On the Primitivity of Operators in SPARQL

Xiaowang Zhang, Jan Van den Bussche

Hasselt University and transnational University of Limburg, B-3500 Hasselt, Belgium

Abstract

The paper studies the primitivity of the basic operators UNION, AND, OPTIONAL, FILTER, and SELECT, as they are used in the SPARQL query language. The question of whether one operator can be expressed in terms of the other operators is answered in detail. It turns out that only AND is non-primitive. These results are shown to be insensitive to the choice of semantics for filter conditions (three-valued or two-valued). It is also shown that these two semantics can simulate each other.

Keywords: RDF databases, SPARQL, primitive operator, expressive power

1. Introduction

Currently there is renewed interest in the classical topic of graph databases [AG08b, Woo12, HKVdBZ13]. Much of this interest has been sparked by SPARQL: the query language for RDF. The Resource Description Framework (RDF) [RDF04] is a popular data model for information in the Web. RDF represents information in the form of directed, labeled graphs. The standard query language for RDF data is SPARQL [SPA13]. The current version 1.1 of SPARQL extends SPARQL 1.0 [SPA08] with important features such as aggregation and regular expressions. Other features, such as negation and subqueries, have also been added, but mainly for efficiency reasons, as they were already expressible, in a more roundabout manner, in version 1.0 (this follows from known results to the effect that every relational algebra query is expressible in SPARQL [AG08a].). Hence, it is still relevant to study the fundamental properties of SPARQL 1.0.

The expressive power of SPARQL has been analyzed in its relationship to the relational algebra [Cyg05], SQL [CLF09], Datalog [AG08a, Pol07], and OWL [SP07]. Also the relationship between expressivity and complexity and optimization of evaluation has been studied [PAG09, SML10, LM12].

The main goal of this paper is to understand the primitivity of the basic operators used in SPARQL patterns: AND, UNION, OPT, FILTER, and SELECT. (SELECT, which performs projection, was added as a subquery feature in SPARQL 1.1.) Indeed, primitivity has been a recurring topic in the investigation of database query languages, e.g., [AHV95, CH80, FGL$^+$11]. It turns out that AND is not primitive: adapting an idea of Angles and Gutierrez [AG08a], we can express AND in terms of OPT and FILTER. We show that this is the sharpest result possible, in the sense that without FILTER, or without OPT, AND is not expressible. We also show that AND can no longer be expressed in terms of OPT and FILTER if one insists on a "well-designed" expression [PAG09]. We then proceed to show that the remaining operators are primitive.

In a final section of the paper, we show that the above results are insensitive to the choice of semantics for filter conditions. Indeed, while the official semantics uses a three-valued logic [APG09], a two-valued semantics has been considered as well [PAG09]. Besides, we point out that the choice of semantics has no impact on the expressivity: the two semantics can express each other.

2. SPARQL

In this section we recall the syntax and semantics of SPARQL patterns, closely following the core SPARQL formalization given by Arenas, Gutierrez and Pérez [PAG09, APG09].

2.1. RDF graphs

Let I, B, and L be infinite sets of *IRIs*, *blank nodes* and *literals*, respectively. These three sets are pairwise disjoint. We denote the union $I \cup B \cup L$ by U, and elements of $I \cup L$ will be referred to as *constants*.

A triple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an *RDF triple*. An *RDF graph* is a finite set of RDF triples.

2.2. Syntax of SPARQL patterns

Assume furthermore an infinite set V of *variables*, disjoint from U. The convention is to write variables starting with the character '?'. SPARQL *patterns* are inductively defined as follows.

- Any triple from $(I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)$ is a pattern (called a *triple pattern*).
- If P_1 and P_2 are patterns, then so are the following: P_1 UNION P_2, P_1 AND P_2, and P_1 OPT P_2.
- If P is a pattern and S is a finite set of variables then SELECT$_S(P)$ is a pattern.
- If P is a pattern and C is a constraint (defined next), then P FILTER C is a pattern; we call C the *filter condition*.
  Here, a *constraint* is a boolean combination of *atomic constraints*; an atomic constraint can have one of the three following forms: bound(?x) (*bound*), ?x =?y (*equality*), and ?$x = c$ (*constant equality*), for ?x, ?$y \in V$ and $c \in I \cup L$.

2.3. Semantics of SPARQL patterns

The semantics of patterns is defined in terms of sets of so-called *mappings*, which are simply total functions $\mu : S \to U$ on some finite set S of variables. We denote the domain S of μ by $\text{dom}(\mu)$.

Now given a graph G and a pattern P, we define the semantics of P on G, denoted by $[\![P]\!]_G$, as a set of mappings, in the following manner.

