# LDQL: A Language for Linked Data Queries

Olaf Hartig

http://olafhartig.de

**Abstract** In this paper, we propose LDQL, that is, a language to query Linked Data on the Web. The novelty of LDQL is that it enables a user to express separately (i) patterns that describe the expected query result, and (ii) Web navigation paths that select the data sources to be used for computing the result. As a downside of this expressiveness, we find that for some LDQL queries, a complete execution is not possible in practice. To address this issue, we show a syntactic property based on which systems can identify queries that do not have this limitation.

## 1 Introduction

In recent years an increasing amount of structured data has been published and interlinked on the World Wide Web (WWW) in adherence to the Linked Data principles [2,3]. These principles are based on standard Web technologies. In particular, (i) the Hypertext Transfer Protocol (HTTP) is used to access data, (ii) HTTP-based Uniform Resource Identifiers (URIs) are used as identifiers for entities described in the data, and (iii) the Resource Description Framework (RDF) is used as data model. Then, any HTTP URI in an RDF triple presents a *data link* that enables software clients to retrieve more data by looking up the URI with an HTTP request. The adoption of these principles has lead to the creation of a globally distributed dataspace: the *Web of Linked Data*.

From a graph database perspective the Web of Linked Data can be conceived of as two graphs of different types: First, the (virtual) union of all RDF triples in the Web of Linked Data represents an *RDF graph* [6]; this graph is distributed over the documents in the Web of Linked Data. Second, these documents constitute the nodes of a *link graph* whose edges represent the aforementioned data links that connect the documents.

The emergence of the Web of Linked Data makes possible an *online execution* of declarative queries over up-to-date data from a virtually unbounded set of data sources, each of which is readily accessible without any need for implementing source-specific APIs or wrappers. This possibility has spawned research interest in approaches to query Linked Data on the WWW as if it was a single (distributed) database. For an overview on query execution techniques proposed in this context refer to [12].

While there does not exist a standard language for expressing such queries, a few options have been proposed. In particular, a first strand of research focuses on extending the scope of SPARQL—which is the standard query language for centralized collections of RDF data [9]—such that an evaluation of SPARQL queries over Linked Data on the WWW has a well-defined semantics [4,10,11,14,16]. A second strand of research focuses on navigational languages (e.g., NautiLOD [8]). A commonality of all these proposals is that querying the aforementioned two graphs that comprise the Web of Linked Data (i.e., the link graph and the RDF data graph) is inherently entangled in each

of the proposed query formalisms. That is, for any query expressed in such a formalism, the definition of query-relevant regions of the link graph and the definition of query-relevant data from the RDF graph within the specified regions depend on one another.

In this paper, we study the alternative approach of avoiding such an inherent dependency. To this end, we propose a novel query language for Linked Data which we call LDQL. In contrast to the aforementioned query formalisms, LDQL separates query components for selecting regions of the link graph from components for specifying the query result that has to be constructed from the data in the selected regions. For the former, LDQL introduces the notion of a link path expression, which is a form of nested regular expression, and for the latter we use SPARQL. More precisely, the most basic type of LDQL queries consists of a link path expression and a SPARQL graph pattern. Additionally, such queries can be combined using conjunctions and disjunctions, where some subqueries may provide the starting points of navigation for other subqueries.

The downside of the expressiveness provided by LDQL are queries for which a complete execution is not possible in practice (because of the lack of a complete, up-to-date data catalog that is inherent to the WWW). To capture this issue formally, we define a notion of Web-safeness for LDQL queries. Then, the obvious question that arises is how to identify those LDQL queries that are Web-safe. Our contribution to answer this question is to show a sufficient syntactic condition for Web-safeness.

The paper is structured as follows: Section 2 introduces a data model that provides the basis for defining the semantics of LDQL formally. In Section 3 we define LDQL and show simple algebraic properties. Thereafter, in Section 4 we focus on Web-safeness. Section 5 concludes the paper and sketches future work. For proofs of the formal results in this paper we refer to the extended version of this paper [13].

## 2 Data Model

In this section we introduce a structural data model that captures the concept of a Web of Linked Data formally. As usual [4,8,10,11,14,16], for the definitions and analysis in this paper, we assume that the Web is fixed during the execution of any single query.

