# Negation in SPARQL

Renzo Angles[1,3] and Claudio Gutierrez[2,3]

[1] Dept. of Computer Science, Universidad de Talca, Chile
[2] Dept. of Computer Science, Universidad de Chile, Chile
[3] Center for Semantic Web Research

**Abstract.** This paper presents a thorough study of negation in SPARQL. The types of negation supported in SPARQL are identified and their main features discussed. Then, we study the expressive power of the corresponding negation operators. At this point, we identify a core SPARQL algebra which could be used instead of the W3C SPARQL algebra. Finally, we analyze the negation operators in terms of their compliance with elementary axioms of set theory.

## 1 Introduction

The notion of negation has been largely studied in database query languages, mainly due to their implications in aspects of expressive power and computational complexity [4–6, 12, 16]. There have been proposed several types of negation, and it seems difficult to get agreement about a standard one. Such heterogeneity comes from intrinsic properties and semantics of each language, features that determine its ability to support specific type(s) of negation(s).

The case of SPARQL is no exception. SPARQL 1.0 did not have any explicit form of negation in its syntax, but could simulate negation by failure. For the next version, SPARQL 1.1, the incorporation of negation generated a lot of debate. As result, SPARQL 1.1 provides four types of negation: negation of filter constraints, by using the Boolean NOT operator; negation as failure, implemented as the combination of an optional graph pattern and the bound operator; difference of graph patterns, expressed by the MINUS operator; and existential negation of graph patterns, expressed by the NOT-EXISTS operator. The main positive and negative aspects of these types of negation are remarked in the SPARQL specifications, and more detailed discussions are available in the records of the SPARQL working group[4]. The goal of this paper is to conduct a formal study about negation in SPARQL.

First, we formalize the syntax and semantics of the different types of negation supported in SPARQL, and discuss their main features. Then, we study the relationships (in terms of expressive power) among the negation operators, first at the level of the SPARQL algebra, and subsequently at the level of SPARQL graph patterns. Finally, we present a case-by-case analysis of the negation operations with respect to their compliance with elementary axioms of set theory.

[4] `https://www.w3.org/2009/sparql/wiki/Design:Negation`

It would be good to have a simple version of negation operators that conform to a known intuition. This is the other main contribution of this paper. First we introduce the DIFF operator as another way of expressing the negation of graph patterns (DIFF is the SPARQL version of the EXCEPT operator of SQL). The semantics of DIFF is based on a simple difference operator introduced at the level of the SPARQL algebra. With this new difference operator, we show that one can define a core algebra (i.e. projection, selection, join, union and simple difference) which is able to express the W3C SPARQL algebra. We show that this algebra is also able to define the SPARQL operators, thus it can be considered a sort of "core" SPARQL algebra. Additionally, we show that the DIFF operator behaves well regarding its compliance with axioms of set theory.

For the sake of space we avoid extended proofs of our results. We refer the reader to the technical report available at `http://arxiv.org/abs/1603.06053`.

## 2 SPARQL graph patterns

The following definition of SPARQL graph patterns is based on the formalism used in [13], but in agreement with the W3C SPARQL specifications [14, 7].

*RDF graphs.* Assume two disjoint infinite sets $I$ and $L$, called IRIs and literals respectively. An *RDF term* is an element in the set $T = I \cup L$[5]. An *RDF triple* is a tuple $(v_1, v_2, v_3) \in I \times I \times T$ where $v_1$ is the *subject*, $v_2$ the *predicate* and $v_3$ 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. Additionally, assume the existence of an infinite set $V$ of variables disjoint from $T$. We will use $\text{var}(\alpha)$ to denote the set of variables occurring in the structure $\alpha$.

A *solution mapping* (or just *mapping* from now on) is a partial function $\mu : V \to T$ where the domain of $\mu$, $\text{dom}(\mu)$, is the subset of $V$ where $\mu$ is defined. The *empty mapping*, denoted $\mu_0$, is the mapping satisfying that $\text{dom}(\mu_0) = \emptyset$. Given $?X \in V$ and $c \in T$, we use $\mu(?X) = c$ to denote the solution mapping variable $?X$ to term $c$. Similarly, $\mu_{?X \to c}$ denotes a mapping $\mu$ satisfying that $\text{dom}(\mu) = \{?X\}$ and $\mu(?X) = c$. Given a finite set of variables $W \subset V$, the restriction of a mapping $\mu$ to $W$, denoted $\mu_{|W}$, is a mapping $\mu'$ satisfying that $\text{dom}(\mu') = \text{dom}(\mu) \cap W$ and $\mu'(?X) = \mu(?X)$ for every $?X \in \text{dom}(\mu) \cap W$. Two mappings $\mu_1, \mu_2$ are *compatible*, denoted $\mu_1 \sim \mu_2$, when for all $?X \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$ it satisfies that $\mu_1(?X) = \mu_2(?X)$, i.e., when $\mu_1 \cup \mu_2$ is also a mapping. Note that two mappings with disjoint domains are always compatible, and that the empty mapping $\mu_0$ is compatible with any other mapping.

