Context-Free Path Queries on RDF Graphs

Xiaowang Zhang[1,2], Zhiyong Feng[1,2(✉)], Xin Wang[1,2], Guozheng Rao[1,2], and Wenrui Wu[1,2]

[1] School of Computer Science and Technology, Tianjin University, Tianjin, China
{xiaowangzhang,zyfeng,wangx,rgz,wenruiwu}@tju.edu.cn
[2] Tianjin Key Laboratory of Cognitive Computing and Application, Tianjin, China

Abstract. Navigational graph queries are an important class of queries that can extract implicit binary relations over the nodes of input graphs. Most of the navigational query languages used in the RDF community, e.g. property paths in W3C SPARQL 1.1 and nested regular expressions in nSPARQL, are based on the regular expressions. It is known that regular expressions have limited expressivity; for instance, some natural queries, like *same generation-queries*, are not expressible with regular expressions. To overcome this limitation, in this paper, we present cfSPARQL, an extension of SPARQL query language equipped with context-free grammars. The cfSPARQL language is strictly more expressive than property paths and nested expressions. The additional expressivity can be used for modelling graph similarities, graph summarization and ontology alignment. Despite the increasing expressivity, we show that cfSPARQL still enjoys a low computational complexity and can be evaluated efficiently.

1 Introduction

The Resource Description Framework (RDF) [30] recommended by World Wide Web Consortium (W3C) is a standard graph-oriented model for data interchange on the Web [6]. RDF has a broad range of applications in the semantic web, social network, bio-informatics, geographical data, etc. [1]. Typical access to graph-structured data is its navigational nature [12,16,21]. Navigational queries on graph databases return binary relations over the nodes of the graph [9]. Many existing navigational query languages for graphs are based on binary relational algebra such as XPath (a standard navigational query language for trees [25]) or regular expressions such as RPQ (regular path queries) [24].

SPARQL [32] recommended by W3C has become the standard language for querying RDF data since 2008 by inheriting classical relational languages such as SQL. However, SPARQL only provides limited navigational functionalities for RDF [28,37]. Recently, there are several proposed languages with navigational capabilities for querying RDF graphs [3–5,7,11,19,26,28,35]. Roughly, Versa [26] is the first language for RDF with navigational capabilities by using XPath over the XML serialization of RDF graphs. SPARQLeR proposed by Kochut et al. [19] extends SPARQL by allowing path variables. SPARQL2L proposed by Anyanwu et al. [7] allows path variables in graph patterns and offers

P. Groth et al. (Eds.): ISWC 2016, Part I, LNCS 9981, pp. 632–648, 2016.
DOI: 10.1007/978-3-319-46523-4_38

good features in nodes and edges such as constraints. PSPARQL proposed by Alkhateeb et al. [5] extends SPARQL by allowing regular expressions in general triple patterns with possibly blank nodes and CASPAR further proposed by Alkhateeb et al. [3,4] allows constraints over regular expressions in PSPARQL where variables are allowed in regular expressions. nSPARQL proposed by Pérez et al. [28] extends SPARQL by allowing nested regular expressions in triple patterns. Indeed, nSPARQL is still expressible in SPARQL if the transitive closure relation is absent [37]. In March 2013, SPARQL 1.1 [33] recommended by W3C allows property paths which strengthen the navigational capabilities of SPARQL1.0 [11,35].

However, those regular expression-based extensions of SPARQL are still limited in representing some more expressive navigational queries which are not expressed in regular expressions. Let us consider a fictional biomedical ontology mentioned in [31] (see Fig. 1). We are interested in a navigational query about those paths that confer similarity (e.g., between *Gene(B)* and *Gene(C)*), which suggests a causal relationship (e.g., between *Gene(S)* and *Phenotype(T)*). This query about similarity arises from the well-known *same generation-query* [2], which is proven to be inexpressible in any regular expression. To express the query, we have to introduce a query embedded with a context-free grammar (CFG) for expressing the language of $\{ww^T | w \textit{ is a string}\}$[31] where w^T is the converse of w. For instance, if $w =$ "$abcdfe$" then $w^T =$ "$e^{-1}f^{-1}d^{-1}c^{-1}b^{-1}a^{-1}$". As we know, CFG has more expressive power than any regular expression [18]. Moreover, the context-free grammars can provide a simplified more user-friendly dialect of Datalog [1] which still allows powerful recursion [18]. Besides, the context-free graph queries have also practical query evaluation strategies. For instance, there are some applications in verification [20]. So it is interesting to introduce a navigational query embedded with context-free grammars to express more practical queries like the same generation-query.

A proposal of conjunctive context-free path queries (written by *Helling's CCFPQ*) for edge-labeled directed graphs has been presented by Helling [14] by allowing context-free grammars in path queries. A naive idea to express same generation-queries is transforming this RDF graph to an edge-labeled directed graph via navigation axes [28] and then using Helling's CCFPQ since an RDF graph can be intuitively taken as an edge-labeled directed graph. However, this transformation is difficult to capture the full information of this RDF graph since there exist some slight differences between RDF graphs and edge-labeled directed graphs, particularly regarding the connectivity [13], thus it could not express some regular expression-based path queries on RDF graphs. For instance, a nested regular expression (nre) of the form $axis::[e]$ on RDF graphs in nSPARQL [28], is always evaluated to the empty set over any edge-labeled directed graph. That is to say, an nre of the form "$axis::[e]$" is hardly expressible in Helling's CCFPQ.

To represent more expressive queries with efficient query evaluation is a renewed interest topic in the classical topic of graph databases [2]. Hence, in this paper, we present a context-free extension of path queries and SPARQL

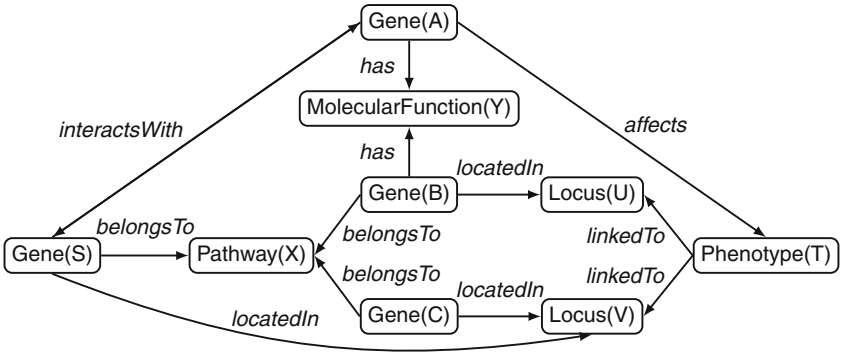

Fig. 1. A biomedical ontology [31]

queries on RDF graphs which can express both nre and nSPARQL [28]. Furthermore, we study several fundamental properties of the proposed context-free path queries and context-free SPARQL queries. The main contributions of this paper can be summarized as follows:

- We present *context-free path queries* (CFPQ) (including *conjunctive context-free path queries* (CCFPQ), *union of simple conjunctive context-free path queries* (UCCFPQ^s), and *union of conjunctive context-free path queries* (UCCFPQ) for RDF graphs and find that CFPQ, CCFPQ, and UCCFPQ have efficient query evaluation where the query evaluation has the polynomial data complexity and the NP-complete combined complexity. Finally, we implement our CFPQs and evaluate experiments on some popular ontologies.
- We discuss the expressiveness of CFPQs by referring to nested regular expressions (nre). We show that CFPQ, CCFPQ, UCCFPQ^s, and UCCFPQ exactly express four fragments of nre, basic nre "nre$_0$", union-free nre "nre$_0$(N)", nesting-free nre "nre$_0$(|)", and full nre, respectively (see Fig. 2). The query evaluation of cfSPARQL has the same complexity as SPARQL.
- We propose *context-free SPARQL* (cfSPARQL) and *union of conjunctive context-free SPARQL* (uccfSPARQL) based on CFPQ and UCCFPQ, respectively. It shows that cfSPARQL has the same expressiveness as that of uccfSPARQL. Furthermore, we prove that cfSPARQL can strictly express both SPARQL and nSPARQL (even nSPARQL$^\neg$: a variant of nSPARQL by allowing nre with negation "nre$^\neg$") (see Fig. 3).

Organization of the Paper. Section 2 recalls nSPARQL and context-free grammar. Section 3 defines CFPQ. Section 4 discusses the expressiveness of CFPQ. Section 5 presents cfSPARQL and Sect. 6 discusses the relations on nre with negation. Section 7 evaluates experiments. We conclude in Sect. 8. Due to the space limitation, all proofs and some further preliminaries are omitted but they are available in an extended technical report in arXiv.org [36].

2 Preliminaries

In this section, we introduce the language nSPARQL and context-free grammar.

2.1 The Syntax and Semantics of nSPARQL

In this subsection, we recall the syntax and semantics of nSPARQL, largely following the excellent expositions [27,28].

RDF Graphs. An RDF statement is a *subject-predicate-object* structure, called *RDF triple* which represents resources and the properties of those resources. For the sake of simplicity similar to [28], we assume that RDF data is composed only IRIs[1]. Formally, let U be an infinite set of *IRIs*. A triple $(s, p, o) \in U \times U \times U$ is called an *RDF triple*. An *RDF graph* G is a finite set of RDF triples. We use $adom(G)$ to denote the *active domain* of G, i.e., the set of all elements from U occurring in G.

For instance, a biomedical ontology shown in Fig. 1 can be modeled in an RDF graph named as G_{bio} where each labeled-edge of the form $a \xrightarrow{p} b$ is directly translated into a triple (a, p, b).

Paths and Traces. Let G be an RDF graph. A *path* $\pi = (c_1 c_2 \dots c_m)$ in G is a non-empty finite sequence of constants from G, where, for every $i \in \{1, \dots, m-1\}$, c_i and c_{i+1} exactly occur in the same triple of G (i.e., (c_i, c, c_{i+1}), (c_i, c_{i+1}, c), and (c, c_i, c_{i+1}) etc.). Note that the precedence between c_i and c_{i+1} in a path is independent of the positions of c_i, c_{i+1} in a triple.

In nSPARQL, three different *navigation axes*, namely, $next$, $edge$, and $node$, and their inverses, i.e., $next^{-1}$, $edge^{-1}$, and $node^{-1}$, are introduced to move through an RDF triple (s, p, o) [28].

Let $\Sigma = \{axis, axis{::}c | c \in U\}$ where $axis \in \{self, next, edge, node, next^{-1}, edge^{-1}, node^{-1}\}$. Let G be an RDF graph. We use $\Sigma(G)$ to denote the set of all symbols $\{axis, axis{::}c | c \in adom(G)\}$ occurring in G.

Let $\pi = (c_1 \dots c_m)$ be a path in G. A *trace* of path π is a string over $\Sigma(G)$ written by $\mathcal{T}(\pi) = l_1 \dots l_{m-1}$ where, for all $i \in \{1, \dots, m-1\}$, $(c_i c_{i+1})$ is labeled by l_i and l_i is of the form $axis$, $axis{::}c$, $axis^{-1}$, or $axis^{-1}{::}c$ [28]. We use $Trace(\pi)$ to denote the set of all traces of π.

Note that it is possible that a path has multiple traces since any two nodes possibly occur in the multiple triples. For example, consider an RDF graph $G = \{(a, b, c), (a, c, b)\}$ and given a path $\pi = (abc)$, both $(edge{::}c)(node{::}a)$ and $(next{::}c)(node^{-1}{::}a)$ are traces of π.

For instance, in the RDF graph G_{bio} (see Fig. 1), a path from *Gene(B)* to *Gene(C)* has a trace: $(next{::}locatedIn)(next^{-1}{::}linkedTo)(next{::}linkedTo)$ $(next^{-1}{::}locatedIn)$.

[1] A standard RDF data is composed of IRIs, blank nodes, and literals. For the purposes of this paper, the distinction between IRIs and literals will not be important.

Nested Regular Expressions. Nested regular expressions (*nre*) are defined by the following formal syntax:

$$e := axis | axis::c\,(c \in U) | axis::[e] | e/e | e|e | e^*.$$

Here the *nesting nre* is of the form $axis::[e]$.

For simplification, we denote some interesting fragments of nre as follows:

- nre_0: *basic nre*, i.e., nre only consisting of "*axis*", "/", and "$*$";
- $\text{nre}_0(|)$: basic nre by adding the operator "|";
- $\text{nre}_0(\text{N})$ to basic nre by adding nesting nre $axis::[e]$.

Patterns. Assume an infinite set V of *variables*, disjoint from U. A *nested regular expression triple* (or *nre-triple*) is a tuple of the form $(?x, e, ?y)$, where $?x, ?y \in V$ and e is an nre[2].

Formally, nSPARQL (graph) patterns are recursively constructed from nre-triples:

- An nre-triple is an nSPARQL pattern;
- All P_1 UNION P_2, P_1 AND P_2, and P_1 OPT P_2 are nSPARQL patterns if P_1 and P_2 are nSPARQL patterns;
- P FILTER C if P is an nSPARQL pattern and C is a constraint;
- $\text{SELECT}_S(P)$ if P is an nSPARQL pattern and S is a set of variables.