- If P is a triple pattern (u, v, w), then
  $$[\![P]\!]_G := \{\mu : \{u, v, w\} \cap V \to U \mid (\mu(u), \mu(v), \mu(w)) \in G\}.$$
  Here, for any mapping μ and any constant $c \in I \cup L$, we agree that $\mu(c)$ equals c itself. In other words, mappings are extended to constants according to the identity mapping.
- If P is of the form P_1 UNION P_2, then $[\![P]\!]_G := [\![P_1]\!]_G \cup [\![P_2]\!]_G$.
- If P is of the form P_1 AND P_2, then $[\![P]\!]_G := [\![P_1]\!]_G \bowtie [\![P_2]\!]_G$, where, for any two sets of mappings Ω_1 and Ω_2, we define
  $$\Omega_1 \bowtie \Omega = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1 \text{ and } \mu_2 \in \Omega_2 \text{ and } \mu_1 \sim \mu_2\}.$$

Table 1: Truth tables for the three-valued semantics.

p	q	$p \wedge q$	$p \vee q$
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*

p	$\neg p$
true	*false*
false	*true*
error	*error*

Here, two mappings μ_1 and μ_2 are called *compatible*, denoted by $\mu_1 \sim \mu_2$, if they agree on the intersection of their domains, i.e., if for every variable $?x \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$, we have $\mu_1(?x) = \mu_2(?x)$. Note that when μ_1 and μ_2 are compatible, their union $\mu_1 \cup \mu_2$ is a well-defined mapping; this property is used in the formal definition above.

- If P is of the form P_1 OPT P_2, then
$$[\![P]\!]_G := ([\![P_1]\!]_G \bowtie [\![P_2]\!]_G) \cup ([\![P_1]\!]_G \smallsetminus [\![P_2]\!]_G),$$
where, for any two sets of mappings Ω_1 and Ω_2, we define
$$\Omega_1 \smallsetminus \Omega_2 = \{\mu_1 \in \Omega_1 \mid \neg\exists \mu_2 \in \Omega_2 : \mu_1 \sim \mu_2\}.$$
- If P is of the form $\text{SELECT}_S(P_1)$, then $[\![P]\!]_G = \{\mu|_{S \cap \text{dom}(\mu)} \mid \mu \in [\![P_1]\!]_G\}$, where $f|_X$ denotes the standard mathematical notion of restriction of a function f to a subset X of its domain.
- Finally, if P is of the form P_1 FILTER C, then $[\![P]\!]_G := \{\mu \in [\![P_1]\!]_G \mid \mu(C) = \textit{true}\}$.
Here, for any mapping μ and constraint C, the evaluation of C on μ, denoted by $\mu(C)$, is defined in terms of a three-valued logic with truth values *true*, *false*, and *error*. Recall that C is a boolean combination of atomic constraints.
For a bound constraint bound$(?x)$, we define:
$$\mu(\text{bound}(?x)) = \begin{cases} \textit{true} & \text{if } ?x \in \text{dom}(\mu); \\ \textit{false} & \text{otherwise.} \end{cases}$$
For an equality constraint $?x = ?y$, we define:
$$\mu(?x = ?y) = \begin{cases} \textit{true} & \text{if } ?x, ?y \in \text{dom}(\mu) \text{ and } \mu(?x) = \mu(?y); \\ \textit{false} & \text{if } ?x, ?y \in \text{dom}(\mu) \text{ and } \mu(?x) \neq \mu(?y); \\ \textit{error} & \text{otherwise.} \end{cases}$$
Thus, when $?x$ and $?y$ do not both belong to dom(μ), the equality constraint evaluates to *error*. Similarly, for a constant-equality constraint $?x = c$, we define:
$$\mu(?x = c) = \begin{cases} \textit{true} & \text{if } ?x \in \text{dom}(\mu) \text{ and } \mu(?x) = c; \\ \textit{false} & \text{if } ?x \in \text{dom}(\mu) \text{ and } \mu(?x) \neq c; \\ \textit{error} & \text{otherwise.} \end{cases}$$
A boolean combination is then evaluated using the truth tables given in Table 1.

3. Primitivity of operators

Let us abbreviate the operator AND by $\mathcal{A}$; FILTER by $\mathcal{F}$; OPT by $\mathcal{O}$; SELECT by $\mathcal{S}$; and UNION by $\mathcal{U}$. Then we can denote any fragment of SPARQL, where only a subset of the five operators is available, by the letter word formed by the operators that are available in the fragment. Thus, for example, $\mathcal{AFSU}$ denotes the fragment where OPT is disallowed.

Since OPT is the least conventional operator of the five, one may wonder whether it is not already expressible in terms of the other four operators. The answer is negative, however, in view of the following well-known and easy-to-prove proposition.

Proposition 1. *Every pattern P in $\mathcal{AFSU}$ is monotone. That is, for any two graphs $G_1 \subseteq G_2$, we have $[\![P]\!]_{G_1} \subseteq [\![P]\!]_{G_2}$.*

Formally, we should define what we mean when we say that some operator $\mathcal{X}$ is "expressible" in some fragment $\mathcal{W}$. We will simply take this to mean here that *for every pattern P in the fragment $\mathcal{WX}$* (i.e., adding $\mathcal{X}$ to $\mathcal{W}$) *there exists a pattern Q in the given fragment $\mathcal{W}$ such that for all graphs G, we have $[\![P]\!]_G = [\![Q]\!]_G$.*

We can now conclude:

Corollary 2. *OPT is not expressible in $\mathcal{AFSU}$.*

Proof. Consider the pattern $P = (?x, p, ?y)$ OPT $(?y, q, ?z)$. Suppose, for the sake of contradiction, that P would be expressible as Q for some $\mathcal{AFSU}$ pattern Q.