We use the RDF data model [6] as a basis for our model of a Web of Linked Data. That is, we assume three pairwise disjoint, infinite sets $\mathcal{U}$ (URIs), $\mathcal{B}$ (blank nodes), and $\mathcal{L}$ (literals). Then, an *RDF triple* is a tuple $\langle s, p, o\rangle \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$. For any RDF triple $t = \langle s, p, o\rangle$ we write $\mathrm{uris}(t)$ to denote the set of all URIs in $t$.

Additionally, we assume another infinite set $\mathcal{D}$ that is disjoint from $\mathcal{U}$, $\mathcal{B}$, and $\mathcal{L}$, respectively. We refer to elements in this set as *Linked Data documents*, or *documents* for short, and use them to represent the concept of Web documents from which Linked Data can be extracted. Hence, we assume a function, say $\mathrm{data}$, that maps each document $d \in \mathcal{D}$ to a finite set of RDF triples $\mathrm{data}(d) \subseteq (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$ such that the data of each document uses a unique set of blank nodes; i.e., for any pair of distinct documents $d, d' \in \mathcal{D}$, and any two RDF triples $t = \langle s, p, o\rangle$ and $t' = \langle s', p', o'\rangle$ with $t \in \mathrm{data}(d)$ and $t' \in \mathrm{data}(d')$, it holds that $\{s, p, o\} \cap \{s', p', o'\} \cap \mathcal{B} = \emptyset$.

Given these preliminaries, we are ready to define a *Web of Linked Data*.

**Definition 1.** A **Web of Linked Data** is a tuple $W = \langle D, adoc\rangle$ that consists of a set of documents $D \subset \mathcal{D}$ and a partial function $adoc\colon \mathcal{U} \to D$ that is surjective.

Function $adoc$ of a Web of Linked Data $W = \langle D, adoc \rangle$ captures the relationship between the URIs that can be looked up in this Web and the documents that can be retrieved by such lookups. Since not every URI can be looked up, the function is partial. For any URI $u \in \mathcal{U}$ with $u \in \mathrm{dom}(adoc)$ (i.e., any URI that can be looked up in $W$), document $d = adoc(u)$ can be considered the authoritative source of data for $u$ in $W$ (hence, the name $adoc$). To accommodate for documents that are authoritative for multiple URIs, we do not require injectivity for function $adoc$. However, we require surjectivity because we conceive documents as irrelevant for a Web of Linked Data if they cannot be retrieved by any URI lookup in this Web.

In this paper, we assume that the set of documents $D$ in any Web of Linked Data $W = \langle D, adoc \rangle$ is finite, in which case we say $W$ is *finite* [11]. Moreover, given a Web of Linked Data $W = \langle D, adoc \rangle$, we say that a URI $u \in \mathrm{uris}(t)$ in an RDF triple $t = \langle s, p, o \rangle$ establishes a *data link* to a document $d \in D$ if $adoc(u) = d$.

As a final concept, our data model formalizes the aforementioned notion of a link graph in which directed edges represent data links between documents. In the following formal definition of this graph, each edge is associated with a label that identifies both the particular RDF triple and the URI in this triple that establishes the corresponding data link. These labels shall provide the basis for defining the navigational component of our query language (cf. Section 3.1).

**Definition 2.** The **link graph** of a Web of Linked Data $W = \langle D, adoc \rangle$ is a directed, edge-labeled multigraph $\langle D, E \rangle$ whose vertices are all documents in $W$, and which has a directed edge $\langle d_{\mathsf{src}}, t, u, d_{\mathsf{tgt}} \rangle \in E$ from document $d_{\mathsf{src}} \in D$ to document $d_{\mathsf{tgt}} \in D$ if the data of $d_{\mathsf{src}}$ contains an RDF triple $t$ with a URI $u \in \mathrm{uris}(t)$ that establishes a data link to $d_{\mathsf{tgt}}$; the edge is labeled with both the triple $t$ and the URI $u$. Hence, the set of edges $E \subseteq D \times \mathcal{T} \times \mathcal{U} \times D$ is defined as follows:

$$E = \big\{ \langle d_{\mathsf{src}}, t, u, d_{\mathsf{tgt}} \rangle \,\big|\, t \in \mathrm{data}(d_{\mathsf{src}}) \text{ such that } u \in \mathrm{uris}(t) \text{ and } adoc(u) = d_{\mathsf{tgt}} \big\}.$$

