On the Primitivity of SPARQL 1.1 Operators

Xiaowang Zhang
School of Computer Science and Technology, Tianjin University, Tianjin 300350, China
Tianjin Key Laboratory of Cognitive Computing and Application, Tianjin 300350, China
Key Laboratory of Computer Network and Information Integration, Ministry of Education, Nanjing, China
xiaowangzhang@tju.edu.cn

ABSTRACT

The paper studies the primitivity of the eleven basic operators used in the SPARQL 1.1 query language. This paper shows that the six operators BIND, FILTER, GRAPH, property path, SELECT, and VALUES are primitive while the left five operators AND, EXISTS, MINUS, OPT, and UNION, are not primitive. It is surprising that OPT and UNION which are primitive in SPARQL 1.0 become no longer primitive in SPARQL 1.1.

Keywords

RDF databases; SPARQL 1.1; primitive operator; expressive power

1. INTRODUCTION

As a recurring interest topic in the classical topic of graph databases [1], the primitivity of an operator is to determine whether the operator can be expressed in terms of the other operators. The standard query language for RDF (Resource Description Framework) data, a popular data model for information in the Web, is SPARQL 1.1 [4] by extending SPARQL 1.0 [3] with important features such as negation, subqueries, aggregation, and regular expressions, which those features will enrich the ability to represent more expressive queries [2, 6].

Zhang and Van den Bussche [5] investigated that, among five SPARQL 1.0 operators: AND, UNION, OPT, FILTER, and SELECT, only AND is not primitive where AND can be expressible by OPT and FILTER.

The main goal of this paper is to investigate the primitivity of the eleven basic operators in the SPARQL 1.1 query language. We show that the six operators BIND, FILTER, GRAPH, property path, SELECT, and VALUES are primitive while the left five operators AND, EXISTS, MINUS, OPT, and UNION, are not primitive.

This paper is further organized as follows. In the next section, we introduce syntax and semantics of SPARQL 1.1 operators. Section 3 shows the five non-primitive operators and Section 4 shows the left six primitive operators.

2. RDF AND SPARQL 1.1

In this section we recall SPARQL1.1 operators, closely following the SPARQL formalization in [4].

RDF graphs.

Let I, B, and L be infinite sets of IRIs, blank nodes and literals, respectively.

A triple (s, p, o) ∈ (I ∪ B) × I × (I ∪ B ∪ L) is called an RDF triple. An RDF graph is a finite set of RDF triples.

A dataset DS is of the form (G, {{g1, G1}, ..., {gk, Gk}}) where G, G1, ..., Gk are RDF graphs and g1, ..., gk are IRIs.

We call G the default graph and (g1, G1), ..., (gk, Gk) named graphs. Let named(DS) = {g1, ..., gk} and DS(def) = G and DS(gi) = Gi for i = 1, ..., k.

SPARQL 1.1 operators.

SPARQL 1.1 patterns are constructed by using triple patterns (possibly adopting property paths) and operators: AND, OPT, FILTER, UNION, GRAPH, SELECT, EXISTS, BIND, MINUS, and VALUES, where SELECT (nested operation is not allowed in SPARQL 1.0), EXISTS, MINUS, BIND, and VALUES are newly added in SPARQL 1.1. The semantics of patterns is defined in terms of sets of so-called mappings, which are simply total functions μ: S → U on some finite set S of variables. The semantics is based on a three-valued logic with truth values true, false, and error and the semantics in this paper is set-based while the semantics is bag-based in the practical applications [3].

3. EXPRESSIVITY OF OPERATORS

Let us abbreviate the operator AND by A; BIND by B; EXISTS by E; FILTER by F; MINUS by M; GRAPH by G; OPT by O; property path by P; subqueries by S; UNION by U; and VALUES by V. We can denote any fragment of SPARQL 1.1, where only a subset of those operators is available, by the letter word formed by the operators that are available in the fragment.

An operator can not contribute the expressivity of a fragment if it is expressible by the fragment. Formally, we should define what we mean when we say that some operator X is “expressible” in some fragment W. We will simply take this to mean here that for every pattern P in the fragment W.X (i.e., adding X to W) there exists a pattern Q in the given fragment W such that for any graph G, we have
In this sense, we say that $P$ and $Q$ are equivalent, denoted by $P \equiv Q$.

We know that EXISTS is already expressible in $\mathcal{AFS}$ and MINUS is already expressible in $\mathcal{OFS}$ as follows:

\[ P \text{ FILTER EXISTS}(Q) \equiv \text{SELECT}_{\text{var}(P)}(P \text{ AND } Q). \]

\[ P \text{ MINUS } Q \equiv \text{SELECT}_{\text{var}(P)}((P \text{ OPT } (?x, ?y, ?z)) \text{ OPT } (Q \text{ OPT } (?x, ?y, ?u)) \text{ FILTER } \neg \text{bound}(?u)), \]

where $?x, ?y, ?z, ?u$ are fresh variables.

Firstly, we can express OPT in $\mathcal{AMSU}$.

**Proposition 1.** Let $P$ and $Q$ be two patterns.

\[ P \text{ OPT } Q \equiv (P \text{ AND } Q) \text{ UNION } \text{SELECT}_{\text{var}(P)}(P \text{ AND } Q) \text{ OPT } (Q \text{ OPT } (?x, ?y, ?z)) \]

where $?x, ?y, ?z$ are three fresh variables.

Finally, we show that UNION is also expressible in $\mathcal{AOSV}$.

