]_D$ if and only if $\mu \in [\![f_{31}(E)]\!]_{g_{31}(D)}$.

   (b) Assume the respective answers t, t_1, t_2, μ, μ_1, and μ_2 described in (a). By definition,

$$\mathrm{card}(t, [\![E]\!]_D) = \mathrm{card}(t_1, [\![E_1]\!]_D) \times \mathrm{card}(t_2, [\![E_1]\!]_D),$$
$$\mathrm{card}(\mu, [\![f_{31}(E)]\!]_{g_{31}(D)}) = \mathrm{card}(\mu_1, [\![f_{31}(E_1)]\!]_{g_{31}(D)}) \times \mathrm{card}(\mu_2, [\![f_{31}(E_1)]\!]_{g_{31}(D)}).$$

   By the induction hypothesis, these two cardinalities 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 $f_{31}(E)$ is the SPARQL query $(f_{31}(E_1)\ \mathrm{EXCEPT}\ f_{31}(E_2))$.

   (a) Let t be an MRA tuple with $\hat{t} = \widehat{E}$, and μ be an SPARQL mapping such that $h_{13}(\mu) = t$. 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 $\mu \in [\![f_{31}(E_1)]\!]_{g_{31}(D)}$ and $\mu \notin [\![f_{31}(E_2)]\!]_{g_{31}(D)}$. Hence, $t \in [\![E]\!]_D$ if and only if $\mu \in [\![f_{31}(E)]\!]_{g_{31}(D)}$.

   (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 $\mathrm{card}(\mu, [\![f_{31}(E)]\!]_{g_{31}(D)}) = \mathrm{card}(\mu, [\![f_{31}(E_1)]\!]_{g_{31}(D)})$. By the induction hypothesis, these two cardinalities are equal.

5. If E is a query $\pi_S(E_1)$ then $\widehat{E} = S$ and $S \subseteq \widehat{E_1}$, and $f_{31}(E)$ is a SPARQL query $(\mathrm{SELECT}\ \ W\ f_{31}(E_1))$ 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 $\mu_1 \in [\![f_{31}(E_1)]\!]_{g_{31}(D)}$ such that $h_{13}(\mu_1) = t_1$. Let μ be $\mu_1|_W$. By construction, $t \in [\![E]\!]_D$ if and only if $\mu \in [\![f_{31}(E)]\!]_{g_{31}(D)}$ and $h_{13}(\mu) = t$.

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

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

$$\mathrm{card}(\mu, [\![f_{31}(E)]\!]_{g_{31}(D)}) = \sum_{\substack{\mu_1 \in [\![f_{31}(E_1)]\!]_{g_{31}(D)} \\ \mu_1|_W = \mu}} \mathrm{card}(\mu_1, [\![f_{31}(E_1)]\!]_{g_{31}(D)}).$$

   By the induction hypothesis, these two cardinalities are equal.

6. If E is a query $\rho_{A/B}(E_1)$ then $\widehat{E} = (\widehat{E_1} \setminus \{A\}) \cup \{B\}$, $f_{31}(E)$ is the SPARQL query that results from consistently renaming variable $?A$ as variable $?B$ in $f_{31}(E_1)$ (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 $\mu_1 \in [\![f_{31}(E_1)]\!]_{g_{31}(D)}$ such that $h_{13}(\mu_1) = t_1$. 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, $h_{13}(\mu) = t$. Hence, $t \in [\![E]\!]_D$ if and only if $\mu \in [\![f_{31}(E)]\!]_{g_{31}(D)}$.

(b) Assume the respective query answers t, t_1, μ, and μ_1 described in (a). By definition,

$$\mathrm{card}(t, [\![E]\!]_D) = \mathrm{card}(t_1, [\![E_1]\!]_D),$$
$$\mathrm{card}(\mu, [\![f_{31}(E)]\!]_{g_{31}(D)}) = \mathrm{card}(\mu_1, [\![f_{31}(E_1)]\!]_{g_{31}(D)}).$$

By the induction hypothesis, these two cardinalities are equal.

