Information Processing Letters 114 (2014) 480–485

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

On the primitivity of operators in SPARQL

Xiaowang Zhang *, Jan Van den Bussche

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

ARTICLE INFO

Article history:
Received 8 November 2013
Received in revised form 25 February 2014
Accepted 26 March 2014
Available online 12 April 2014
Communicated by Jef Wijsen

Keywords:
Databases
RDF
SPARQL
Primitive operator
Expressive power

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.

1. Introduction

Currently there is renewed interest in the classical topic of graph databases [2,21,11]. Much of this interest has been sparked by SPARQL: the query language for RDF. The Resource Description Framework (RDF) [16] is a popular data model for information on the Web. RDF represents information in the form of directed, labeled graphs. The standard query language for RDF data is SPARQL [20]. The current version 1.1 of SPARQL extends SPARQL 1.0 [19] 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 [1]). 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 [9], SQL [8], Datalog [1,15], and OWL [18]. Also the relationship between expressivity and complexity and optimization of evaluation has been studied [14,17,12].

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., [3,7,10]. It turns out that AND is not primitive: adapting an idea of Angles and Gutierrez [1], 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 [14]. 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 [5], a two-valued semantics has been considered as well [14]. Besides, we point out that the choice of semantics has no impact on the expressivity: the two semantics can express each other.

* Corresponding author.

http://dx.doi.org/10.1016/j.ipl.2014.03.014
0020-0190/

2. SPARQL

In this section we recall the syntax and semantics of SPARQL patterns, closely following the core SPARQL formalization given by Pérez et al. [14] and Arenas et al. [5].

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 $\text{SELECT}_S(P)$ 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
  $$\Omega_1 \bowtie \Omega = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1 \text{ and } \mu_2 \in \Omega_2 \text{ and } \mu_1 \sim \mu_2\}.$$
  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
  $$[\![P]\!]_G := \big([\![P_1]\!]_G \bowtie [\![P_2]\!]_G\big) \cup \big([\![P_1]\!]_G \setminus [\![P_2]\!]_G\big),$$
  where, for any two sets of mappings Ω_1 and Ω_2, we define
  $$\Omega_1 \setminus \Omega_2 = \{\mu_1 \in \Omega_1 \mid \neg\exists \mu_2 \in \Omega_2 : \mu_1 \sim \mu_2\}.$$
- 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:
  $$\mu\big(\text{bound}(?x)\big) = \begin{cases} \textit{true} & \text{if } ?x \in \text{dom}(\mu); \\ \textit{false} & \text{otherwise.} \end{cases}$$
  For an equality constraint $?x = ?y$, we define:
  $$\mu(?x = ?y) = \begin{cases} \textit{true} \\ \quad \text{if } ?x, ?y \in \text{dom}(\mu) \text{ and } \mu(?x) = \mu(?y); \\ \textit{false} \\ \quad \text{if } ?x, ?y \in \text{dom}(\mu) \text{ and } \mu(?x) \neq \mu(?y); \\ \textit{error} \\ \quad \text{otherwise.} \end{cases}$$
  Thus, when $?x$ and $?y$ do not both belong to $\text{dom}(\mu)$, 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,