Semantics. Given an RDF graph G and an nre e, the evaluation of e on G, denoted by $[\![e]\!]_G$, is a binary relation. More details can be found in [28]. Here, we recall the semantics of nesting nre of the form $axis::[e]$ as follows:

$$[\![axis::[e]]\!]_G = \{(a,b) | \exists\, c, d \in adom(G),\ (a,b) \in [\![axis::c]\!]_G \text{ and } (c,d) \in [\![e]\!]_G\}.$$

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

Basically, the semantics of an nre-triple (u, e, v) is defined as follows:

$$[\![(u,e,v)]\!]_G = \{\mu\colon \{u,v\} \cap V \to U | (\mu(u), \mu(v)) \in [\![e]\!]_G\}.$$

Here, for any mapping μ and any constant $c \in U$, we agree that $\mu(c)$ equals c itself.

Let P be an nSPARQL pattern, the semantics of P on G, denoted by $[\![P]\!]_G$, is analogously defined as usual following the semantics of SPARQL [27,28].

Query Evaluation. A *SPARQL (SELECT) query* is an nSPARQL pattern. Given a RDF graph G, a pattern P, and a mapping μ, the query evaluation problem of nSPARQL is to decide whether μ is in $[\![P]\!]_G$. The complexity of query evaluation problem is PSpace-complete [27].

[2] In nSPARQL [28], nre-triples allow a general form (v, e, u) where $u, v \in U \cup V$. In this paper, we mainly consider the case $u, v \in V$ to simplify our discussion.

2.2 Context-Free Grammar

In this subsection, we recall context-free grammar. For more details, we refer the interested readers to some references about formal languages [18].

A *context-free grammar* (COG) is a 3-tuple $\mathcal{G} = (N, A, R)$ [3] where

- N is a finite set of variables (called *non-terminals*);
- A is a finite set of constants (called *terminals*);
- R is a finite set of production rules r of the form $v \to S$, where $v \in N$ and $S \in (N \cup A)^*$ (the asterisk $*$ represents the Kleene star operation). We write $v \to \epsilon$ if ϵ is the empty string.

A string over $N \cup A$ can be written to a new string over $N \cup A$ by applying production rules. Consider a string avb and a production rule $r : v \to avb$, we can obtain a new string $aavbb$ by applying this rule r one time and another new string $aaavbbb$ by applying the rule r twice. Analogously, strings with increasing length can be obtained in this rule.

Let $S, T \in (N \cup A)^*$. We write $(S \xrightarrow{\mathcal{G}} T)$ if T can be obtained from S by applying production rules of $\mathcal{G}$ within a finite number of times.

The *language* of grammar $\mathcal{G} = (N, A, R)$ w.r.t. start non-terminal $v \in N$ is defined by $\mathcal{L}(\mathcal{G}_v) = \{S \text{ a finite string over } A \mid v \xrightarrow{\mathcal{G}} S\}$.

For example, $\mathcal{G} = (N, A, R)$ where $N = \{v\}, A = \{a, b\}$, and $R = \{v \to ab, v \to avb\}$. Thus $\mathcal{L}(\mathcal{G}_v) = \{a^n b^n | n \geq 1\}$.

3 Context-Free Path Queries

In this section, we introduce context-free path queries on RDF graphs based on context-free path queries on directed graphs [14] and nested regular expressions [28].

3.1 Context-Free Path Queries and Their Extensions

In this subsection, we firstly define *conjunctive context-free path queries* on RDF graphs and then present some variants (it also can been seen as extensions).

Conjunctive Context-Free Path Queries. In this paper, we assume that $N \cap V = \emptyset$ and $A \subseteq \Sigma$ for all CFG $\mathcal{G} = (N, A, R)$.

Definition 1. *Let $\mathcal{G} = (N, A, R)$ be a CFG and m a positive integer. A* conjunctive context-free path query *(*CCFPQ*) is of the form $\mathbf{q}(?x, ?y)$[4], where,*

$$\mathbf{q}(?x, ?y) := \bigwedge_{i=1}^{m} \alpha_i, \tag{1}$$

where

[3] We deviate from the usual definition of context-free grammar by not including a special start non-terminal following [14].

[4] In this paper, we simply write a conjunctive query as a Datalog rule.

- α_i *is a triple pattern either of the form* $(?x, ?y, ?z)$ *or of the form* $v(?x, ?y)$*;*
- $\{?x, ?y\} \subseteq vars(\mathbf{q})$ *where* $vars(\mathbf{q})$ *denotes a collection of all variables occurring in the body of* $\mathbf{q}$*;*
- $\{v_1, \ldots, v_m\} \subseteq N$.

We regard the name of query $\mathbf{q}(?x, ?y)$ *as* $\mathbf{q}$ *and call the right of Eq. (1) as the* body *of* $\mathbf{q}$.

Remark 1. In our CCFPQ, we allow a triple pattern of the form $(?x, ?y, ?z)$ to characterize those queries w.r.t. ternary relationships such as nre-triple patterns of nSPARQL [28] to be discussed in Sect. 4. The formula $v(?x, ?y)$ is used to capture context-free path queries [14].

We say a *simple conjunctive context-free path query* (written by $CCFPQ^s$) if only the form $v(?x, ?y)$ is allowed in the body of a CCFPQ. We also say a *context-free path query* (written by *CFPQ*) if $m = 1$ in the body of a CCFPQ^s.

Semantically, let $\mathcal{G} = (N, A, R)$ be a CFG and G an RDF graph, given a CCFPQ $\mathbf{q}(?x, ?y)$ of the form (1), $[\![\mathbf{q}(?x, ?y)]\!]_G$ is defined as follows:

$$\{\mu|_{\{?x,?y\}} | \mathrm{dom}(\mu) = vars(\mathbf{q}) \text{ and } \forall\, i = 1, \ldots, m, \mu|_{vars(\alpha_i)} \in [\![\alpha_i]\!]_G\}, \qquad (2)$$

where the semantics of $v(?x, ?y)$ over G is defined as follows:

$$[\![v(?x, ?y)]\!]_G = \{\mu | \mathrm{dom}(\mu) = \{?x, ?y\} \text{ and } \exists\, \pi = (\mu(?x) c_1 \ldots c_m \mu(?y)) \text{ a path in } G, \mathit{Trace}(\pi) \cap \mathcal{L}(\mathcal{G}_v) \neq \emptyset\}.$$

Intuitively, $[\![v(?x, ?y)]\!]_G$ returns all pairs connected by a path in G which contains a trace belonging to the language generated from this CFG starting at non-terminal v.