7. If E is a query $\sigma_{A=B}(E_1)$ then $\widehat{E} = \widehat{E}_1$, $f_{31}(E) = (q(\widehat{E}), \Pi)$, and $f_{31}(E_1)$ 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 $\mu \in [\![f_{31}(E_1)]\!]_{g_{31}(D)}$ such that $h_{13}(\mu) = t$. By construction, $\mu(A) = \mu(B)$. Then, $t \in [\![E]\!]_D$ if and only if $\mu \in [\![f_{31}(E)]\!]_{g_{31}(D)}$.

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

$$\mathrm{card}(t, [\![E]\!]_D) = \mathrm{card}(t_1, [\![E_1]\!]_D),$$
$$\mathrm{card}(\mu, [\![f_{31}(E)]\!]_{g_{31}(D)}) = \mathrm{card}(\mu_1, [\![f_{31}(E_1)]\!]_{g_{31}(D)}).$$

By the induction hypothesis, these two cardinalities are equal.

Hence, the triple (f_{31}, g_{31}, h_{13}) is a simulation of MRA in NRMD$^{\neg}$. □

Claim 9 (SPARQL to MRA)**.** *The triple* (f_{13}, g_{13}, h_{31}) *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 $[\![Q]\!]_G = h_{31}([\![f_{13}(Q)]\!]_{g_{13}})$. 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 $f_{13}(Q)$, and μ is the SPARQL mapping $h_{31}(t)$. To show that $[\![Q]\!]_G = h_{31}([\![f_{13}(Q)]\!]_{g_{13}})$, we prove that $\mu \in [\![Q]\!]_G$ if and only if $t \in [\![f_{13}(Q)]\!]_D$ and $\mathrm{card}(\mu, [\![Q]\!]_G) = \mathrm{card}(t, [\![f_{13}(Q)]\!]_D)$.

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 $f_{12}(Q)$ 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 $t \in [\![f_{13}(Q)]\!]_D$. Hence, $\mu \in [\![Q]\!]_G$ if and only if $t \in [\![f_{13}(Q)]\!]_D$. 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 $(P_1 \,\mathrm{AND}\, P_2)$. 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:

$$\begin{aligned} f_{13}(Q) &= f_{13}(P_1) * f_{13}(P_2) \\ &= \pi_{X,Y,Z}(\rho_{A_1/X_1}(\rho_{A_2/X_2}(\rho_{A/X}(\mathrm{Comp}))) \bowtie \rho_{X/X_1}(f_{13}(P_1)) \bowtie \rho_{X/X_2}(f_{13}(P_2))). \end{aligned}$$

   (a) If $t \in \llbracket f_{13}(Q) \rrbracket_D$ then, there are MRA tuples $t_1 \in \llbracket f_{13}(P_1) \rrbracket_D$, $t_2 \in \llbracket f_{13}(P_2) \rrbracket_D$, and $t_3 \in \llbracket \mathrm{Comp} \rrbracket_D$ 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 $\mu_1 = h_{31}(t_1)$, $\mu_2 = h_{31}(t_2)$, and $\mu_3 = h_{31}(t_3)$. By the induction hypothesis in P_1 and P_2, $t_1 \in \llbracket f_{13}(P_1) \rrbracket_D$ and $t_2 \in \llbracket f_{13}(P_2) \rrbracket_D$ if and only if $\mu_1 \in \llbracket P_1 \rrbracket_G$ and $\mu_2 \in \llbracket P_2 \rrbracket_G$. 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 $\mu \in \llbracket Q \rrbracket_G$. Hence, $t \in \llbracket f_{13}(Q) \rrbracket_D$ if and only if $\mu \in \llbracket Q \rrbracket_G$.

   (b) By definition,

$$\mathrm{card}(\mu, \llbracket Q \rrbracket_G) = \sum_{\substack{\mu_1 \in \llbracket P_1 \rrbracket_G \\ \mu_2 \in \llbracket P_2 \rrbracket_G \\ \mu_1 \sim \mu_2 \\ \mu = \mu_1 \cup \mu_2}} \mathrm{card}(\mu_1, \llbracket P_1 \rrbracket_G) \times \mathrm{card}(\mu_2, \llbracket P_2 \rrbracket_G),$$

$$\mathrm{card}(t, \llbracket f_{13}(Q) \rrbracket_D) = \sum_{\substack{t_1 \in \llbracket f_{13}(P_1) \rrbracket_D \\ t_2 \in \llbracket f_{13}(P_2) \rrbracket_D \\ t_3 \in \llbracket \mathrm{Comp} \rrbracket_D \\ \varphi(t_1, t_2, t_3)}} \mathrm{card}(t_3, \llbracket \mathrm{Comp} \rrbracket_D) \times \mathrm{card}(t_1, \llbracket P_1 \rrbracket_D) \times \mathrm{card}(t_2, \llbracket P_2 \rrbracket_D),$$