Consider the graphs $G_1 = \{(a, p, b)\}$ and $G_2 = G_1 \cup \{(b, q, c)\}$. Then $[\![P]\!]_{G_1} = \{\mu_1\}$ and $[\![P]\!]_{G_2} = \{\mu_2\}$ where $\mu_1 = (?x = a, ?y = b)$ and $\mu_2 = (?x = a, ?y = b, ?z = c)$. Since $[\![P]\!]_{G_1} = [\![Q]\!]_{G_1}$, we have $\mu_1 \in [\![Q]\!]_{G_1}$. Since $G_1 \subseteq G_2$, by Proposition 1, we have $[\![Q]\!]_{G_1} \subseteq [\![Q]\!]_{G_2}$. Hence, $\mu_1 \in [\![Q]\!]_{G_2} = [\![P]\!]_{G_2}$. The latter set equals $\{\mu_2\}$, however, and since $\mu_1 \neq \mu_2$, we have arrived at a contradiction. □

Remark 3. The example in the above proof suggests that patterns involving OPT might satisfy some weaker notion of monotonicity, involving extension of mappings rather than straight inclusion of one set of mappings in another. This so-called "weak monotonicity" still does not hold in general, but it holds for so-called "well-designed" patterns [AP11]. □

We next show that AND is not primitive:

Proposition 4. *AND is expressible in $\mathcal{FO}$.*

Proof. First we recall the MINUS operator [AP11]. For any two patterns P and Q, we can consider the extended pattern P MINUS Q with the following semantics: for any graph G, $[\![P \text{ MINUS } Q]\!]_G = [\![P]\!]_G \setminus [\![Q]\!]_G$.

It is well known [AG08a, AP11] that P MINUS Q is expressible as (P OPT (Q AND $(?x, ?y, ?z)$)) FILTER $\neg$ bound$(?x)$, where $?x$, $?y$ and $?z$ are fresh variables not occurring in P or Q. Actually, we can express MINUS already in the fragment $\mathcal{OF}$ because in the above expression, we can replace AND by OPT without changing the semantics (this works because $?x$, $?y$, $?z$ are fresh variables.).

We can now express P_1 AND P_2 as (P_1 OPT P_2) MINUS (P_1 MINUS P_2). To see the correctness of this expression, first, using the equality P_1 OPT P_2 = (P_1 AND P_2) UNION (P_1 MINUS P_2), we can rewrite it as

$$((P_1 \text{ AND } P_2) \text{ MINUS } (P_1 \text{ MINUS } P_2)) \text{ UNION } ((P_1 \text{ MINUS } P_2) \text{ MINUS } (P_1 \text{ MINUS } P_2)).$$

The second term in the above expression is empty, so we can concentrate on the first term and must show that it is equivalent to P_1 AND P_2. Thereto, for any two sets of mappings Ω_1 and Ω_2, we must show that $\Omega_1 \bowtie \Omega_2$ is contained in $(\Omega_1 \bowtie \Omega_2) \smallsetminus (\Omega_1 \smallsetminus \Omega_2)$; the other containment is obvious. Thus, let $\mu \in \Omega_1 \bowtie \Omega_2$ and let $\mu' \in \Omega_1 \smallsetminus \Omega_2$ be arbitrary. We must show that $\mu \nsim \mu'$. Assume, for the sake of contradiction, that $\mu \sim \mu'$. We have $\mu = \mu_1 \cup \mu_2$ for some $\mu_1 \in \Omega_1$ and $\mu_2 \in \Omega_2$. Since $\mu \sim \mu'$ and $\mu_2 \subseteq \mu$, also $\mu_2 \sim \mu'$. However, since $\mu_2 \in \Omega_2$, this contradicts that $\mu' \in \Omega_1 \smallsetminus \Omega_2$.

□

One may wonder whether OPT and filters are really needed in order to express AND. That OPT is really needed is quite obvious, in view of the property that any pattern using neither AND nor OPT can return mappings with at most three variables in their domain only. This property is clearly violated by the pattern $(?x, ?y, ?z)$ AND $(?u, ?v, ?w)$, for example. We next confirm that filters are really needed to express AND; this is Proposition 6. First we need a technical lemma.