Example 1. Let $\mathcal{G} = (N, A, R)$ be a CFG where $N = \{u, v\}, A = \{next::locatedIn, next^{-1}::locatedIn, next::linkedTo, next^{-1}::linkedTo\}$, and $P = \{v \to (next::locatedIn)\, u\, (next^{-1}::locatedIn), u \to (next^{-1}::linkedTo)\, u\, (next::linkedTo), u \to \epsilon\}$. Consider a CFPQ $\mathbf{q}$ be of the form $v(?x, ?y)$. The query $\mathbf{q}$ represents the relationship of similarity (between two genes) since $\mathcal{L}(\mathcal{G}_v) = \{(next^{-1}::locatedIn)^n (next^{-1}::linkedTo)(next::linkedTo)(next::locatedIn)^n | n \geq 1\}$. Consider the RDF graph G_{bio} in Fig. 1, $[\![\mathbf{q}(?x, ?y)]\!]_{G_{\mathrm{bio}}} = \{(?x = Gene(B), ?y = Gene(C))\}$. Clearly, the query $\mathbf{q}$ returns all pairs with similarity.

Query Evaluation. Let $\mathcal{G} = (N, A, R)$ be a CFG and G an RDF graph. Given a CCFPQ $\mathbf{q}(?x, ?y)$ and a tuple $\mu = (?x = a, ?y = b)$, the *query evaluation problem* is to decide whether $\mu \in [\![\mathbf{q}(?x, ?y)]\!]_G$, that is, whether the tuple μ is in the result of the query $\mathbf{q}$ on the RDF graph G. There are two kinds of computational complexity in the query evaluation problem [1,2]:

- the *data complexity* refers to the complexity w.r.t. the size of the RDF graph G, given a fixed query $\mathbf{q}$; and
- the *combined complexity* refers to the complexity w.r.t. the size of query $\mathbf{q}$ and the RDF graph G.

A CFG $\mathcal{G} = (N, A, R)$ is said to be in *norm form* if all of its production rules are of the form $v \to uw, v \to a$, or $v \to \epsilon$ where $v, u, w \in N$ and $a \in A$. Note that this norm form deviates from the usual *Chomsky Normal Form* [22] where the start non-terminals are absent. Indeed, every CFG is equivalent to a CFG in norm form, that is, for every CFG $\mathcal{G}$, there exists some CFG $\mathcal{G}'$ in norm form constructed from $\mathcal{G}$ in polynominal time such that $\mathcal{L}(\mathcal{G}_v) = \mathcal{L}(\mathcal{G}'_v)$ for every $v \in N$ [14].

Let G be an RDF graph and $\mathcal{G} = (N, A, R)$ a CFG. Given a non-terminal $v \in N$, let $\mathcal{R}_v(G)$ be *the context-free relation* of G w.r.t. v can be defined as follows:

$$\mathcal{R}_v(G) := \{(a, b) | \exists\, \pi = (ac_1 \dots c_m b) \text{ a path in } G, \mathit{Trace}(\pi) \cap \mathcal{L}(\mathcal{G}_v) \neq \emptyset\}. \quad (3)$$

Conveniently, the query evaluation of CCFPQ over an RDF graph can be reduced into the conjunctive first-order query over the context-free relations. Based on the conjunctive context-free recognizer for graphs presented in [14], we directly obtain a conjunctive context-free recognizer (see Algorithm 1) for RDF graphs by adding a convertor to transform an RDF graph into an edge-labeled directed graph (see Algorithm 2).

Algorithm 1. Conjunctive context-free recognizer for RDF

Input: G: an RDF graph; $\mathcal{G} = (N, A, R)$: a CFG in norm form; $v \in N$.
Output: $\{(v, a, b) | (a, b) \in \mathcal{R}_v(G)\}$

Θ:=$\{(v, a, a) | (a \in \mathit{adom}(G)) \wedge (v \to \epsilon \in P)\}$
  $\cup\{(v, a, b) | ((a, l, b) \in \mathit{Convertor}(G)) \wedge (v \to l) \in P\}$
$\Theta_{new} := \Theta$
while $\Theta_{new} \neq \emptyset$ **do**
  pick and remove a (v, a, b) from Θ_{new}
  for all $(u, a', a) \in \Theta$ **do**
    for all $v' \to uv \in R \wedge ((v', a', b) \notin \Theta)$ **do**
      Θ_{new}:=$\Theta_{new} \cup \{(v', a', b)\}$
      Θ:=$\Theta \cup \{(v', a', b)\}$
    end for
  end for
  for all $(u, b, b') \in \Theta$ **do**
    for all $u' \to vu \in R \wedge ((u', a, b') \notin \Theta)$ **do**
      Θ_{new}:=$\Theta_{new} \cup \{(u', a, b')\}$
      Θ:=$\Theta \cup \{(u', a, b')\}$
    end for
  end for
end while
return Θ

Given a path π and a context-free grammar $\mathcal{G}$, Algorithm 1 is sound and complete to decide whether the path π in RDF graphs has a trace generated from the grammar $\mathcal{G}$.

Algorithm 2. RDF convertor

Input: G: an RDF graph
Output: $Convertor(G) = (\mathcal{V}, \mathcal{E})$
1: $\mathcal{V}$:=$adom(G)$
2: $\mathcal{E}$:=$\{(c, self, c), (c, self::c, c) | c \in adom(G)\}$
3: $G_{new} := G$
4: **while** $G_{new} \neq \emptyset$ **do**
5: pick and remove a triple (s, p, o) from G_{new}
6: $\mathcal{E}$:=$\mathcal{E} \cup \{(s, next::p, o), (s, next, o), (o, next^{-1}::p, s), (o, next^{-1}, s),$
$(s, edge::o, p), (s, edge, p), (p, edge^{-1}::o, s), (p, edge^{-1}, s),$
$(p, node::s, o), (p, node, o), (o, node^{-1}::s, p), (o, node^{-1}, p)\}$
7: **end while**
8: **return** $Convertor(G)$

Proposition 1. *Let G be an RDF graph and $\mathcal{G} = (N, A, R)$ a CFG in norm form. For every $v \in N$, let Θ be the result computed in Algorithm 1, $(v, a, b) \in \Theta$ if and only if $(a, b) \in \mathcal{R}_v(G)$.*

Moreover, we can easily observe the worst-case complexity of Algorithm 1 since the complexity of Algorithm 2 is $\mathcal{O}(|G|)$.