A *selection formula* is defined recursively as follows: (i) If $?X, ?Y \in V$ and $c \in I \cup L$ then $(?X = c)$, $(?X = ?Y)$ and $\text{bound}(?X)$ are atomic selection formulas; (ii) If $F$ and $F'$ are selection formulas then $(F \wedge F')$, $(F \vee F')$ and

[5] In addition to $I$ and $L$, RDF and SPARQL consider a domain of anonymous resources called blank nodes. The occurrence of blank nodes introduces several issues that are not discussed in this paper. Based on the results presented in [8], we avoid the use of blank nodes assuming that their absence does not largely affect our results.

$\neg(F)$ are boolean selection formulas. The evaluation of a selection formula $F$ under a mapping $\mu$, denoted $\mu(F)$, is defined in a three-valued logic with values *true* ($\top$), *false* ($\bot$), and *error* ($\xi$). We say that $\mu$ satisfies $F$ where $\mu(F) = \mathit{true}$. The semantics of $\mu(F)$ is defined as follows:

- If $F$ is $?X = c$ and $?X \in \mathrm{dom}(\mu)$, then $\mu(F) = \mathit{true}$ when $\mu(?X) = c$ and $\mu(F) = \mathit{false}$ otherwise. If $?X \notin \mathrm{dom}(\mu)$ then $\mu(F) = \mathit{error}$.
- If $F$ is $?X = ?Y$ and $?X, ?Y \in \mathrm{dom}(\mu)$, then $\mu(F) = \mathit{true}$ when $\mu(?X) = \mu(?Y)$ and $\mu(F) = \mathit{false}$ otherwise. If either $?X \notin \mathrm{dom}(\mu)$ or $?Y \notin \mathrm{dom}(\mu)$ then $\mu(F) = \mathit{error}$.
- If $F$ is $\mathrm{bound}(?X)$ and $?X \in \mathrm{dom}(\mu)$ then $\mu(F) = \mathit{true}$ else $\mu(F) = \mathit{false}$.
- If $F$ is $(F \wedge F')$ then $\top \wedge \top = \top$, $\top \wedge \bot = \bot$, $\top \wedge \xi = \xi$, $\bot \wedge \top = \bot$, $\bot \wedge \bot = \bot$, $\bot \wedge \xi = \bot$, $\xi \wedge \top = \xi$, $\xi \wedge \bot = \bot$, $\xi \wedge \xi = \xi$.
- If $F$ is $(F \vee F')$ then $\top \vee \top = \top$, $\top \vee \bot = \top$, $\top \vee \xi = \top$, $\bot \vee \top = \top$, $\bot \vee \bot = \bot$, $\bot \vee \xi = \xi$, $\xi \vee \top = \top$, $\xi \vee \bot = \xi$, $\xi \vee \xi = \xi$.
- If $F$ is $\neg(F)$ then $\neg\top = \bot$, $\neg\bot = \top$, $\neg\xi = \xi$.

A *multiset* (or *bag*) of solution mappings is an unordered collection in which each solution mapping may appear more than once. A multiset will be represented as a set of solution mappings, each one annotated with a positive integer which defines its multiplicity (i.e. its cardinality). We use the symbol $\Omega$ to denote a multiset and $\mathrm{card}(\mu, \Omega)$ to denote the cardinality of the mapping $\mu$ in the multiset $\Omega$. In this sense, it applies that $\mathrm{card}(\mu, \Omega) = 0$ when $\mu \notin \Omega$. We use $\Omega_0$ to denote the multiset $\{\mu_0\}$ such that $\mathrm{card}(\mu_0, \Omega_0) > 0$ ($\Omega_0$ is called the join identity). The domain of a solution mapping $\Omega$ is defined as $\mathrm{dom}(\Omega) = \bigcup_{\mu \in \Omega} \mathrm{dom}(\mu)$.