Lemma 5. *For every $\mathcal{OSU}$ pattern P there exists a singleton graph G (graph consisting of a single RDF triple) such that $[\![P]\!]_G$ is nonempty.*

Proof. By induction on the structure of P. If P is a triple pattern $(?x, ?y, ?z)$, choose an arbitrary mapping $\mu\colon \{?x, ?y, ?z\} \cap V \to I$ and let $G = \{(\mu(?x), \mu(?y), \mu(?z))\}$. If P is P_1 UNION P_2, the claim readily follows by induction. If P is P_1 OPT P_2, then by induction, P_1 is nonempty on some singleton graph G. But then by the semantics of OPT, P is nonempty on G as well. Finally, if P is SELECT$_S(P_1)$ then again P_1 is nonempty on some singleton graph G. Hence P is nonempty on G as well. Note that in the worst case, $[\![P_1]\!]_G$ only contains mappings μ with $\mathrm{dom}(\mu) \cap S = \emptyset$. But then still, the projection $\mu|_\emptyset$ of such a mapping yields the empty mapping, which then belongs to $[\![P]\!]_G$, so $[\![P]\!]_G$ is still nonempty. □

Proposition 6. *AND is not expressible in $\mathcal{OSU}$.*

Proof. Consider the pattern $P = (?x, p, ?y)$ AND $(?x, q, ?y)$. It is clear that P is always empty on any singleton graph. By Lemma 5 however, every $\mathcal{OSU}$ pattern Q, is nonempty on some singleton graph. Hence, the claim follows. □

Remark 7. The pattern expressing MINUS, used in the proof of Proposition 4, is not well-designed [PAG09]. Indeed, when restricting to well-designed patterns, it turns out that AND regains its primitivity. Specifically, the following proposition contrasts Proposition 4:

Proposition 8. *AND is not expressible by any union of well-designed $\mathcal{FO}$ patterns.*

Proof. We will show that any well-designed $\mathcal{FO}$ pattern P that is satisfiable ($[\![P]\!]_G$ is nonempty for some graph G) is already satisfiable by a singleton graph. Then this property also follows for unions. Since this property clearly does not hold for $(?x, p, ?y)$ AND $(?x, q, ?y)$, this will prove the proposition.

For any $\mathcal{FO}$ pattern P, we define the set *lmsp*(P) of *leftmost* subpatterns of P, inductively as follows: if P is a triple pattern t, then *lmsp*$(P) := \{t\}$; if P is of the form P_1 FILTER C or P_1 OPT P_2, then *lmsp*$(P) :=$ *lmsp*$(P_1) \cup \{P\}$. There is exactly one triple pattern t in *lmsp*(P), which is the leftmost leaf in the parse tree of P; the leftmost subpatterns correspond to the nodes on the path from the lefmost leaf to the root. Furthermore, for every $Q \in$ *lmsp*(P) we define Q' inductively as follows: if Q is a triple pattern t, then $Q' := t$; if Q is of the form Q_1 FILTER C then $Q' := Q_1'$ FILTER C; if Q is of the form Q_1 OPT Q_2 then $Q' := Q_1'$.

It follows from the definition of well-designedness [PAG09] that if P is well-designed, and Q_1 FILTER C is in *lmsp*(P), then all variables from C already occur in the leftmost triple pattern t of P. Using this observation, we can prove the following claim. *Let P be a well-designed pattern in $\mathcal{FO}$ and let t be its leftmost triple pattern; let S be the set of variables in t. Let $Q \in lmsp(P)$. Then for any graph G and any $\mu \in [\![Q]\!]_G$ we have $S \subseteq \mathrm{dom}(\mu)$ and $\mu|_S \in [\![Q']\!]_{\{\mu(t)\}}$.* We prove the claim by induction on the height of Q. If Q is t itself, the claim is trivial. If Q is of the form Q_1 FILTER C, then $\mu \in [\![Q_1]\!]_G$ and $\mu(C) = \mathit{true}$. By induction, $S \subseteq \mathrm{dom}(\mu)$ and $\mu|_S \in [\![Q'_1]\!]_{\{\mu(t)\}}$. Since all variables in C already occur in S, we have $\mu|_S(C) = \mu(C) = \mathit{true}$ so $\mu|_S \in [\![Q'_1 \text{ FILTER } C]\!]_{\{\mu(t)\}} = [\![Q']\!]_{\{\mu(t)\}}$. If Q is of the form Q_1 OPT Q_2, we can always write $\mu = \mu_1 \cup \mu_2$ with $\mu_1 \in [\![Q_1]\!]_G$. By induction, $S \subseteq \mathrm{dom}(\mu_1) \subseteq \mathrm{dom}(\mu)$ and $\mu|_S = \mu_1|_S \in [\![Q'_1]\!]_{\{\mu_1(t)\}} = [\![Q'_1]\!]_{\{\mu(t)\}} = [\![Q']\!]_{\{\mu(t)\}}$, as desired.

Applying the claim to $Q = P$, we obtain that if P evaluates to a nonempty result on some graph G, then P' evaluates to a nonempty result on some singleton graph H. By Lemma 4.3 of Pérez et al. [PAG09] (again using that P is well-designed), this in turn implies that P is nonempty on H, which is what we have to prove. □

Let us now turn to the question of primitivity of FILTER. Thereto we need another lemma which uses a simplified notion of *mapping scheme* of a pattern, defined as follows: $MS(u, v, w) := \{\{u, v, w\} \cap V\}$; $MS(P_1 \text{ UNION } P_2) := MS(P_1) \cup MS(P_2)$; $MS(P_1 \text{ OPT } P_2) = MS(P_1 \text{ AND } P_2) := \{M_1 \cup M_2 \mid M_1 \in MS(P_1), M_2 \in MS(P_2)\}$; and $MS(\text{SELECT}_S(P_1)) := \{M \cap S \mid M \in MS(P_1)\}$.