Proposition 2. *Let G be an RDF graph and $\mathcal{G} = (N, A, R)$ a CFG. Algorithm 1 applied to G and $\mathcal{G}$ has a worst-case complexity of $\mathcal{O}((|N||G|)^3)$.*

As a result, we can conclude the following proposition.

Proposition 3. *The followings hold:*

1. *The query evaluation of CCFPQ has polynomial data complexity;*
2. *The query evaluation of CCFPQ has NP-complete combined complexity.*

Union of CCFPQ. An extension of CCFPQ capturing more expressive power such as disjunctive capability is introducing the union of CCFPQ. For instance, given a gene (e.g., *Gene(B)*) in the biomedical ontology (see Fig. 1), we wonder to find those genes which are relevant to this gene, that is, those genes either are similar to it (e.g., *Gene(C)*) or belong to the same pathway (e.g., *Gene(S)*).

A *union of conjunctive context-free path query* (UCCFPQ) is of the form

$$\mathbf{q}(?x, ?y) := \bigvee_{i=1}^{m} \mathbf{q}_i(?x, ?y), \tag{4}$$

where $\mathbf{q}_i(?x, ?y)$ is a CCFPQ for all $i = 1, \ldots, m$.

Analogously, we can define *union of simple conjunctive context-free path query* written by $UCCFPQ^s$.

Semantically, let G be an RDF graph, we define

$$[\![\mathbf{q}(?x, ?y)]\!]_G = \bigcup_{i=1}^{m} [\![\mathbf{q}_i(?x, ?y)]\!]_G, \tag{5}$$

where $[\![\mathbf{q}_i(?x, ?y)]\!]_G$ is defined as the semantics of CCFPQ for all $i = 1, \ldots, m$.

In Example 1, based on $\mathcal{G} = (N, A, R)$, we construct a CFG $\mathcal{G}' = (N', A', R')$ where $N' = N \cup \{s\}, A = A \cup \{next{::}belongsTo, next^{-1}{::}belongsTo\}$, and $R' = R \cup \{s \to (next{::}belongsTo)s(next^{-1}{::}belongsTo)\}$. Consider a UCCFPQ $\mathbf{q}(?x, ?y) := v(?x, ?y) \vee s(?x, ?y), [\![\mathbf{q}(?x, ?y)]\!]_{G_{\text{bio}}} = \{(?x = Gene(B), ?y = Gene(C)), (?x = Gene(B), ?y = Gene(S))\}$. That is, $[\![\mathbf{q}(?x, ?y)]\!]_{G_{\text{bio}}}$ returns all pairs where the first gene is relevant to the latter.

Note that the query evaluation of UCCFPQ has the same complexity as that of the evaluating of CCFPQ since we can simply evaluate a number (linear in the size of a UCCFPQ) of CCFPQs in isolation [2].

4 Expressivity of (U)(C)CFPQ

In this section, we investigate the expressivity of (U)(C)CFPQ by referring to nested regular expressions [28] and fragments of nre.

We discuss the relations between variants of UCCFPQ and variants of (nested) regular expressions and obtain the following results:

1. nre_0-triples can be expressed in CFPQ;
2. nre_0(N)-triples can be expressed in CCFPQ;
3. $\text{nre}_0(|)$-triples can be expressed in UCCFPQ^s;
4. nre-triples can be expressed in UCCFPQ.

1. nre_0 in CFPQ. The following proposition shows that CFPQ can express nre_0-triples.

Proposition 4. *For every nre_0-triple $(?x, e, ?y)$, there exist some CFG $\mathcal{G} = (N, A, R)$ and some CFPQ $\mathbf{q}(?x, ?y)$ such that for every RDF graph G, we have $[\![(?x, e, ?y)]\!]_G = [\![\mathbf{q}(?x, ?y)]\!]_G$.*

2. nre_0(N) in CCFPQ. Let $\mathcal{G}$ be a CFG. A *CCFPQ* $\mathbf{q}(?x, ?y)$ is in *nested norm form* if the following holds:

$$\mathbf{q}(?x, ?y) := ((?x', ?y', ?z') \wedge v(?x, ?y)) \wedge \mathbf{q}_1(?u, ?w), \tag{6}$$

where

- $\{?x, ?y\} \cap \{?x', ?y', ?z'\} \neq \emptyset$;
- $\{?x', ?y', ?z'\} \cap \{?u, ?w\} \neq \emptyset$;
- $\mathbf{q}_1(?u, ?w)$ is a CCFPQ.

Note that $(?x', ?y', ?z')$ is used to express a nested nre of the form $axis{::}[e]$ and $v(?x, ?y)$ is necessary to express a nested nre of the form $self{::}[e]$.

The following proposition shows that CCFPQ can express nre_0(N)-triples.

Proposition 5. *For every nre_0(N)-triple $(?x, e, ?y)$, there exist a CFG $\mathcal{G} = (N, A, R)$ and a CCFPQ $\mathbf{q}(?x, ?y)$ in nested norm form (6) such that for every RDF graph G, we have $[\![(?x, e, ?y)]\!]_G = [\![\mathbf{q}(?x, ?y)]\!]_G$.*

3. nre$_0(|)$ in UCCFPQ^s. Let e be an nre. We say e is in *union norm form* if e is of the following form $e_1|e_2|\ldots|e_m$ where e_i is an nre$_0$(N) for all $i = 1, \ldots, m$.

We can conclude that each nre-triple is equivalent to an nre in union norm form.

Proposition 6. *For every nre-triple $(?x, e, ?y)$, there exists some e' in union norm form such that $[\![(?x, e, ?y)]\!]_G = [\![(?x, e', ?y)]\!]_G$ for every RDF graph G.*

The following proposition shows that UCCFPQ^s can express nre$_0(|)$.

Proposition 7. *For every nre$_0(|)$-triple $(?x, e, ?y)$, there exists some CFG $\mathcal{G} = (N, A, R)$ and some UCCFPQ^s $\mathbf{q}(?x, ?y)$ in nested norm form such that for every RDF graph G, we have $[\![(?x, e, ?y)]\!]_G = [\![\mathbf{q}(?x, ?y)]\!]_G$.*

4. nre in UCCFPQ. By Propositions 5 and 7, we can conclude that

Proposition 8. *For every nre-triple $(?x, e, ?y)$, there exists some CFG $\mathcal{G} = (N, A, R)$ and some UCCFPQ $\mathbf{q}(?x, ?y)$ in nested norm form such that for every RDF graph G, we have $[\![(?x, e, ?y)]\!]_G = [\![\mathbf{q}(?x, ?y)]\!]_G$.*

However, those results above in this subsection are not vice versa since the context-free language is not expressible in any nre.