*Example 1.* As a running example for this paper assume a simple Web of Linked Data $W_{\mathsf{ex}} = \langle D_{\mathsf{ex}}, adoc_{\mathsf{ex}} \rangle$ with three documents, $d_{\mathsf{A}}$, $d_{\mathsf{B}}$ and $d_{\mathsf{C}}$ (i.e., $D_{\mathsf{ex}} = \{d_{\mathsf{A}}, d_{\mathsf{B}}, d_{\mathsf{C}}\}$). The data in these documents are the following sets of RDF triples:

$$\begin{aligned} \mathrm{data}(d_{\mathsf{A}}) = \{&\langle u_{\mathsf{A}}, p_1, u_{\mathsf{B}} \rangle, & \mathrm{data}(d_{\mathsf{B}}) = \{&\langle u_{\mathsf{B}}, p_1, u_{\mathsf{C}} \rangle\}; \\ &\langle u_{\mathsf{B}}, p_2, u_{\mathsf{C}} \rangle\}; & \mathrm{data}(d_{\mathsf{C}}) = \{&\langle u_{\mathsf{A}}, p_2, u_{\mathsf{C}} \rangle\}; \end{aligned}$$

and function $adoc_{\mathsf{ex}}$ is given as follows: $adoc_{\mathsf{ex}}(u_{\mathsf{A}}) = d_{\mathsf{A}}$, $adoc_{\mathsf{ex}}(u_{\mathsf{B}}) = d_{\mathsf{B}}$, and $adoc_{\mathsf{ex}}(u_{\mathsf{C}}) = d_{\mathsf{C}}$ (i.e., $\mathrm{dom}(adoc_{\mathsf{ex}}) = \{u_{\mathsf{A}}, u_{\mathsf{B}}, u_{\mathsf{C}}\}$). This Web contains 8 data links. For instance, URI $u_{\mathsf{A}}$ in the RDF triple in $\mathrm{data}(d_{\mathsf{C}})$ establishes a data link to document $d_{\mathsf{A}}$. Hence, the corresponding edge in the link graph of $W_{\mathsf{ex}}$ is $\big\langle d_{\mathsf{C}}, \langle u_{\mathsf{A}}, p_2, u_{\mathsf{C}} \rangle, u_{\mathsf{A}}, d_{\mathsf{A}} \big\rangle$. Figure 1 illustrates this link graph with all 8 edges.

## 3 Definition of LDQL

This section defines our Linked Data query language, LDQL. LDQL queries are meant to be evaluated over a Web of Linked Data and each such query is built from two types

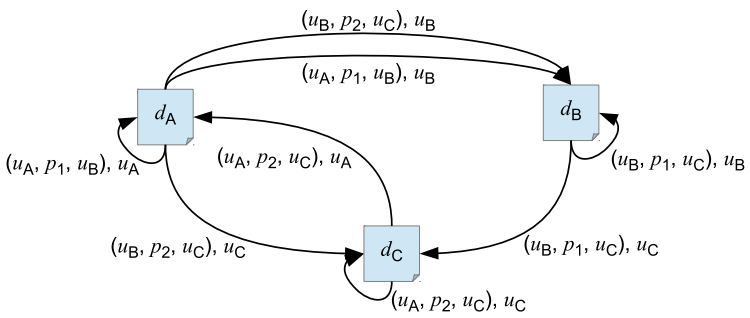


**Figure 1.** The link graph of our example Web of Linked Data $W_{\mathsf{ex}}$.

of components: First, there are link path expressions for selecting query-relevant documents of the queried Web of Linked Data; second, there are SPARQL graph patterns for specifying the query result that has to be constructed from the data in the selected documents. For this paper, we assume that the reader is familiar with the definition of SPARQL [9], including the algebraic formalization introduced by Pérez et al. [15].

In the following, we first formalize our notion of link path expressions. Thereafter, we define a syntax and semantics of LDQL queries and show simple algebraic properties of such queries. For brevity, we introduce LDQL also based on an algebraic syntax.