Lemma 9. *Let G be the complete graph on two constants $a, b \in I$ (i.e., $G = \{a, b\} \times \{a, b\} \times \{a, b\}$). Let P be any $\mathcal{AOSU}$ pattern in which no other constants occur than a or b. Then $[\![P]\!]_G$ consists of all possible mappings $\mu\colon M \to \{a, b\}$ with $M \in MS(P)$.*

Proof. By induction on the structure of P. If P is a triple pattern (u, v, w), we are given that each of u, v and w is either a variable or a constant in $\{a, b\}$. From the definition of G it is then clear that every mapping $\mu\colon \{u, v, w\} \cap V \to \{a, b\}$ belongs to $[\![P]\!]_G$.

If P is of the form P_1 UNION P_2, the claim readily follows by induction.

If P is of the form P_1 AND P_2, let $\mu : M_1 \cup M_2 \to \{a, b\}$ be arbitrary with $M_i \in MS(P_i)$ for $i = 1, 2$. Let $\mu_i := \mu|_{M_i}$. By induction, $\mu_i \in [\![P_i]\!]_G$, and $\mu_1 \sim \mu_2$ since they are both restrictions of the same mapping μ. Hence, $\mu = \mu_1 \cup \mu_2 \in [\![P]\!]_G$ as desired. Conversely, any $\mu \in [\![P]\!]_G$ is of the form $\mu_1 \cup \mu_2$ with $\mu_i \in [\![P_i]\!]_G$. By induction, $\mathrm{dom}(\mu_i) \in MS(P_i)$, so $\mathrm{dom}(\mu) = \mathrm{dom}(\mu_1) \cup \mathrm{dom}(\mu_2) \in MS(P)$ as desired.

If P is of the form P_1 OPT P_2, the claim follows from the previous case since $[\![P]\!]_G = [\![P_1 \text{ AND } P_2]\!]_G$ for the complete graph G under consideration. Indeed, take any $\mu_1 \in [\![P_1]\!]_G$. Take any $S_2 \in MS(P_2)$ and consider a mapping $\mu_2 : S_2 \to \{a, b\}$ compatible to μ_1. By induction, $\mu_2 \in [\![P_2]\!]_G$. Hence, for every $\mu_1 \in [\![P_1]\!]_G$ there exists a compatible $\mu_2 \in [\![P_2]\!]_G$.

Finally, if P is of the form $\text{SELECT}_S(P_1)$, let $\mu : S \cap M \to \{a, b\}$ be arbitrary with $M \in MS(P_1)$. Let $\bar{\mu} : M \to \{a, b\}$ be an arbitrary extension of μ to M. By induction, $\bar{\mu} \in [\![P_1]\!]_G$. Hence, $\mu = \bar{\mu}|_{S \cap \mathrm{dom}(\bar{\mu})} \in [\![P]\!]_G$ as desired. Conversely, any $\mu \in [\![P]\!]_G$ is of the form $\bar{\mu}|_{S \cap \mathrm{dom}(\bar{\mu})}$ with $\bar{\mu} \in [\![P_1]\!]_G$. By induction, $\mathrm{dom}(\bar{\mu}) \in MS(P_1)$. Hence, $\mathrm{dom}(\mu) = S \cap \mathrm{dom}(\bar{\mu}) \in MS(P)$ as desired. □

As a consequence we obtain:

Proposition 10. *FILTER is not expressible in $\mathcal{AOSU}$.*

Proof. Consider the pattern $P = (?x, ?y, ?z)$ FILTER $?x = ?y$. Suppose, for the sake of contradiction, that P is expressible as Q for some $\mathcal{AOSU}$ pattern Q.

Let G be the graph from Lemma 9. It is clear that any triple pattern involving constants other than a and b is empty on G. Hence, in Q we may replace all these patterns by the special expression $\emptyset$ standing for the empty pattern (the pattern that evaluates to the empty set on every graph). We can then normalize these expressions $\emptyset$ away by applying the following rewrite rules:

$$\begin{array}{ccc} \emptyset \text{ AND } P_2 \to \emptyset, & P_1 \text{ AND } \emptyset \to \emptyset, & \emptyset \text{ UNION } P_2 \to P_2, \\ P_1 \text{ UNION } \emptyset \to P_1, & \emptyset \text{ OPT } P_2 \to \emptyset, & P_1 \text{ OPT } \emptyset \to P_1, \end{array}$$

and $\text{SELECT}_S(\emptyset) \to \emptyset$. If the entire pattern rewrites to $\emptyset$, then Q is empty on G, while P is not empty on G, which is impossible. Hence, we may now assume that Q involves no constants other than a and b, so that Lemma 9 applies.