**Proposition 2.** Let $P$ and $Q$ be two patterns, we consider a pattern $Q'$ in $\mathcal{AOSV}$ s.t. $P \text{ UNION } Q \equiv Q'$ as follows:

\[ Q' = \text{SELECT}_S(((P_1 \text{ OPT } P_2) \text{ OPT } P_3) \text{ OPT } P_4) \text{ AND } P_5 \]

where $?x, ?y, ?z$ are fresh variables and $a, b, c, d$ are fresh constants; and

\[ S = \text{var}(P) \cup \text{var}(Q) \]

\[ P_1 = (\text{VALUES } (?x) \{(a), (b)\}) \]

\[ P_2 = (P \text{ OPT } (\text{VALUES } (?x, ?y) \{(a, c)\})) \]

\[ P_3 = (Q \text{ OPT } (\text{VALUES } (?x, ?y) \{(b, c)\})) \]

\[ P_4 = (\text{VALUES } (?y) \{(d)\}) \]

\[ P_5 = (\text{VALUES } (?y) \{(c)\}) \]

4. **PRIMITIVITY OF OPERATORS**

An operator $\mathcal{X}$ is “primitive” if $\mathcal{X}$ cannot be expressible by other operators.

Clearly, based on the discussions in Section 3, the five operators, namely, AND, EXISTS, MINUS, OPT, and UNION, are not primitive since AND is not primitive [5].

In the rest of this paper, we will discuss the primitivity of the left six operators.

As we well known, the transition is not expressible in first-order logic and SPARQL 1.0 has the same expressivity of first-order logic [3]. It is clear that path property is not expressible in SPARQL1.0. Then property path is primitive since other newly added operators are still expressible in first-order logic [4].

To show the primitivity of BIND, we need a lemma.

**Lemma 3.** Let $P$ be a BIND-free pattern and $DS$ a dataset. For each mapping $\mu \in [P]_G$, the image of $\mu$ is in $\text{const}(DS) \cup \text{const}(P)$.

**Proposition 4.** BIND is primitive.

**Proof (Sketch).** Consider the following pattern:

\[ P = (?x, r, ?y) \text{ BIND CONCAT}(?x?) AS ?z \]

and the dataset $DS = \{(a, r, b)\}$ where $\{a, b, ab\} \cap \text{const}(Q) = \emptyset$. \[ \square \]

Let us now turn to the question of primitivity of FILTER. Analogously, we can conclude the following lemma [5].

**Lemma 5.** Let $DS = (G)$ where $G$ is the complete graph on two constants $a, b \in I$ (i.e., $G = \{a, b\} \times \{a, b\}$). Let $P$ be any FILTER-free pattern with $\{a, b\} \cap \text{const}(P) = \emptyset$. If there exists some mappings $\mu \in [P]_DS$ and $M \subseteq \text{dom}(\mu)$ such that $\mu(\mu(a, b)) \cap \text{const}(P) = \emptyset$. Then property path, $\forall x \in M$ and $\mu(\mu(a, b)) \not\subseteq \text{const}(P)$, and for all $?y \in \text{dom}(\mu) - M$ then for all mapping $\mu' : M \rightarrow \{a, b\}$, we can conclude that $\mu' \cup \mu |_{\text{dom}(\mu) - M} \in [P]_DS$.

**Proposition 6.** FILTER is primitive.

**Proof (Sketch).** Consider the following pattern:

\[ P = (?x, r, ?y) \text{ FILTER } \exists x = ?y \text{ and DS is the dataset from Lemma 5. In [5], by Lemma 5, we can conclude that } \mu = (?x \rightarrow a, ?y \rightarrow b, ?z \rightarrow a) \text{ is a mapping in } [P]_DS. \]

Therefore, we have arrived at a contradiction. \[ \square \]

**Proposition 7.** GRAPH is primitive.

**Proof (Sketch).** Consider the following pattern:

\[ P = \text{GRAPH}(?x, ?y, ?z) \text{ and the dataset } DS = (\emptyset, G) \]

where $G = \{(c, c, c)\}$ with $c \not\in \text{const}(Q)$. \[ \square \]

Finally, we will show that SELECT is primitive.

**Lemma 8.** Let $DS = (G)$ be a dataset. For any SELECT-free pattern $P$, if $\text{const}(P) \cap \text{const}(G) = \emptyset$ then $[P]_G$ does not contain the empty mapping $\mu_0$ (i.e., $\text{dom}(\mu_0) = \emptyset$).

**Proposition 9.** SELECT is primitive.

**Proof (Sketch).** Consider the following pattern:

\[ P = \text{SELECT}_\emptyset(?x, ?y, ?z) \text{ and a dataset } DS = (G) \]

where $G = \{(a, a, a)\}$ with $a \not\in \text{const}(Q)$. \[ \square \]

**Proposition 10.** VALUES is primitive.

**Proof (Sketch).** Consider the following pattern:

\[ P = (\text{VALUES } (?x), \{(c)\}) \]

and the empty dataset $DS_0$. \[ \square \]

Finally, we can conclude the most important result.

**Theorem 11.** Only operators BIND, FILTER, GRAPH, property path, SELECT, and VALUES are primitive.

**Acknowledgments**

This work is supported by the National Key Research and Development Program of China (2016YFB1000603), the National Natural Science Foundation of China (61502336), the Key Technology Research and Development Program of Tianjin (16YFZCGX00210), and the open funding project of Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education (K93-9-2016-05).

5. REFERENCES