### 3.1 Link Path Expressions

Link Path Expressions (LPEs) are a form of nested regular expressions for navigation over the link graph of a Web of Linked Data. The base case for LPEs is to select single link graph edges in the context of a designated URI. To this end, we introduce *link patterns*, that is, tuples $\langle s, p, o\rangle \in \big(\mathcal{U} \cup \{\_, +\}\big) \times \big(\mathcal{U} \cup \{\_, +\}\big) \times \big(\mathcal{U} \cup \mathcal{L} \cup \{\_, +\}\big)$, where $\_$ is a special symbol that denotes a wildcard and $+$ is another special symbol that denotes a placeholder for the context URI. Then, a link graph edge $\langle d_{\mathsf{src}}, t, u, d_{\mathsf{tgt}}\rangle$ with $t = \langle x_1, x_2, x_3\rangle$ *matches* a link pattern $lp = \langle y_1, y_2, y_3\rangle$ in the context of a URI $u_{\mathsf{ctx}}$ if for each $i \in \{1, 2, 3\}$, any of the following three properties holds:

1. $y_i = \_$, or
2. $y_i = +$ and $x_i = u_{\mathsf{ctx}}$, or
3. $y_i = x_i$.

*Example 2.* Consider the link pattern $lp_{\mathsf{ex}} = \langle \_, p_2, \_\rangle$. The link graph of our example Web $W_{\mathsf{ex}}$ (cf. Example 1) contains two edges that match $lp_{\mathsf{ex}}$ in the context of URI $u_{\mathsf{A}}$, namely, the edges $\big\langle d_{\mathsf{A}}, \langle u_{\mathsf{B}}, p_2, u_{\mathsf{C}}\rangle, u_{\mathsf{B}}, d_{\mathsf{B}}\big\rangle$ and $\big\langle d_{\mathsf{A}}, \langle u_{\mathsf{B}}, p_2, u_{\mathsf{C}}\rangle, u_{\mathsf{C}}, d_{\mathsf{C}}\big\rangle$.

The rationale for adopting the notion of a designated context URI in our definition of link patterns is to support cases in which link graph navigation has to be focused solely on data links that are *authoritative*, where we call a data link authoritative if it is established by an RDF triple in a document $d_{\mathsf{src}}$ such that $d_{\mathsf{src}}$ is the authoritative document for some URI in the triple. Formally, in terms of link graph edges, an edge

$\langle d_{\mathsf{src}}, t, u, d_{\mathsf{tgt}} \rangle \in E$ in the link graph $\langle D, E \rangle$ of a Web of Linked Data $W = \langle D, adoc \rangle$ represents an authoritative data link if $adoc(u') = d_{\mathsf{src}}$ for some URI $u' \in \mathrm{uris}(t)$.

*Example 3.* Consider the link pattern $lp'_{\mathsf{ex}} = \langle \_, p_2, + \rangle$, which is a more restrictive variation of the link pattern discussed in the previous example. In contrast to $lp_{\mathsf{ex}}$, there does not exist any edge in the link graph of $W_{\mathsf{ex}}$ that matches $lp'_{\mathsf{ex}}$ in the context of URI $u_{\mathsf{A}}$. Any such edge would have to represent an authoritative data link; more precisely, the RDF triple of such a data link must have the context URI (i.e., $u_{\mathsf{A}}$) in the object position, which is not the case for the two edges that match the less restrictive link pattern $lp_{\mathsf{ex}}$. Notice also that, if we use URI $u_{\mathsf{C}}$ as context instead, there exists an edge that matches link pattern $lp'_{\mathsf{ex}}$ in the context of $u_{\mathsf{C}}$, namely $\big\langle d_{\mathsf{C}}, \langle u_{\mathsf{A}}, p_2, u_{\mathsf{C}} \rangle, u_{\mathsf{A}}, d_{\mathsf{A}} \big\rangle$.

Given the notion of a link pattern, we define LPEs as nested regular expressions over the alphabet that consists of all possible link patterns. Hence, an LPE is an expression defined by the following grammar (where $lp$ is an arbitrary link pattern):

$$lpe \;=\; \varepsilon \;\mid\; lp \;\mid\; lpe/lpe \;\mid\; lpe|lpe \;\mid\; lpe^* \;\mid\; [lpe]$$

Note that our notion of LPEs does not provide an operator for navigating paths in their inverse direction. The reason for omitting such an operator is that traversing arbitrary data links backwards is impossible on the WWW.

The semantics of LPEs is based on a designated context URI (similar to our notion of matching edges for link patterns). In particular, the semantics requires that each path of link graph edges that satisfies an LPE starts from the document that is authoritative for the given context URI. Then, the result of evaluating an LPE represents the end nodes of all such paths. More precisely, the result is a set of URIs where, informally, the lookup of each such URI returns the document that constitutes the end node of the corresponding path. The following definition captures the semantics of LPEs formally.