Proposition 9. *CFPQ is not expressible in any nre.*

5 Context-Free SPARQL

In this section, we introduce an extension language *context-free SPARQL* (for short, *cfSPARQL*) of SPARQL by using context-free triple patterns, plus SPARQL basic operators UNION, AND, OPT, FILTER, and SELECT and its expressiveness.

A *context-free triple pattern* (cftp) is of the form $(?x, \mathbf{q}, ?y)$ where $\mathbf{q}(?x, ?y)$ is a CFPQ. Analogously, we can define *union of conjunctive context-free triple pattern* (for short, *uccftp*) by using UCCFPQ.

cfSPARQL and Query Evaluation. Formally, cfSPARQL (graph) patterns are then recursively constructed from context-free triple patterns:

- A cftp is a cfSPARQL pattern;
- A triple pattern of the form $(?x, ?y, ?z)$ is a cfSPARQL pattern;
- All P_1 UNION P_2, P_1 AND P_2, and P_1 OPT P_2 are cfSPARQL patterns if P_1, P_2 are cfSPARQL patterns;
- P FILTER C if P is a cfSPARQL pattern and C is a contraint;
- SELECT$_S(P)$ if P is a cfSPARQL pattern and S is a set of variables.

Remark 2. In cfSPARQL, we allow triple patterns of form $(?x, ?y, ?z)$ (see Item 2), which can express any SPARQL triple pattern together with FILTER [38], to ensure that SPARQL is still expressible in cfSPARQL while SPARQL is not expressible in nSPARQL since any triple pattern $(?x, ?y, ?z)$ is not expressible in nSPARQL [28]. Our generalization of nSPARQL inherits the power of queries without more cost and maintains the coherence between CFPQ and "nested" nre of the form *axis*::[e]. Moreover, this extension in cfSPARQL coincides with our proposed CCFPQ where triple patterns of the form $(?x, ?y, ?z)$ are allowed.

Semantically, let P be a cfSPARQL pattern and G an RDF graph, $[\![(?x, \mathbf{q}, ?y)]\!]_G$ is defined as $[\![\mathbf{q}(?x, ?y)]\!]_G$ and other expressive cfSPARQL patterns are defined as normal [27,28].

Proposition 10. *SPARQL is expressible in cfSPARQL but not vice versa.*

A *cfSPARQL query* is a pattern.

We can define *union of conjunctive context-free SPARQL query* (for short, *uccfSPARQL*) by using uccftp in the analogous way.

At the end of this subsection, we discuss the complexity of evaluation problem of uccfSPARQL queries.

For a given RDF graph G, a uccftp P, and a mapping μ, the query evaluation problem is to decide whether μ is in $[\![P]\!]_G$.

Proposition 11. *The evaluation problem of uccfSPARQL queries has the same complexity as the evaluation problem of SPARQL queries.*

As a direct result of Proposition 8, we can conclude

Corollary 1. *nSPARQL is expressible in uccfSPARQL but not vice versa.*

On the Expressiveness of cfSPARQL. In this subsection, we show that cfSPARQL has the same expressiveness as uccfSPARQL. In other words, cfSPARQL is enough to express UCCFPQ on RDF graphs.

Since every cfSPARQL pattern is a uccfSPARQL pattern, we merely show that uccfSPARQL is expressible in cfSPARQL.

Proposition 12. *For every uccfSPARQL pattern P, there exists some cfSPARQL pattern Q such that $[\![P]\!]_G = [\![Q]\!]_G$ for any RDF graph G.*

6 Relations on (Nested) Regular Expressions with Negation

In this section, we discuss both the relation between UCCFPQ and nested regular expressions with negation and the relation between cfSPARQL and variants of nSPARQL.

Nested Regular Expressions with Negation. *A nested regular expression with negation (nre$^\neg$)* is an extension of nre by adding two new operators "difference $(e_1 - e_2)$" and "negation (e^c)" [37].

Semantically, let e, e_1, e_2 be three nre$^\neg$s and G an RDF graph,

- $[\![e_1 - e_2]\!]_G = \{(a, b) \in [\![e_1]\!]_G | (a, b) \notin [\![e_2]\!]_G\}$;
- $[\![e^c]\!]_G = \{(a, b) \in adom(G) \times adom(G) | (a, b) \notin [\![e]\!]_G\}$.

Analogously, an nre$^\neg$-triple pattern is of the form $(?x, e, ?y)$ where e is an nre$^\neg$. Clearly, nre$^\neg$-triple pattern is non-monotone.

Since nre is monotone, nre is strictly subsumed in nre$^\neg$ [37]. Though property paths in SPARQL 1.1 [29,33] are not expressible in nre since property paths allow

the negation of IRIs, property paths can be still expressible in the following subfragment of nre$^{\neg}$: let $c, c_1, \ldots, c_{n+m} \in U$,

$$e := next::c | e/e | self::[e] | e^* | e^+ | next^{-1}::[e] | (next::c_1 | \ldots | next::c_n | next^{-1}::c_{n+1} | \ldots | next^{-1}::c_{n+m})^c.$$

Note that e^+ can be expressible as the expression $e^* - self$.

Proposition 13. *uccftp is not expressible in any nre$^{\neg}$-triple pattern.*

Due to the non-monotonicity of nre$^{\neg}$, we have that nre$^{\neg}$ is beyond the expressiveness of any union of conjunctive context-free triple patterns even the star-free nre$^{\neg}$ (for short, sf-nre$^{\neg}$) where the Kleene star ($*$) is not allowed in nre$^{\neg}$.

Proposition 14. *sf-nre$^{\neg}$-triple pattern is not expressible in any uccftp.*

In short, nre$^{\neg}$-triple pattern and uccftp cannot express each other. Indeed, negation could make the evaluation problem hard even allowing a limited form of negation such as property paths [23].

cfSPARQL Can Express nSPARQL$^{\neg}$. Following nSPARQL, we can analogously construct the language nSPARQL$^{\neg}$ which is built on nre$^{\neg}$, by adding SPARQL operators UNION, AND, OPT, FILTER, and SELECT.

Though uccftps cannot express nre$^{\neg}$-triple patterns by Proposition 13, cfSPARQL can express nSPARQL$^{\neg}$ since nSPARQL$^{\neg}$ is still expressible in nSPARQL [37].