*W3C SPARQL algebra.* Let $\Omega_1, \Omega_2$ be multisets of mappings, $W$ be a set of variables and $F$ be a selection formula. The *W3C SPARQL algebra for multisets of mappings* is composed of the operations of projection, selection, join, difference, left-join, union and minus, defined respectively as follows:

- $\pi_W(\Omega_1) = \{\mu' \mid \mu \in \Omega_1, \mu' = \mu_{|W}\}$
  where $\mathrm{card}(\mu', \pi_W(\Omega_1)) = \sum_{\mu' = \mu_{|W}} \mathrm{card}(\mu, \Omega_1)$
- $\sigma_F(\Omega_1) = \{\mu \in \Omega_1 \mid \mu(F) = \mathit{true}\}$
  where $\mathrm{card}(\mu, \sigma_F(\Omega_1)) = \mathrm{card}(\mu, \Omega_1)$
- $\Omega_1 \bowtie \Omega_2 = \{\mu = (\mu_1 \cup \mu_2) \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2, \mu_1 \sim \mu_2\}$
  where $\mathrm{card}(\mu, \Omega_1 \bowtie \Omega_2) = \sum_{\mu = (\mu_1 \cup \mu_2)} \mathrm{card}(\mu_1, \Omega_1) \times \mathrm{card}(\mu_2, \Omega_2)$
- $\Omega_1 \setminus_F \Omega_2 = \{\mu_1 \in \Omega_1 \mid \forall \mu_2 \in \Omega_2, (\mu_1 \nsim \mu_2) \vee (\mu_1 \sim \mu_2 \wedge (\mu_1 \cup \mu_2)(F) \neq \mathit{true})\}$
  where $\mathrm{card}(\mu_1, \Omega_1 \setminus_F \Omega_2) = \mathrm{card}(\mu_1, \Omega_1)$
- $\Omega_1 \cup \Omega_2 = \{\mu \mid \mu \in \Omega_1 \vee \mu \in \Omega_2\}$
  where $\mathrm{card}(\mu, \Omega_1 \cup \Omega_2) = \mathrm{card}(\mu, \Omega_1) + \mathrm{card}(\mu, \Omega_2)$
- $\Omega_1 - \Omega_2 = \{\mu_1 \in \Omega_1 \mid \forall \mu_2 \in \Omega_2, \mu_1 \nsim \mu_2 \vee \mathrm{dom}(\mu_1) \cap \mathrm{dom}(\mu_2) = \emptyset\}$
  where $\mathrm{card}(\mu_1, \Omega_1 - \Omega_2) = \mathrm{card}(\mu_1, \Omega_1)$
- $\Omega_1 ⟕_F \Omega_2 = \sigma_F(\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus_F \Omega_2)$
  where $\mathrm{card}(\mu, \Omega_1 ⟕_F \Omega_2) = \mathrm{card}(\mu, \sigma_F(\Omega_1 \bowtie \Omega_2)) + \mathrm{card}(\mu, \Omega_1 \setminus_F \Omega_2)$

*Syntax of graph patterns.* A SPARQL *graph pattern* is defined recursively as follows: A tuple from $(I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)$ is a graph pattern called a *triple pattern.* [6] If $P_1$ and $P_2$ are graph patterns then $(P_1 \,\text{AND}\, P_2)$, $(P_1 \,\text{UNION}\, P_2)$, $(P_1 \,\text{OPT}\, P_2)$, $(P_1 \,\text{MINUS}\, P_2)$ and $(P_1 \,\text{NOT-EXISTS}\, P_2)$ are graph patterns. If $P_1$ is a graph pattern and $C$ is a filter constraint (as defined below) then $(P_1 \,\text{FILTER}\, C)$ is a graph pattern.

A *filter constraint* is defined recursively as follows: (i) If $?X, ?Y \in V$ and $c \in I \cup L$ then $(?X = c)$, $(?X = ?Y)$ and bound$(?X)$ are *atomic filter constraints*; (ii) If $C_1$ and $C_2$ are filter constraints then $(!C_1)$, $(C_1 \mid\mid C_2)$ and $(C_1 \,\&\&\, C_2)$ are *complex filter constraints.* Given a filter constraint $C$, we denote by $f(C)$ the selection formula obtained from $C$. Note that there exists a simple and direct translation from filter constraints to selection formulas and viceversa.

Given a triple pattern $t$ and a mapping $\mu$ such that $\text{var}(t) \subseteq \text{dom}(\mu)$, we denote by $\mu(t)$ the triple obtained by replacing the variables in $t$ according to $\mu$. Overloading the above definition, we denote by $\mu(P)$ the graph pattern obtained by the recursive substitution of variables in every triple pattern and filter constraint occurring in the graph pattern $P$ according to $\mu$.