Since P returns mappings with domain equal to $\{?x, ?y, ?z\}$, and Q expresses P on G, by Lemma 9, the set $\{?x, ?y, ?z\}$ must belong to $MS(Q)$. However, by the same Lemma, the mapping $(?x = a, ?y = b, ?z = a)$ belongs to $[\![Q]\!]_G = [\![P]\!]_G$. Since this mapping does not satisfy the filter condition, however, we have arrived at a contradiction. $\square$

The primitivity of UNION is already known: it follows from the known property [PAG09, Claim 3.9] that a UNION-free pattern can never return two different but compatible mappings, whereas this is clearly possible for the pattern $(?x, p, ?y)$ UNION $(?y, q, ?z)$ on the graph $\{(a, p, b), (b, q, c)\}$. One can actually see the primitivity of UNION in a different way as well. It is readily verified that UNION-free patterns evaluated on a singleton graph can return at most one mapping, whereas the pattern $(?x, r, ?y)$ UNION $(?y, r, ?x)$ returns two mappings on the singleton graph $\{(a, r, b)\}$.

Finally, we discuss the primitivity of SELECT. The following lemma and proposition provide the result.

Let P be a pattern and G be a graph. We denote $[\![P]\!]_G^{?x} = \{\mu \in [\![P]\!]_G \mid \text{dom}(\mu) = \{?x\}\}$.

An $?x$-variable triple pattern is a triple pattern in which $?x$ is the only occurring variable (i.e., *in one of the forms* $(?x, a, b)$, $(a, ?x, b)$, $(a, b, ?x)$, $(?x, a, ?x)$, $(a, ?x, ?x)$, $(?x, ?x, a)$, *or* $(?x, ?x, ?x)$).

Lemma 11. *For every pattern P in $\mathcal{AFOU}$, there exists some finite set T of $?x$-variable triple patterns such that for every graph G and for every $\mu \in [\![P]\!]_G^{?x}$, there exists some triple pattern $t \in T$ such that $\mu \in [\![t]\!]_G$.*

Proof. By induction on the structure of P. Basically, if $P = t$ is a triple pattern and t is not an $?x$-variable triple pattern then $T = \emptyset$ since $[\![t]\!]_G^{?x} = \emptyset$ for any graph G. But, if t is an $?x$-variable triple pattern then $T = \{t\}$.

Inductively, if P is of the form P_1 UNION P_2 or P_1 FILTER C, the claims follows by induction.

If P is of the form P_1 AND P_2, by induction, we have set T_1 and T_2 for P_1 and P_2 respectively. We set $T = T_1 \cup T_2$. Let $\mu \in [\![P]\!]_G^{?x}$. Since $\mu \in [\![P]\!]_G$, there exist $\mu_1 \in [\![P_1]\!]_G$ and $\mu_2 \in [\![P_2]\!]_G$ such that $\mu = \mu_1 \cup \mu_2$ and $\mu_1 \sim \mu_2$. Since $\text{dom}(\mu) = \{?x\}$, there are three cases:

- $\text{dom}(\mu_1) = \emptyset$ and $\text{dom}(\mu_2) = \{?x\}$. In this case, $\mu = \mu_2$. By induction, $\mu_2 \in [\![t]\!]_G$ for some $t \in T_2 \subseteq T$.
- $\text{dom}(\mu_1) = \{?x\}$ and $\text{dom}(\mu_2) = \emptyset$. In this case, $\mu = \mu_1$. By induction, $\mu_1 \in [\![t]\!]_G$ for some $t \in T_1 \subseteq T$.
- $\text{dom}(\mu_1) = \{?x\} = \text{dom}(\mu_2)$. In this case, $\mu = \mu_1 = \mu_2$. Again the claim follows by induction.

Finally, let P be of the form P_1 OPT P_2. We again seet $T = T_1 \cup T_2$. If $\mu \in [\![P]\!]_G$ then either $\mu \in [\![P_1 \text{ AND } P_2]\!]_G$ or $\mu \in [\![P_1]\!]_G$. In the first case, we reason as for P_1 AND P_2; and the second case follows immediately by induction. □

Proposition 12. *SELECT is not expressible in $\mathcal{AFOU}$.*

Proof. Consider the pattern $P = \text{SELECT}_{?x}((?x, ?y, ?z))$. Suppose, for the sake of contradiction, that P is expressible as Q for some $\mathcal{AFOU}$ pattern Q. In this sense, $[\![P]\!]_G = [\![Q]\!]_G = [\![Q]\!]_G^{?x}$. Let $G = \{(a, p, b)\}$ be a singleton graph where a, p, b do not occur as constants in Q. Thus for every $?x$-variable triple pattern t occurring in Q, we have $[\![t]\!]_G = \emptyset$. Then, by Lemma 11, we can conclude that $[\![Q]\!]_G^{?x} = \emptyset$. However, $[\![P]\!]_G = \{?x = a\}$. We have arrived at the contradiction. □

4. Two- versus three-valued semantics

As an alternative [PAG09] to the three-valued semantics for filter conditions, we may redefine the evaluation of a constraint C on a mapping μ, now denoted by $\mu(C)_2$, in terms of standard two-valued logic, as follows. For bound constraints, the definition does not change.