**Definition 3.** Let $lpe$ be an LPE, let $W = \langle D, adoc \rangle$ be a Web of Linked Data with link graph $\langle D, E \rangle$, and let $u_{\mathsf{ctx}}$ be a URI. The **$u_{\mathsf{ctx}}$-based evaluation** of $lpe$ over $W$, denoted by $[\![lpe]\!]_W^{u_{\mathsf{ctx}}}$, is a set of URIs that is empty if $u_{\mathsf{ctx}} \notin \mathrm{dom}(adoc)$; if $u_{\mathsf{ctx}} \in \mathrm{dom}(adoc)$, then the set is defined recursively as follows (where $d_{\mathsf{ctx}} = adoc(u_{\mathsf{ctx}})$, $lp$ is a link pattern, and $lpe$, $lpe_1$, $lpe_2$ are arbitrary LPEs):

$$\begin{aligned}
[\![\, \varepsilon \,]\!]_W^{u_{\mathsf{ctx}}} &= \{u_{\mathsf{ctx}}\} \\
[\![lp]\!]_W^{u_{\mathsf{ctx}}} &= \{u \mid \langle d_{\mathsf{ctx}}, t, u, d \rangle \in E \text{ that matches } lp \text{ in the context of } u_{\mathsf{ctx}}\} \\
[\![lpe_1/lpe_2]\!]_W^{u_{\mathsf{ctx}}} &= \{u \in [\![lpe_2]\!]_W^{u'} \mid u' \in [\![lpe_1]\!]_W^{u_{\mathsf{ctx}}}\} \\
[\![lpe_1|lpe_2]\!]_W^{u_{\mathsf{ctx}}} &= [\![lpe_1]\!]_W^{u_{\mathsf{ctx}}} \cup [\![lpe_2]\!]_W^{u_{\mathsf{ctx}}} \\
[\![lpe^*]\!]_W^{u_{\mathsf{ctx}}} &= \{u_{\mathsf{ctx}}\} \cup [\![lpe]\!]_W^{u_{\mathsf{ctx}}} \cup [\![lpe/lpe]\!]_W^{u_{\mathsf{ctx}}} \cup [\![lpe/lpe/lpe]\!]_W^{u_{\mathsf{ctx}}} \cup \ldots \\
[\![\, [lpe] \,]\!]_W^{u_{\mathsf{ctx}}} &= \{u_{\mathsf{ctx}} \mid [\![lpe]\!]_W^{u_{\mathsf{ctx}}} \neq \emptyset\}.
\end{aligned}$$

*Example 4.* Let $lpe_{\mathsf{ex}}$ be the LPE $\langle \_, p_2, \_ \rangle$ (i.e., the link pattern discussed in Example 2). For our example Web $W_{\mathsf{ex}}$ and context URI $u_{\mathsf{A}}$, we know by Example 2 that the link graph edges $\big\langle d_{\mathsf{A}}, \langle u_{\mathsf{B}}, p_2, u_{\mathsf{C}} \rangle, u_{\mathsf{B}}, d_{\mathsf{B}} \big\rangle$ and $\big\langle d_{\mathsf{A}}, \langle u_{\mathsf{B}}, p_2, u_{\mathsf{C}} \rangle, u_{\mathsf{C}}, d_{\mathsf{C}} \big\rangle$ match the pattern in the context of URI $u_{\mathsf{A}}$. Hence, the LPE selects documents $d_{\mathsf{B}} = adoc_{\mathsf{ex}}(u_{\mathsf{B}})$ and $d_{\mathsf{C}} = adoc_{\mathsf{ex}}(u_{\mathsf{C}})$. More precisely, we have $[\![lpe_{\mathsf{ex}}]\!]_{W_{\mathsf{ex}}}^{u_{\mathsf{A}}} = \{u_{\mathsf{B}}, u_{\mathsf{C}}\}$.

*Example 5.* As another example consider the slightly more complex LPE $lpe'_{\mathsf{ex}}$ which is of the form $\langle \_, p_1, \_ \rangle^* / [\langle \_, p_2, \_ \rangle]$. This LPE selects documents that can be reached via arbitrarily long paths of data links with predicate $p_1$ and, additionally, have some outgoing data link with predicate $p_2$. For our example Web $W_{\mathsf{ex}}$ and context URI $u_{\mathsf{A}}$, all three documents, $d_{\mathsf{A}}$, $d_{\mathsf{B}}$ and $d_{\mathsf{C}}$, can be reached via a $p_1$-path from URI $u_{\mathsf{A}}$, but only $d_{\mathsf{A}}$ and $d_{\mathsf{C}}$ pass the $p_2$-related test. Hence, we have $[\![lpe'_{\mathsf{ex}}]\!]^{u_{\mathsf{A}}}_{W_{\mathsf{ex}}} = \{u_{\mathsf{A}}, u_{\mathsf{C}}\}$.