*Semantics of SPARQL graph patterns.* The evaluation of a SPARQL graph pattern $P$ over an RDF graph $G$ is defined as a function $[\![P]\!]_G$ (or $[\![P]\!]$ where $G$ is clear from the context) which returns a multiset of solution mappings. Let $P_1, P_2, P_3$ be graph patterns and $C$ be a filter constraint. The evaluation of a graph pattern $P$ over a graph $G$ is defined recursively as follows:

1. If $P$ is a triple pattern $t$, then $[\![P]\!]_G = \{\mu \mid \text{dom}(\mu) = \text{var}(t) \wedge \mu(t) \in G\}$ where each mapping $\mu$ has cardinality 1.
2. $[\![(P_1 \,\text{AND}\, P_2)]\!]_G = [\![P_1]\!]_G \bowtie [\![P_2]\!]_G$
3. If $P$ is $(P_1 \,\text{OPT}\, P_2)$ then
   (a) if $P_2$ is $(P_3 \,\text{FILTER}\, C)$ then $[\![P]\!]_G = [\![P_1]\!]_G ⟕_C [\![P_3]\!]_G$
   (b) else $[\![P]\!]_G = [\![P_1]\!]_G ⟕_{(true)} [\![P_2]\!]_G$
4. $[\![(P_1 \,\text{MINUS}\, P_2)]\!]_G = [\![P_1]\!]_G - [\![P_2]\!]_G$
5. $[\![(P_1 \,\text{NOT-EXISTS}\, P_2)]\!]_G = \{\mu \mid \mu \in [\![P_1]\!]_G \wedge [\![\mu(P_2)]\!]_G = \emptyset\}$
6. $[\![(P_1 \,\text{UNION}\, P_2)]\!]_G = [\![P_1]\!]_G \cup [\![P_2]\!]_G$
7. $[\![(P_1 \,\text{FILTER}\, C)]\!]_G = \sigma_{f(C)}([\![P_1]\!]_G)$

## 3 Types of negation in SPARQL

We can distinguish four types of negation in SPARQL: negation of filter constraints, negation as failure, negation by MINUS and negation by NOT-EXISTS. The main features of these types of negation will be discussed in this section.

[6] We assume that any triple pattern contains at least one variable.

**Negation of filter constraints.** The most basic type of negation in SPARQL is the one allowed in filter graph patterns by including constraints of the form $(!C)$. Following the semantics of SPARQL, a graph pattern $(P\,\mathrm{FILTER}(!C))$, returns the mappings $\mu$ in $[\![P]\!]$ such that $\mu$ satisfies the filter constraint $(!C)$, i.e. $\mu$ does not satisfy the constraint $C$. The negation of filter constraints is a feature well established in SPARQL and does not deserve major discussion. In the rest of the paper we concentrate our interest on the negation of graph patterns.

**Negation as failure.** SPARQL 1.0 does not include an operator to express the negation of graph patterns. In an intent of patching this issue, the SPARQL specification remarks that the negation of graph patterns can be implemented as a combination of an optional graph pattern and a filter constraint containing the bound operator (see [14], Sec. 11.4.1). This style of negation, called *negation as failure* in logic programming, can be illustrated as a graph pattern $P$ of the form $((P_1\,\mathrm{OPT}\,P_2)\,\mathrm{FILTER}(!\,\mathrm{bound}(?X)))$ where $?X$ is a variable of $P_2$ not occurring in $P_1$. Note that, the evaluation of $P$ returns the mappings of $[\![(P_1\,\mathrm{OPT}\,P_2)]\!]$ satisfying that variable $?X$ is unbounded, i.e. $?X$ does not match $P_2$. In other words, $P$ returns "the solution mappings of $P_1$ that are not compatible with the solutions mappings of $P_2$". Unfortunately, there are some issues with this approach [3]. A general solution is included in our technical report.

In order to facilitate the study of negation by failure in SPARQL, we introduce the operator DIFF as an explicit way of expressing it. It is very important to note that the DIFF operator is not defined in SPARQL.

**Definition 1 (DIFF).** *Let $P_1$ and $P_2$ be graph patterns. The* DIFF *operator is defined as* $[\![(P_1\,\mathrm{DIFF}\,P_2)]\!] = \{\mu_1 \in [\![P_1]\!] \mid \forall \mu_2 \in [\![P_2]\!], \mu_1 \nsim \mu_2\}$.

**Negation by MINUS.** SPARQL 1.1 introduced the MINUS operator as an explicit way of expressing the negation (or difference) of graph patterns. Note that DIFF and MINUS have similar definitions. The difference is given by the restriction about disjoint mappings included by the MINUS operator. Such restriction, named Antijoin Restriction inside the SPARQL working group[7], was introduced to avoid solutions with vacuously compatible mappings. Such restriction causes different results for DIFF and MINUS. Basically, if $P_1$ and $P_2$ do not have variables in common then $[\![(P_1\,\mathrm{DIFF}\,P_2)]\!] = \emptyset$ whereas $[\![(P_1\,\mathrm{MINUS}\,P_2)]\!] = [\![P_1]\!]$.