Corollary 2. *nSPARQL$^{\neg}$ is expressible in cfSPARQL.*

6.1 Overview

Finally, Figs. 2 and 3 provide the implication of the results on RDF graphs for the general relations between variants of CFPQ and nre and the general relations between cfSPARQL and nSPARQL where $\mathcal{L}_1 \rightarrow \mathcal{L}_2$ denotes that $\mathcal{L}_1$ is expressible in $\mathcal{L}_2$ and $\mathcal{L}_1 \leftrightarrow \mathcal{L}_2$ denotes that $\mathcal{L}_1 \rightarrow \mathcal{L}_2$ and $\mathcal{L}_2 \rightarrow \mathcal{L}_1$. Analogously, nSPARQL$^{\text{sf}}$ is an extension of SPARQL by allowing star-free nre$^{\neg}$-triple patterns.

7 Implementation and Evaluation

In this section, we have implemented the two algorithms for CFPQs without any optimization. Two context-free path queries over RDF graphs were evaluated and we found some results which cannot be captured by any regular expression-based path queries from RDF graphs.

The experiments were performed under Windows 7 on a Intel i5-760, 2.80GHz CPU system with 6GB memory. The program was written in Java 7 with maximum 2GB heap space allocated for JVM. Ten popular ontologies like *foaf*, *wine*, and *pizza* were used for testing.

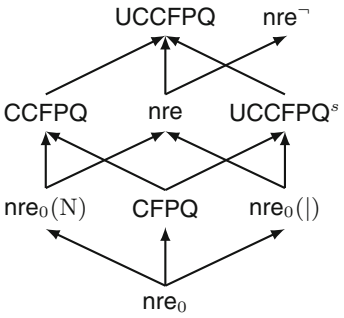

Fig. 2. Known relations between variants of CFPQ and variants of nre

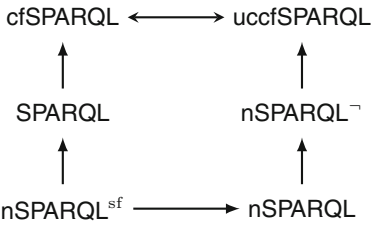

Fig. 3. Known relations between variants of cfSPARQL and variants of nSPARQL

Table 1. The evaluation results of Q_1 and Q_2

Ontology	# triples	Query 1		Query 2	
		Time (ms)	# results	Time (ms)	# results
protege	41	468	509	5	0
funding	144	499	296	125	77
skos	254	1044	810	16	1
foaf	454	5027	1929	1154	324
generation	319	6091	2164	13	0
univ-bench	306	20981	2540	532	228
travel	327	13971	2499	281	151
people+pets	703	82081	9472	247	120
biomedical-measure-primitive	459	420604	15156	1068851	9178
atom-primitive	561	515285	15454	4711499	13940
pizza	1980	3233587	56195	255853	4694
wine	2012	4075319	66572	273	79

Query 1. Consider a CFG $\mathcal{G}_1 = (N, A, R)$ where $N = \{S\}, A = \{next^{-1}{::}subClassOf, next{::}subClassOf, next^{-1}{::}type, next{::}type\}$, and $R = \{S \to (next^{-1}{::}subClassOf)\, S\, (next{::}subClassOf), S \to (next^{-1}{::}type)\, S\, (next{::}type), S \to \varepsilon\}$. The query Q_1 based on the grammar $\mathcal{G}_1$ can return all pairs of concepts or individuals at the same layer of the hierarchy of RDF graphs. Table 1 shows the experimental results of Q_1 over the testing ontologies. Note that $\#results$ denotes that number of pairs of concepts or individuals corresponding to Q_1.

Taking the ontology *foaf*, for example, the query Q_1 over *foaf* returns pairs of concepts like (*foaf:Document*, *foaf:Person*), which shows that the two concepts, *Document* and *Person*, are at the same layer of the hierarchy of *foaf*, where the top concept (*owl:Thing*) is at the first layer.

Query 2. Similarly, consider a CFG $\mathcal{G}_2 = (N, A, R)$ where $N = \{S, B\}, A = \{next^{-1}{::}subClassOf, next{::}subClassOf\}$, and $R = \{S \to BS, B \to (next{::}subClassOf)\, B\, (next^{-1}{::}subClassOf), B \to B(next^{-1}{::}subClassOf)\, B \to (next{::}subClassOf)(next^{-1}{::}subClassOf), S \to \varepsilon\}$. The query Q_2 based on the grammar $\mathcal{G}_2$ can return all pairs of concepts which are at adjacent two layers of the hierarchy of RDF graphs. We also take the ontology *foaf*, for example, the query Q_2 over *foaf* returns pairs of concepts like (*foaf:Person*, *foaf:PersonalProfileDocument*), which denotes that *Person* is at higher layer than *PersonalProfileDocument*, since *PersonalProfileDocument* is a subclass of *Document*. Table 1 shows the experimental results of Q_2 over the testing ontologies.

8 Conclusions

In this paper, we have proposed context-free path queries (including some variants) to navigate through an RDF graph and the context-free SPARQL query language for RDF built on context-free path queries by adding the standard SPARQL operators. Some investigation about some fundamental properties of those context-free path queries and their context-free SPARQL query languages has been presented. We proved that CFPQ, CCFPQ, UCCFPQ^s, and UCCFPQ strictly express basic nested regular expression (nre_0), $\text{nre}_0(\text{N})$, $\text{nre}_0(|)$, and nre, respectively. Moreover, uccfSPARQL has the same expressiveness as cfSPARQL; and both SPARQL and nSPARQL are expressible in cfSPARQL. Furthermore, we looked at the relationship between context-free path queries and nested regular expressions with negation (which can express property paths in SPARQL1.1) and the relationship between cfSPARQL queries and nSPARQL queries with negation (nSPARQL$^\neg$). We found that neither CFPQ nor nre$^\neg$ can express each other while nSPARQL$^\neg$ is still expressible in cfSPARQL. Finally, we discussed the query evaluation problem of CFPQ and cfSPARQL on RDF graphs. The query evaluation of UCCFPQ maintains the polynomial time data complexity and NP-complete combined complexity the same as conjunctive first-order queries and the query evaluation of cfSPARQL maintains the complexity as the same as SPARQL. These results provide a starting point for further research on expressiveness of navigational languages for RDF graphs and the relationships among regular path queries, nested regular path queries, and context-free path queries on RDF graphs.

There are a number of practical open problems. In this paper, we restrict that RDF data does not contain *blank nodes* as the same treatment in nSPARQL. We have to admit that blank nodes do make RDF data more expressive since a blank node in RDF is taken as an existentially quantified variable [17]. An interesting future work is to extend our proposed (U)(C)CFPQ for general RDF data with blank nodes by allowing path variables which are already valid in some extensions of SPARQL such as SPARQ2L [7], SPARQLeR [19], PSPARQL [5], and CPSPARQL [3,4], which are popular in querying general RDF data with blank nodes.