While LPEs allows users to select documents from the queried Web of Linked Data, in the context of LDQL these documents are used to form an RDF dataset for evaluating a given SPARQL graph pattern. Formally, we specify this dataset as follows.

**Definition 4.** Let $u_{\mathsf{ctx}}$ be a URI, let $lpe$ be an LPE, and let $W = \langle D, adoc \rangle$ be a Web of Linked Data. The $u_{\mathsf{ctx}}$-$lpe$**-selected dataset** over $W$ is an RDF dataset $\mathfrak{D} = \langle G_0, \mathcal{N} \rangle$ (as per [9,1]) whose default graph $G_0$ is the union of all Named Graphs of the dataset, and that contains a Named Graph $\langle u, G \rangle \in \mathcal{N}$ for every URI $u \in [\![lpe]\!]^{u_{\mathsf{ctx}}}_W$ whose lookup (in $W$) results in retrieving a document; hence,

$$G_0 = \bigcup\nolimits_{\langle u,G\rangle \in \mathcal{N}} G, \text{ and}$$
$$\mathcal{N} = \{\langle u, G\rangle \mid u \in [\![lpe]\!]^{u_{\mathsf{ctx}}}_W \text{ and } u \in \mathrm{dom}(adoc) \text{ and } G = \mathrm{data}(adoc(u))\}.$$

*Example 6.* Consider context URI $u_{\mathsf{A}}$ and the aforementioned example LPE $lpe'_{\mathsf{ex}}$ for which we have $[\![lpe'_{\mathsf{ex}}]\!]^{u_{\mathsf{A}}}_{W_{\mathsf{ex}}} = \{u_{\mathsf{A}}, u_{\mathsf{C}}\}$ (cf. Example 5). Then, the $u_{\mathsf{A}}$-$lpe'_{\mathsf{ex}}$-selected dataset over $W_{\mathsf{ex}}$ is $\mathfrak{D}_{\mathsf{ex}} = \langle G_{\mathsf{ex}}, \mathcal{N}_{\mathsf{ex}} \rangle$ with $\mathcal{N}_{\mathsf{ex}} = \{\langle u_{\mathsf{A}}, \mathrm{data}(d_{\mathsf{A}})\rangle, \langle u_{\mathsf{C}}, \mathrm{data}(d_{\mathsf{C}})\rangle\}$ and, thus, $G_{\mathsf{ex}} = \{\langle u_{\mathsf{A}}, p_1, u_{\mathsf{B}}\rangle, \langle u_{\mathsf{B}}, p_2, u_{\mathsf{C}}\rangle, \langle u_{\mathsf{A}}, p_2, u_{\mathsf{C}}\rangle\}$ (cf. Example 1).

### 3.2 LDQL Queries

We are now ready to define LDQL queries formally.

**Definition 5.** An **LDQL query** is defined recursively as follows:

1. Any tuple $q = \langle u, lpe, P\rangle$ consisting of a URI $u$, an LPE $lpe$, and a SPARQL graph pattern $P$ is an LDQL query—hereafter, referred to as a *basic LDQL query*;
2. Any tuple $q = \langle ?v, lpe, P\rangle$ consisting of a variable $?v$, an LPE $lpe$, and a SPARQL graph pattern $P$ is an LDQL query;
3. If $q_1$ and $q_2$ are LDQL queries, then $(q_1 \,\mathsf{AND}\, q_2)$ and $(q_1 \,\mathsf{UNION}\, q_2)$ are LDQL queries.

Before introducing the formal semantics of LDQL queries, we provide some intuition about it. As per Definition 5, the most basic type of an LDQL query consists of a URI, an LPE, and a SPARQL graph pattern. Informally, the result of such a query $q = \langle u, lpe, P\rangle$ is the set of SPARQL solution mappings that can be obtained by evaluating the SPARQL pattern $P$ over the $u$-$lpe$-selected dataset.