Note that both, DIFF and MINUS resemble the EXCEPT operator of SQL[11]. Considering that EXCEPT makes reference to the difference of two relations (tables), we say that DIFF and MINUS express the *difference of two graph patterns*.

**Negation by NOT-EXISTS.** Another type of negation defined in SPARQL 1.1 is given by the NOT-EXISTS operator. The main feature of this type of negation is the possible occurrence of correlation. Given a graph pattern $P = (P_1\,\mathrm{NOT\text{-}EXISTS}\,P_2)$, we will say that $P_1$ and $P_2$ are *correlated* when there

[7] http://lists.w3.org/Archives/Public/public-rdf-dawg/2009JulSep/0030.html

exist variables occurring in both $P_1$ and $P_2$; such variables are called *correlated variables*. In this case, the evaluation of $P$ is attained by replacing variables in $P_2$ with the corresponding values given by the current mapping $\mu$ of $[\![P_1]\!]$, and testing whether the evaluation of the graph pattern $\mu(P_2)$ returns no solutions. This way of evaluating correlated queries is based on the nested iteration method [10] of SQL. The correlation of variables in NOT-EXISTS introduces several issues that have been studied in the context of subqueries in SPARQL [1, 2]. Some of these issues are discussed in Section 4.

## 4 Expressive Power

In this section we study the expressive power of the negation operators defined in the above section, i.e. DIFF, MINUS and NOT-EXISTS.

**The core SPARQL algebra** Let us introduce a "core" algebra for SPARQL which is able to express all the high-level operators of the W3C SPARQL language. Recall that the W3C SPARQL algebra, defined in Section 2, is composed by the operators of *projection*, *selection*, *join*, *union*, *left-join* and *minus*. Our core algebra is based on a new operator called "simple difference".

**Definition 2 (Simple difference).** *The simple difference between two solution mappings, $\Omega_1$ and $\Omega_2$, is defined as $\Omega_1 \setminus \Omega_2 = \{\mu_1 \in \Omega_1 \mid \forall \mu_2 \in \Omega_2, \mu_1 \nsim \mu_2\}$ where $\operatorname{card}(\mu_1, \Omega_1 \setminus \Omega_2) = \operatorname{card}(\mu_1, \Omega_1)$.*

**Definition 3 (Core SPARQL algebra).** *The core SPARQL algebra is composed by the operations of projection, selection, join, union and simple difference.*

Next, we will show that the core SPARQL algebra contains the W3C SPARQL algebra. Specifically, we will show that the operators of *difference*, *left-join* and *minus* (of the W3C SPARQL algebra) can be simulated with the *simple difference* operator (of the core SPARQL algebra).

**Lemma 1.** *The difference operator (of the W3C SPARQL algebra) is expressible in the core SPARQL algebra.*[8]

*Proof.* Let $\theta$ be a function for fresh renaming of variables. Given a set of variables $X$, we use $\Delta(X, \theta)$ to represent the expression

$$\bigwedge_{?x \in X} (?x = \theta(?x) \vee (\neg \operatorname{bound}(?x) \wedge \neg \operatorname{bound}(\theta(?x))).$$

Given a multiset $\Omega$ and a set of variables $X$, the function $\bot_X(\Omega)$ returns a copy of $\Omega$ where all unbound slots have been substituted by a free constant, e.g.

[8] When preparing the camera ready version, we received a notice from E. Kostylev and R. Kontchakov about their paper "On Expressibility of Non-Monotone Operators in SPARQL" presented at KR 2016. They also pointed to some bugs in our proof of Lemma 1 which were fixed in this version. We thank them for this notice.

*null*. Additionally, we define the multisets $\Omega^{\theta} = \sigma_{\Delta(\mathrm{dom}(\Omega),\theta)}(\Omega \bowtie \theta(\Omega))$ and $\Omega^{\perp} = \perp_{\mathrm{dom}(\theta(\Omega))}(\Omega^{\theta})$.

Given two multisets of solution mappings $\Omega_1$ and $\Omega_2$, we have that

$$\Omega_1 \setminus_F \Omega_2 = \pi_{\mathrm{dom}(\Omega_1)}(\Omega_1^{\perp} \setminus (\pi_{\mathrm{dom}(\theta(\Omega_1))}(\sigma_F(\Omega_1^{\perp} \bowtie \Omega_2)))).$$

**Lemma 2.** *The left-join operator (of the W3C SPARQL algebra) is expressible in the core SPARQL algebra.*

*Proof.* By definition, $\Omega_1 ⟕_F \Omega_2 = \sigma_F(\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus_F \Omega_2)$, and by previous lemma $\setminus_F$ can be simulated in core SPARQL algebra.