Acknowledgments. The authors thank Jelle Hellings and Guohui Xiao for their helpful and constructive comments. This work is supported by the program of the National Key Research and Development Program of China (2016YFB1000603) and the National Natural Science Foundation of China (NSFC) (61502336, 61373035). Xiaowang Zhang is supported by Tianjin Thousand Young Talents Program.

References

1. Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann, Burlington (2000)
2. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Boston (1995)
3. Alkhateeb, F., Baget, J.-F., Euzenat, J.: Constrained regular expressions in SPARQL. In: Proceedings of SWWS 2008, pp. 91–99 (2008)
4. Alkhateeb, F., Euzenat, J.: Constrained regular expressions for answering RDF-path queries modulo RDFS. Int. J. Web Inf. Syst. **10**(1), 24–50 (2014)
5. Alkhateeb, F., Baget, J.-F., Euzenat, J.: Extending SPARQL with regular expression patterns (for querying RDF). J. Web Semant. **7**(2), 57–73 (2009)
6. Angles, R., Gutierrez, C.: Survey of graph database models. ACM Comput. Surv. **40**(1), 1 (2008)
7. Anyanwu, K., Maduko, A., Sheth, A.P.: SPARQ2L: towards support for subgraph extraction queries in RDF databases. In: Proceedings of WWW 2007, pp. 797–806 (2007)
8. Arenas, M., Pérez, J., Gutierrez, C.: On the semantics of SPARQL. In: De Virgilio, R., Giunchiglia, F., Tanca, L. (eds.) Semantic Web Information Management - A Model-Based Perspective, pp. 281–307. Springer, Berlin (2009)
9. Barceló, P.: Querying graph databases. In: Proceedings of PODS 2013, pp. 175–188 (2013)
10. Bischof, S., Martin, C., Polleres, A., Schneider, P.: Collecting, integrating, enriching and republishing open city data as linked data. In: Arenas, M., et al. (eds.) ISWC 2015. LNCS, vol. 9367, pp. 57–75. Springer, Heidelberg (2015). doi:10.1007/978-3-319-25010-6_4
11. Fionda, V., Pirrò, G., Consens, M.P.: Extended property paths: writing more SPARQL queries in a succinct way. In: Proceedings of AAAI 2015, pp. 102–108 (2015)
12. Fletcher, G.H.L., Gyssens, M., Leinders, D., Surinx, D., den Bussche, J.V., Gucht, D.V., Vansummeren, S., Wu, Y.: Relative expressive power of navigational querying on graphs. Inf. Sci. **298**, 390–406 (2015)
13. Hayes, J., Gutierrez, C.: Bipartite graphs as intermediate model for RDF. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 47–61. Springer, Heidelberg (2004)
14. Hellings, J.: Conjunctive context-free path queries. In: Proceedings of ICDT 2014, pp. 119–130 (2014)
15. Hellings, J., Fletcher, G.H.L., Haverkort, H.J.: Efficient external-memory bisimulation on DAGs. In: Proceedings of SIGMOD 2012, pp. 553–564 (2012)
16. Hellings, J., Kuijpers, B., Van den Bussche, J., Zhang, X.: Walk logic as a framework for path query languages on graph databases. In: Proceedings of ICDT 2013, pp. 117–128 (2013)
17. Hogan, A., Arenas, M., Mallea, A., Polleres, A.: Everything you always wanted to know about blank nodes. J. Web Semant. **27**, 42–69 (2014)

18. Hopcroft, J., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Boston (1979)
19. Kochut, K.J., Janik, M.: SPARQLeR: extended SPARQL for semantic association discovery. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 145–159. Springer, Heidelberg (2007)
20. Lange, M.: Model checking propositional dynamic logic with all extras. J. Appl. Log. **4**(1), 39–49 (2006)
21. Libkin, L., Reutter, J.L., Vrgoc, D.: Trial for RDF: adapting graph query languages for RDF data. In Proceedings of PODS 2013, pp. 201–212 (2013)
22. Linz, P.: An Introduction to Formal Languages and Automata, 5th edn. Jones & Bartlett Publishers, Burlington (2012)
23. Losemann, K., Martens, W.: The complexity of regular expressions and property paths in SPARQL. ACM Trans. Database Syst. **38**(4), 24 (2013)
24. Reutter, J.L., Romero, M., Vardi, M.Y.: Regular queries on graph databases. In: Proceedings of ICDT 2015, pp. 177–194 (2015)
25. Marx, M., de Rijke, M.: Semantic characterizations of navigational XPath. SIGMOD Rec. **34**(2), 41–46 (2005)
26. Olson, M., Ogbuij, U.: The Versa Specification, October 2001
27. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. ACM Trans. Database Syst. **34**(3), 16 (2009)
28. Pérez, J., Arenas, M., Gutierrez, C.: nSPARQL: a navigational language for RDF. J. Web Semant. **8**(4), 255–270 (2010)
29. Polleres, A., Wallner, J.P.: On the relation between SPARQL1.1 and answer set programming. J. Appl. Non-Class. Log. **23**(1–2), 159–212 (2013)
30. RDF primer. W3C Recommendation, Febraury 2004
31. Sevon, P., Eronen, L.: Subgraph queries by context-free grammars. J. Integr. Bioinform. **5**(2), 100 (2008)
32. SPARQL query language for RDF. W3C Recommendation, January 2008
33. SPARQL 1.1 query language. W3C Recommendation, March 2013
34. Tian, Y., Hankins, R.A., Patel, J.M.: Efficient aggregation for graph summarization. In: Proceedings of SIGMOD 2008, pp. 567–580 (2008)
35. Kostylev, E.V., Reutter, J.L., Romero, M., Vrgoč, D.: SPARQL with property paths. In: Arenas, M., et al. (eds.) ISWC 2015. LNCS, vol. 9366, pp. 3–18. Springer, Berlin (2015)
36. Zhang, X., Feng, Z., Wang, X., Rao, G., Wu, W.: Context-free path queries on RDF graphs. Revised version (2016). arXiv:1506.00743
37. Zhang, X., den Bussche, J.V.: On the power of SPARQL in expressing navigational queries. Comput. J. **58**(11), 2841–2851 (2015)
38. Zhang, X., den Bussche, J.V., Picalausa, F.: On the satisfiability problem for SPARQL patterns. J. Artif. Intel. Res. **56**, 403–428 (2016)