*Example 7.* Consider a basic LDQL query $\langle u_{\mathsf{A}}, lpe'_{\mathsf{ex}}, B_{\mathsf{ex}}\rangle$ whose LPE is the aforementioned example LPE $lpe'_{\mathsf{ex}}$ (cf. Example 5) and whose SPARQL graph pattern is a basic graph pattern that contains two triple patterns, $B_{\mathsf{ex}} = \{\langle ?x, p_1, ?y\rangle, \langle ?x, p_2, ?z\rangle\}$. According to the semantics outlined above (and defined formally below), the result of

this query over our example Web $W_{\mathsf{ex}}$ consists of a single solution mapping, namely $\mu = \{?x \mapsto u_{\mathsf{A}}, ?y \mapsto u_{\mathsf{B}}, ?z \mapsto u_{\mathsf{C}}\}$. To see why we obtain this result, recall from Example 6 that the default graph of the $u_{\mathsf{A}}$-$lpe'_{\mathsf{ex}}$-selected dataset over $W_{\mathsf{ex}}$ is $G_{\mathsf{ex}} = \{\langle u_{\mathsf{A}}, p_1, u_{\mathsf{B}}\rangle, \langle u_{\mathsf{B}}, p_2, u_{\mathsf{C}}\rangle, \langle u_{\mathsf{A}}, p_2, u_{\mathsf{C}}\rangle\}$. Then, by the standard (set) semantics of SPARQL [1], the result of evaluating basic graph pattern $B_{\mathsf{ex}}$ over this graph $G_{\mathsf{ex}}$ is a singleton set that contains solution mapping $\mu$.

The other types of LDQL queries also have a set of solution mappings as their result: Operators AND and UNION represent conjunctions and disjunctions, respectively. The second base type of LDQL queries, tuples with a variable instead of a fixed (context) URI, is similar to basic LDQL queries with the difference that the variable can range over alternative context URIs; this type of query is meant to be used in conjunctions in which the other conjunct is used to select context URIs at query execution time (e.g., see query $q''_{\mathsf{ex}}$ in Example 8 below). The following definition formalizes the query semantics.

**Definition 6.** The **evaluation** of an LDQL query $q$ over a Web of Linked Data $W$, denoted by $[\![q]\!]_W$, is defined recursively as follows (where $u$ is a URI, $?v$ is a variable, $lpe$ is an LPE, $P$ is a SPARQL graph pattern, $[\![P]\!]^{\mathfrak{D}}_{G_0}$ denotes the evaluation of $P$ over an RDF graph $G_0$ in an RDF dataset $\mathfrak{D} = \langle G_0, \mathcal{N}\rangle$ [1, Definition 13.3], and $q_1$ and $q_2$ are arbitrary LDQL queries):

$$
\begin{aligned}
[\![\langle u, lpe, P\rangle]\!]_W &= [\![P]\!]^{\mathfrak{D}}_{G_0} \qquad (\ \mathfrak{D} = \langle G_0, \mathcal{N}\rangle \text{ is the } u\text{-}lpe\text{-selected dataset over } W),\\
[\![\langle ?v, lpe, P\rangle]\!]_W &= \textstyle\bigcup_{u \in \mathcal{U}} \big([\![\langle u, lpe, P\rangle]\!]_W \bowtie \{\mu_u\}\big) \qquad (\text{ where } \mu_u = \{?v \mapsto u\}\ ),\\
[\![(q_1 \ \mathsf{AND}\ q_2)]\!]_W &= [\![q_1]\!]_W \bowtie [\![q_2]\!]_W,\\
[\![(q_1 \ \mathsf{UNION}\ q_2)]\!]_W &= [\![q_1]\!]_W \cup [\![q_2]\!]_W.
\end{aligned}
$$