**Lemma 3.** *The minus operator (of the W3C SPARQL algebra) is expressible in the core SPARQL algebra.*

*Proof.* Let $\theta$ and $\theta'$ be functions for fresh renaming of variables. Following the notations of the proof of Lemma 1, we have that

$$(\Omega_1 - \Omega_2) = \pi_{\mathrm{dom}(\Omega_1)}(\Omega_1^{\perp} \setminus (\perp_{\mathrm{dom}(\theta(\Omega_1))}(\pi_{\mathrm{dom}(\theta(\Omega_1))}(\sigma_F(\Omega_1^{\theta} \bowtie \Omega_2^{\theta'})))))$$

where $F$ is the selection formula

$$\bigvee_{?x \in \mathrm{dom}(\Omega_1)} (\theta(?x) = \theta'(?x)).$$

Based on Lemmas 1, 2 and 3, we can present our main result about the expressive power of the core SPARQL algebra:

**Theorem 1.** *The W3C SPARQL algebra is expressible with the core SPARQL algebra.*

This result implies that the W3C SPARQL algebra could be implemented by a subset of the original operators (projection, selection, join and union), plus the simple difference operator.

**A core fragment with negation** Let us redefine the DIFF operator by using the simple difference operator of the core algebra as $[\![(P_1 \,\mathrm{DIFF}\, P_2)]\!] = [\![P_1]\!] \setminus [\![P_2]\!]$. Assume that SPARQL$^{\text{DIFF}}$ is the language defined (recursively) by graph patterns of the form $(P \,\mathrm{AND}\, P')$, $(P \,\mathrm{UNION}\, P')$, $(P \,\mathrm{DIFF}\, P')$ and $(P \,\mathrm{FILTER}\, C)$.

**Lemma 4.** *The* OPT *operator is expressible in SPARQL$^{\text{DIFF}}$.*

*Proof.* Given a graph pattern $P$ of the form $(P_1 \,\mathrm{OPT}\, P_2)$, we have two cases:
(i) if $P_2$ is $(P_3 \,\mathrm{FILTER}\, C)$ then $[\![P]\!] = [\![P_1]\!] ⟕_C [\![P_3]\!]$;
(ii) else, $[\![P]\!] = [\![P_1]\!] ⟕_{(true)} [\![P_2]\!]$.
For case (i), we can design a graph pattern based on the simulations described in Lemmas 1 and 2. For case (ii), we have that $((P_1 \,\mathrm{OPT}\, P_2)\,\mathrm{FILTER}(true))$ can be rewritten as $((P_1 \,\mathrm{AND}\, P_2)\,\mathrm{UNION}(P_1 \,\mathrm{DIFF}\, P_2))$.

**Lemma 5.** *The* MINUS *operator is expressible in SPARQL$^{\text{DIFF}}$.*

*Proof.* We have that $(P_1 \,\mathrm{MINUS}\, P_2)$ can be rewritten to a DIFF-based graph pattern following the simulation described in the proof of Lemma 3.

**NOT-EXISTS vs DIFF** Intuitively, one can assume that any graph pattern $P = (P_1 \, \text{NOT-EXISTS} \, P_2)$ can be expressed by using $P' = (P_1 \, \text{DIFF} \, P_2)$ or $(P_1 \, \text{EXCEPT} \, P_2)$. This is for example argued by Kaminski et.al [9] (Lemma 3) but the translation given there does not work. Thus it would be interesting to identify a subset of NOT-EXISTS graph patterns that can be expressed by using the DIFF operator. To do this, we will introduce the notion of "safe" and "unsafe" variables. The set of *safe variables* in a graph pattern $P$, denoted $\text{svar}(P)$, is defined recursively as follows: If $P$ is a triple pattern, then $\text{svar}(P) = \text{var}(P)$; If $P$ is $(P_1 \, \text{AND} \, P_2)$ then $\text{svar}(P) = \text{svar}(P_1) \cup \text{svar}(P_2)$; If $P$ is $(P_1 \, \text{UNION} \, P_2)$ or $(P_1 \, \text{OPT} \, P_2)$ then $\text{svar}(P) = \text{svar}(P_1) \cap \text{svar}(P_2)$; If $P$ is $(P_1 \, \text{FILTER} \, C)$, $(P_1 \, \text{MINUS} \, P_2)$, $(P_1 \, \text{NOT-EXISTS} \, P_2)$ or $(P_1 \, \text{DIFF} \, P_2)$ then $\text{svar}(P) = \text{svar}(P_1)$. Therefore, a variable occurring in $\text{svar}(P)$ is called a *safe variable*, otherwise it is considered *unsafe* in $P$.