For equality and constant-equality constraints, we redefine:

$$\mu(?x = ?y)_2 := \begin{cases} \mathit{true} & \text{if } ?x, ?y \in \text{dom}(\mu) \text{ and } \mu(?x) = \mu(?y); \\ \mathit{false} & \text{otherwise.} \end{cases}$$

$$\mu(?x = c)_2 := \begin{cases} \mathit{true} & \text{if } ?x \in \text{dom}(\mu) \text{ and } \mu(?x) = c; \\ \mathit{false} & \text{otherwise.} \end{cases}$$

A boolean combination is then evaluated using classical boolean logic.

We can then define the *two-valued semantics* of a pattern P on a graph G, denoted by $[\![P]\!]_G^2$, in exactly the same way as above, except that the semantics of filter patterns now uses the two-valued evaluation of filter conditions. Formally, if P is of the form P_1 FILTER C, then $[\![P]\!]_G^2 := \{\mu \in [\![P_1]\!]_G^2 \mid \mu(C)_2 = \mathit{true}\}$.

Example 13. Let G be the graph consisting of a single triple (a, p, b) with $a, p, b \in I$. Consider the following pattern P: $((?x, p, ?y) \text{ OPT } (?y, q, ?z)) \text{ FILTER } \neg(?x = ?z)$, where $?x$, $?y$ and $?z$ are variables and $q \in I$ is a constant different from p. Then $[\![P]\!]_G$ is empty, but $[\![P]\!]_G^2$ is not, containing the mapping $\mu = (?x = a, ?y = b)$. Indeed, $[\![(?x, p, ?y)\text{OPT}(?y, q, ?z)]\!]_G = \{\mu\}$. Since $?z \notin \text{dom}(\mu)$, we have $\mu(\neg(?x = ?z))_2 = \mathit{true}$ whereas $\mu(\neg ?x = ?z) = \mathit{error}$. □

The above example suggests the two semantics differ mainly in their treatment of negation. And indeed we have the following proposition. A pattern is called *positive* if every filter condition is a positive constraint; a constraint is called positive if it is built up from atomic constraints using only disjunction and conjunction, but no negation.

Proposition 14. *For every positive pattern P and every graph G, we have $[\![P]\!]_G = [\![P]\!]_G^2$.*

Proof. The claim follows from the observation that, for every positive constraint C and every mapping μ, we have $\mu(C) = \mathit{true}$ if and only if $\mu(C)_2 = \mathit{true}$. This is readily verified by induction on the structure of C. □

Nevertheless, we have the following theorem which shows that the choice of semantics does not impact expressivity.

Proposition 15. *For every pattern P there exists a pattern Q such that for every graph G, we have $[\![P]\!]_G = [\![Q]\!]^2_G$. Conversely, for every pattern Q there exists a pattern P such that for every graph G, we have $[\![Q]\!]^2_G = [\![P]\!]_G$.*

Proof. For any constraint C, define $C^{(2)}$ to be the constraint obtained from C by replacing every equality constraint $?x = ?y$ by $(?x = ?y \wedge \text{bound}(?x) \wedge \text{bound}(?y))$, and every constant-equality constraint $?x = c$ by $(?x = c \wedge \text{bound}(?x))$. It is then readily verified by induction that for every mapping μ, we have $\mu(C) = \mu(C^{(2)})_2$. Hence, replacing every filter condition C in P by $C^{(2)}$, we obtain the desired pattern Q.

For the converse direction, from Q to P, we first observe that disjunction in filter conditions can be avoided. Indeed, constraints can be put into disjunctive normal form, and P FILTER $(C_1 \vee C_2)$ is equivalent to (P FILTER C_1) UNION (P FILTER C_2). (This observation holds for the three-valued and the two-valued semantics alike.) Furthermore, the double negation axiom $\neg\neg C \equiv C$ is also valid in three-valued logic. So, we may assume that every filter condition in Q is a conjunction of possibly negated atomic constraints.

Now consider such a constraint C that is a conjunction of possibly negated atomic constraints. Then define $C^{(3)}$ as the constraint obtained from C by leaving every unnegated atomic constraint untouched, and replacing every negated equality constraint $\neg(?x = ?y)$ by $(\neg(?x = ?y) \vee \neg\,\text{bound}(?x) \vee \neg\,\text{bound}(?y))$, and every negated constant-equality constraint $\neg(?x = c)$ by $(\neg(?x = c) \vee \neg\,\text{bound}(?x))$. It is then readily verified by induction that for every mapping μ, we have $\mu(C)_2 = \mu(C^{(3)})$. By rewriting all filter conditions in Q in this manner we obtain the desired pattern P. $\square$

It is natural to ask whether the results of Section 3 still hold under the two-valued semantics. Indeed, it can be verified that all the proofs of that section remain valid. Hence we can conclude the main result of this paper.