   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, $\mathrm{card}(\mu_1, \llbracket P_1 \rrbracket_G) = \mathrm{card}(t_1, \llbracket f_{13}(P_1) \rrbracket_D)$ and $\mathrm{card}(\mu_2, \llbracket P_2 \rrbracket_G) = \mathrm{card}(t_2, \llbracket f_{13}(P_2) \rrbracket_D)$. By construction, $\mathrm{card}(t_3, \llbracket \mathrm{Comp} \rrbracket_D) = 1$. Hence, $\mathrm{card}(\mu, \llbracket Q \rrbracket_G) = \mathrm{card}(t, \llbracket f_{13}(Q) \rrbracket_D)$.

3. Case Q is a query $(P_1 \,\mathrm{EXCEPT}\, P_2)$. Let $?\bar{X}$ be the list of SPARQL variables in set $\mathrm{inScope}(Q)$. The MRA query $f_{13}(Q)$ is then $f_{13}(P_1) \setminus f_{13}(P_2)$.

   (a) By definition, $t \in \llbracket f_{13}(Q) \rrbracket_D$ if and only if $t \in \llbracket f_{13}(P_1) \rrbracket_D$ and $t \notin \llbracket f_{13}(P_2) \rrbracket_D$. By the induction hypothesis, the last condition is equivalent to $\mu \in \llbracket P_1 \rrbracket_G$ and $\mu \notin \llbracket P_2 \rrbracket_G$. By the SPARQL semantics, $t \in \llbracket f_{13}(Q) \rrbracket_G$ if and only if $\mu \in \llbracket Q \rrbracket_G$.

   (b) By definition, $\mathrm{card}(\mu, \llbracket Q \rrbracket_G) = \mathrm{card}(\mu, \llbracket P_1 \rrbracket_G)$ and $\mathrm{card}(t, \llbracket f_{13}(Q) \rrbracket_D) = \mathrm{card}(t, \llbracket f_{13}(P_1) \rrbracket_D)$. By the induction hypothesis, $\mathrm{card}(\mu, \llbracket P_1 \rrbracket_G) = \mathrm{card}(t, \llbracket f_{13}(P_1) \rrbracket_D)$. Hence, $\mathrm{card}(\mu, \llbracket Q \rrbracket_G) = \mathrm{card}(t, \llbracket f_{13}(Q) \rrbracket_D)$.

4. Case Q is a SPARQL query $(P_1 \,\mathrm{UNION}\, P_2)$. The MRA expression $f_{13}(Q)$ is then $f_{13}(P_1) \cup f_{13}(P_2)$.

   (a) By definition, $t \in \llbracket f_{13}(Q) \rrbracket_G$ if and only if $t \in \llbracket f_{13}(P_1) \rrbracket_D$ or $t \in \llbracket f_{13}(P_2) \rrbracket_D$. By the induction hypothesis, $t \in \llbracket f_{13}(Q) \rrbracket_G$ if and only if $\mu \in \llbracket P_1 \rrbracket_G$ or $\mu \in \llbracket P_1 \rrbracket_G$. By the SPARQL semantics, $t \in \llbracket f_{13}(Q) \rrbracket_G$ if and only if $\mu \in \llbracket Q \rrbracket_G$.