**Definition 4 ($\textbf{SPARQL}^{\textbf{NEX}}_{\textbf{safe}}$).** *Define $SPARQL^{NEX}_{safe}$ as the fragment of SPARQL graph patterns satisfying that the occurrence of a subpattern $(P \, \text{NOT-EXISTS} \, P')$ implies that, for every correlated variable $?X$ between $P$ and $P'$, it holds that $?X \in \text{svar}(P')$.*

Note that, $\text{SPARQL}^{\text{NEX}}$ does not allow graph patterns of the form $(P_1 \, \text{NOT-EXISTS}(P_2 \, \text{NOT-EXISTS} \, P_3))$ where $P_3$ contains correlated variables occurring in $P_1$ but not occurring in $P_2$.

**Lemma 6.** *The* NOT-EXISTS *graph patterns allowed in $SPARQL^{NEX}_{safe}$ are expressible in $SPARQL^{DIFF}$.*

*Proof.* Following the definition of $\text{SPARQL}^{\text{NEX}}_{\text{safe}}$, we have that for any graph pattern $(P_1 \, \text{NOT-EXISTS} \, P_2)$ it satisfies that the domain of $[\![P_2]\!]$ contains all the correlated variables between $P_1$ and $P_2$. Given such condition, it is easy to see that $[\![(P_1 \, \text{DIFF} \, P_2)]\!]$ returns the same solutions as $[\![(P_1 \, \text{NOT-EXISTS} \, P_2)]\!]$.

Based on Lemmas 4, 5 and 6, we can present our main result about the expressive power of the DIFF operator and $\text{SPARQL}^{\text{DIFF}}$.

**Theorem 2.** *$SPARQL^{DIFF}$ contains $SPARQL^{NEX}_{safe}$.*

## 5 Properties of the SPARQL negation operators

In this section we study the behavior of the negation operators in terms of elementary equivalences found in set theory. Specifically, we consider the following axioms concerning set-theoretic differences [15]:

| | |
|---|---|
| **(a)** $A \setminus A \equiv \emptyset$ | **(g)** $A \setminus (A \cap B) \equiv A \setminus B$ |
| **(b)** $A \setminus \emptyset \equiv A$ | **(h)** $A \cap (A \setminus B) \equiv A \setminus B$ |
| **(c)** $\emptyset \setminus A \equiv \emptyset$ | **(i)** $(A \setminus B) \cup B \equiv A \cup B$ |
| **(d)** $A \setminus (A \setminus (A \setminus B)) \equiv A \setminus B$ | **(j)** $(A \cup B) \setminus B \equiv A \setminus B$ |
| **(e)** $(A \cap B) \setminus B \equiv \emptyset$ | **(k)** $A \setminus (B \cap C) \equiv (A \setminus B) \cup (A \setminus C)$ |
| **(f)** $(A \setminus B) \cap B \equiv \emptyset$ | **(l)** $A \setminus (B \cup C) \equiv (A \setminus B) \cap (A \setminus C)$ |

In order to evaluate the set-based equivalences in the context of SPARQL, we need to assume two conditions: (1) As a general rule, a set-based operator requires two "objects" with the same structure (e.g. two tables with the same schema). Such requirement is implicitly satisfied in SPARQL thanks to the definition of solution mappings as partial functions. (2) SPARQL does not provide an explicit operator for intersecting two graph patterns $P_1$ and $P_2$. Note however, that a graph pattern $(P_1 \text{ AND } P_2)$ resembles the intersection operation (under set-semantics) when $[\![P_1]\!]$ and $[\![P_2]\!]$ have the same domain of variables.

Given the above two conditions, we can apply a direct translation from a set-theoretic equivalence to a graph pattern equivalence. Specifically, the set-difference operator will be mapped to a SPARQL negation operator (DIFF, MINUS or NOT-EXISTS), the set-intersection operator will be mapped to AND, and the set-union operator will be replaced by UNION. Considering the specific features of NOT-EXISTS (i.e. correlation of variables), we will restrict our analysis to DIFF and MINUS.