Theorem 16. *The only non-primitive operator is* AND*, both under the three-valued as under the two-valued semantics.*

5. Conclusion

In order that programmers who use SPARQL can get the most out of their queries, as well as implementers of SPARQL processing engines have the widest array of techniques available, it is important to have as much insight as possible in the precise semantics of SPARQL patterns and the interplay among their several operators [SML10]. In this note, we have focused on the basic operators present in SPARQL 1.0. Similar investigations can be done on the manyfold extensions of SPARQL that are being considered, not in the least in SPARQL 1.1, and indeed investigations along these lines are currently ongoing, e.g., [FGL$^+$11, BPR12, LPPS12].

Acknowledgment

We would like to thank the anonymous referees for their critical comments and questions which helped us to substantially improve the paper. In particular, one of the referees pointed out mistakes in a previous draft and asked about what is now Proposition 8. We also thank Claudio Gutierrez for a helpful e-mail communication. This work has been funded by the project of Research Foundation Flanders (FWO) under grant G.0489.10N.

[AG08a] R. Angles, C. Gutierrez, The expressive power of SPARQL, in: A. Sheth, S. Staab, et al. (Eds.), Proc. of ISWC'08, LNCS 5318, Springer, 2008, pp.114–129.

[AG08b] R. Angles and C. Gutierrez. Survey of graph database models, ACM Comput. Surv., 40(1)(2008): article 1.

[AHV95] S. Abiteboul, R. Hull, V. Vianu. *Foundations of Databases*, Addison-Wesley, 1995.

[AP11] M. Arenas, J. Pérez, Querying semantic web data with SPARQL, in: M. Lenzerini, T. Schwentick (Eds.), Proc. of PODS'11, ACM, 2011, pp. 305–316.

[APG09] M. Arenas, J. Pérez, C. Gutierrez, On the semantics of SPARQL, in: R. De Virgilio, F. Giunchiglia, L. Tanca (Eds.), Semantic Web Information Management—A Model-Based Perspective, Springer, 2009, pp. 281–307.

[BPR12] P. Barceló, J. Pérez, J.L. Reutter, Relative expressiveness of nested regular expressions, in: J. Freire, D. Suciu (Eds.) Proc. of AMW'12, vol. 866 of CEUR Workshop Proc., 2012, pp. 180–195.

[CH80] A.K. Chandra, D. Harel, Computable queries for relational data bases, J. Comput. Syst. Sci., 21(2)(1980):156–178.

[CLF09] A. Chebotko, S. Lu, F. Fotouhi, Semantics preserving SPARQL-to-SQL translation, Data Knowl. Eng., 68(10)(2009):973–1000.

[Cyg05] R. Cyganiak, A relational algebra for SPARQL, Technical Report HPL-2005-170, HP Labs, 2005.

[FGL$^+$11] G.H.L. Fletcher, M. Gyssens, D. Leinders, J. Van den Bussche, D. Van Gucht, S. Vansummeren, Y. Wu, Relative expressive power of navigational querying on graphs, in: T. Milo (Ed.), Proc. of ICDT'11, ACM, 2011, pp.197–207

[HKVdBZ13] J. Hellings, B. Kuijpers, J. Van den Bussche, X. Zhang, Walk logic as a framework for path query languages on graph databases, in: W. Tan et al., Proc. of ICDT'13, ACM, 2013, pp.117–128.

[LM12] K. Losemann, W. Martens, The complexity of evaluating path expressions in SPARQL, in: M. Benedikt, M. Krötzsch, M. Lenzerini, Proc. of PODS'12, ACM, 2012, pp. pp. 101–112.

[LPPS12] A. Letelier, J. Pérez, R. Pichler, S. Skritek, Static analysis and optimization of semantic web queries, in: M. Benedikt, M. Krötzsch, M. Lenzerini, Proc. of PODS'12, ACM, 2012, pp. 89–100.

[PAG09] J. Pérez, M. Arenas, C. Gutierrez, Semantics and complexity of SPARQL. ACM Trans. Database Syst., 34(3)(2009):article 16.

[Pol07] A. Polleres. From SPARQL to rules (and back), in: C.L. Williamson, M.E. Zurko (Eds.), Proc. of WWW'07, ACM, 2007, pp. 787–796.

[RDF04] RDF primer, W3C Recommendation, February 2004.

[SML10] M. Schmidt, M. Meier, G. Lausen, Foundations of SPARQL query optimization, in: L. Segoufin (Ed.), Proc. of ICDT'10, 2010, ACM, pp. 4–33.

[SP07] E. Sirin and B. Parsia. SPARQL-DL: SPARQL query for OWL-DL, in: C. Golbreich, A. Kalyanpur, B. Parsia (Eds.), Proc. of OWLED'07, vol. 258 of CEUR Workshop Proc., 2007.

[SPA08] SPARQL query language for RDF, W3C Recommendation, January 2008.

[SPA13] SPARQL 1.1 query language, W3C Recommendation, March 2013.

[Woo12] P. Wood, Query languages for graph databases, *SIGMOD Record*, 41(1) (2012): 50–60.