   (b) By definition,

$$\mathrm{card}(t, \llbracket f_{13}(Q) \rrbracket_D) = \mathrm{card}(t, \llbracket f_{13}(P_1) \rrbracket_D) + \mathrm{card}(t, \llbracket f_{13}(P_2) \rrbracket_D),$$

$$\mathrm{card}(\mu, \llbracket Q \rrbracket_G) = \mathrm{card}(\mu, \llbracket P_1 \rrbracket_G) + \mathrm{card}(\mu, \llbracket P_2 \rrbracket_G).$$

By the induction hypothesis, $\mathrm{card}(t, \llbracket f_{13}(P_1) \rrbracket_D) = \mathrm{card}(\mu, \llbracket P_1 \rrbracket_G)$ and $\mathrm{card}(t, \llbracket f_{13}(P_2) \rrbracket_D) = \mathrm{card}(\mu, \llbracket P_2 \rrbracket_G)$. Hence $\mathrm{card}(t, \llbracket f_{13}(Q) \rrbracket_D) = \mathrm{card}(\mu, \llbracket Q \rrbracket_G)$.

5. Case Q is a SPARQL query $(P\,\mathrm{FILTER}\,\varphi)$ 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 $f_{13}(Q)$ is then $\sigma_\psi(f_{13}(P))$ where ψ is the MRA selection condition defined as follows:

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

   (a) By definition, $t \in \llbracket f_{13}(Q) \rrbracket_D$ if and only if $t \in \llbracket f_{13}(P) \rrbracket_D$ 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, $t \in \llbracket f_{13}(P) \rrbracket_D$ if and only if $\mu \in \llbracket P \rrbracket_G$. Hence, $\mu \in \llbracket f_{13}(Q) \rrbracket_D$ if and only if $\mu \in \llbracket Q \rrbracket_G$.

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

$$\mathrm{card}(t, \llbracket f_{13}(Q) \rrbracket_D) = \mathrm{card}(t, \llbracket f_{13}(P) \rrbracket_D),$$
$$\mathrm{card}(\mu, \llbracket Q \rrbracket_G) = \mathrm{card}(\mu, \llbracket P \rrbracket_G).$$

   By the induction hypothesis, $\mathrm{card}(t, \llbracket f_{13}(P) \rrbracket_D) = \mathrm{card}(\mu, \llbracket P \rrbracket_G)$. Hence, $\mathrm{card}(t, \llbracket f_{13}(Q) \rrbracket_D) = \mathrm{card}(\mu, \llbracket Q \rrbracket_G)$.

6. Case Q is a SPARQL query $(\mathrm{SELECT}\ ?\bar{X}\ P)$. The MRA expression $f_{13}(Q)$ is then $\pi_{\bar{X}}(f_{13}(P) \bowtie \Delta_{\bar{Y}})$, 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, $t \in \llbracket f_{13}(Q) \rrbracket_D$ if and only if $t(Y) = \bot$ for every attribute name $Y \in \bar{Y}$ and there exists a solution $t' \in \llbracket f_{13}(P) \rrbracket_D$ such that $t'(A) = t(A)$ for every attribute $A \in \bar{X} \setminus \bar{Y}$. Let $\mu' = h_{31}(t')$. By the induction hypothesis, $t' \in \llbracket f_{13}(P) \rrbracket_D$ if and only if $\mu' \in \llbracket P \rrbracket_G$. By construction $\mu = \mu'|_{?\bar{X}}$. Hence, $t \in \llbracket f_{13}(Q) \rrbracket_D$ if and only if $t \in \llbracket P \rrbracket_G$.

   (b) By construction,

$$\mathrm{card}(t, \llbracket f_{13}(Q) \rrbracket_D) = \sum_{\substack{t'|_{\mathrm{inScope}(Q)} = t \\ t' \in \llbracket f_{13}(P) \rrbracket_D}} \mathrm{card}(t', \llbracket f_{13}(P) \rrbracket_D),$$
$$\mathrm{card}(\mu, \llbracket Q \rrbracket_G) = \sum_{\substack{\mu'|_{\mathrm{inScope}(Q)} = \mu \\ \mu' \in \llbracket P \rrbracket_G}} \mathrm{card}(\mu', \llbracket P \rrbracket_G),$$

   By the induction hypothesis, $\mathrm{card}(t', \llbracket f_{13}(P) \rrbracket_D) = \mathrm{card}(\mu', \llbracket P \rrbracket_G)$. Hence, $\mathrm{card}(t, \llbracket f_{13}(Q) \rrbracket_D) = \mathrm{card}(\mu', \llbracket Q \rrbracket_G)$.

Hence, the triple (f_{13}, g_{13}, h_{31}) is a simulation of SPARQL in MRA. □