Let $P_1$, $P_2$, $P_3$, $P_4$ and $P_5$ be graph patterns satisfying that $[\![P_1]\!] = \emptyset$, $[\![P_2]\!] = \{\mu_0\}$, $\text{var}(P_3) = \text{var}(P_4)$ and $\text{var}(P_3) \cap \text{var}(P_5) = \emptyset$. By combining the above four graph patterns, we conducted a case-by-case analysis consisting of twenty five cases for equivalences (a)-(j) and one hundred twenty five cases for equivalences (k) and (l). The differences between set and bag semantics were specially considered for equivalences (h), (i), (k) and (l).

DIFF satisfies most equivalences with exception of (h), (i), (k) and (l). Equivalence (h) presents five cases which are not valid under bag semantics, although they are valid under set semantics. A similar condition occurs with ten and seven cases for equivalences (k) and (l) respectively. Additionally, we found ten cases which are not satisfied by equivalence (i). MINUS does not satisfy equivalences (e) to (l). We found several cases where equivalences (e), (f), (g), (j) and (k) are not satisfied. Similarly, there exist multiple cases where equivalences (h), (i) and (l) do not apply under bag semantics, but works for set semantics. We would like to remark that the "odd results" presented by the MINUS operator arise because its restriction about disjoint solution mappings.

In summary, we have that each negation operator presents a particular behavior for the axioms studied here. Although none of them was able to satisfy all the axioms, we think that it does not mean that they are badly defined. In fact, the heterogeneity of the operators is a motivation to study their intrinsic properties and to try the definition of a set of desired properties for negation in SPARQL. The details about our case-by-case analysis are included in the technical report.

## 6 Conclusions

In this paper we presented a systematic analysis of the types of negation supported in SPARQL 1.0 and SPARQL 1.1. After introducing the standard relational negation (the DIFF operator) we were able to build a core and intuitive algebra (the same as the standard relational algebra) in SPARQL and prove that it is able to define the graph pattern operators.

We think that having a clear understanding of the operators of negation of SPARQL helps both, developers of databases, and users of the query language. We also think that the core language we identified (which is precisely the well known and intuitive relational algebra) is a much easier way to express queries for database practitioners, who learn from the beginning SQL, which now with this new algebra, can be found in the world of SPARQL.

**Acknowledgements.** R. Angles and C. Gutierrez are founded by the Millennium Nucleus Center for Semantic Web Research under Grant NC120004. We also thank the anonymous referees for their helpful feedback.

## References

1. Angles, R., Gutierrez, C.: SQL Nested Queries in SPARQL. In: Proc. of the 4th Alberto Mendelzon Workshop on Foundations of Data Management (2010)
2. Angles, R., Gutierrez, C.: Subqueries in SPARQL. In: Proc. of the 5th Alberto Mendelzon Workshop on Foundations of Data Management (2011)
3. Angles, R., Gutierrez, C.: Negation in SPARQL. Talk at 8th Alberto Mendelzon International Workshop on Foundations of Data Management (AMW) (2014)
4. Bárány, V., Cate, B., Segoufin, L.: Guarded negation. Journal of the ACM 62(3), 22:1–22:26 (2015)
5. Bidoit, N.: Negation in rule-based database languages: A survey. Theoretical Computer Science 78(1), 3–83 (1991)
6. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing 9(3), 365–386 (1991)
7. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language - W3C Recommendation. http://www.w3.org/TR/2013/REC-sparql11-query-20130321/ (March 21 2013)
8. Hogan, A., Arenas, M., Mallea, A., Polleres, A.: Everything you always wanted to know about blank nodes. Journal of Web Semantics 27(1) (2014)
9. Kaminski, M., Kostylev, E.V., Cuenca Grau, B.: Semantics and Expressive Power of Subqueries and Aggregates in SPARQL 1.1. In: Proc. of the International Conference on World Wide Web. pp. 227–238 (2016)
10. Kim, W.: On optimizing an SQL-like nested query. ACM Transactions on Database Systems (TODS) 7(3), 443–469 (1982)
11. Melton, J., Simon, A.R.: SQL:1999 Understanding Relational Language Components. Morgan Kaufmann (May 2001)
12. Naqvi, S.A.: Negation as failure for first-order queries. In: Proc. of the Symposium on Principles of Database Systems. pp. 114–122. ACM (1986)
13. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and Complexity of SPARQL. ACM Transactions on Database Systems (TODS) 34(3), 1–45 (2009)
14. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C Recommendation. http://www.w3.org/TR/2008/REC-115-sparql-query-20080115/ (January 15 2008)
15. Suppes, P.: Axiomatic Set Theory. The University Series in Undergraduate Mathematics, D. Van Nostrand Company, Inc. (1960)
16. Wagner, G.: A database needs two kinds of negation. In: Proc. of the Symposium on Mathematical Fundamentals of Database and Knowledge Base Systems. pp. 357–371 (1991)