*Example 8.* Consider an LDQL query $q_{\mathsf{ex}} = \big\langle ?x, \varepsilon, \langle ?x, p_1, ?z\rangle\big\rangle$, which is of the second base type in Definition 5 (with the SPARQL graph pattern being a single triple pattern). Additionally, let $q'_{\mathsf{ex}} = \big\langle u_{\mathsf{A}}, lpe'_{\mathsf{ex}}, \{\langle ?x, p_1, ?y\rangle, \langle ?x, p_2, ?z\rangle\}\big\rangle$ be the basic LDQL query introduced in Example 7, and let $q''_{\mathsf{ex}}$ be the conjunction of these two queries; i.e., $q''_{\mathsf{ex}} = (q_{\mathsf{ex}} \ \mathsf{AND}\ q'_{\mathsf{ex}})$. By Example 7 we know that $[\![q'_{\mathsf{ex}}]\!]_{W_{\mathsf{ex}}} = \{\mu\}$ (with solution mapping $\mu$ as given in Example 7). Furthermore, based on the data given in Example 1, it is easy to see that $[\![q_{\mathsf{ex}}]\!]_{W_{\mathsf{ex}}} = \{\mu_1, \mu_2\}$ with $\mu_1 = \{?x \mapsto u_{\mathsf{A}}, ?z \mapsto u_{\mathsf{B}}\}$ and $\mu_2 = \{?x \mapsto u_{\mathsf{B}}, ?z \mapsto u_{\mathsf{C}}\}$. For the evaluation of query $q''_{\mathsf{ex}}$ over $W_{\mathsf{ex}}$, the result sets $[\![q_{\mathsf{ex}}]\!]_{W_{\mathsf{ex}}}$ and $[\![q'_{\mathsf{ex}}]\!]_{W_{\mathsf{ex}}}$ have to be joined. While for $\mu_1 \in [\![q_{\mathsf{ex}}]\!]_{W_{\mathsf{ex}}}$, there does not exist a compatible join candidate in $[\![q'_{\mathsf{ex}}]\!]_{W_{\mathsf{ex}}}$, solution mapping $\mu_2 \in [\![q_{\mathsf{ex}}]\!]_{W_{\mathsf{ex}}}$ is compatible with $\mu \in [\![q'_{\mathsf{ex}}]\!]_{W_{\mathsf{ex}}}$. Consequently, we have $[\![q''_{\mathsf{ex}}]\!]_{W_{\mathsf{ex}}} = \{\mu'\}$ with $\mu' = \mu \cup \mu_2$.

### 3.3 Algebraic Properties of LDQL Queries

As a basis for the discussion in the next section, we show simple algebraic properties.

**Lemma 1.** *The operators* AND *and* UNION *are associative and commutative, respectively, and the operator* AND *distributes over* UNION*. That is, if $q_1$, $q_2$, and $q_3$ are LDQL queries, then the following semantic equivalences hold:*

- $(q_1 \ \mathsf{AND}\ q_2) \equiv (q_2 \ \mathsf{AND}\ q_1)$;

- $(q_1 \text{ UNION } q_2) \equiv (q_2 \text{ UNION } q_1)$;
- $\big(q_1 \text{ AND } (q_2 \text{ AND } q_3)\big) \equiv \big((q_1 \text{ AND } q_2) \text{ AND } q_3\big)$;
- $\big(q_1 \text{ UNION } (q_2 \text{ UNION } q_3)\big) \equiv \big((q_1 \text{ UNION } q_2) \text{ UNION } q_3\big)$;
- $\big(q_1 \text{ AND } (q_2 \text{ UNION } q_3)\big) \equiv \big((q_1 \text{ AND } q_2) \text{ UNION } (q_1 \text{ AND } q_3)\big)$.

*Proof.* Since the definition of LDQL operators AND and UNION is equivalent to their SPARQL counterparts, the semantic equivalences in Lemma 1 follow from corresponding equivalences for SPARQL graph patterns as shown by Pérez et al. [15, Lemma 2.5].

Lemma 1 allows us to write sequences of either AND or UNION without parentheses. Hereafter, we assume that every UNION-*free LDQL query* is represented as an LDQL query of the form $(q_1 \text{ AND } q_2 \text{ AND } \ldots \text{ AND } q_m)$ where each subquery $q_i$ $(1 \leq i \leq m)$ is either of the form $\langle u, lpe, P\rangle$ (i.e., a basic LDQL query) or of the form $\langle ?v, lpe, P\rangle$. Moreover, we say that an LDQL query is in UNION *normal form* if it is of the form $(q_1 \text{ UNION } q_2 \text{ UNION } \ldots \text{ UNION } q_n)$ such that each subquery $q_i$ $(1 \leq i \leq n)$ is a UNION-free LDQL query. The following result is an immediate consequence of Lemma 1.

**Corollary 1.** *For every LDQL query, there exists a semantically equivalent LDQL query that is in* UNION *normal form.*

## 4 Web-Safeness of LDQL Queries

In this section we study the "Web-safeness" of LDQL queries, where, informally, we call a query *Web-safe* if a complete execution of the query over the WWW is possible in practice (which is not the case for all LDQL queries as we shall see).

To provide a more formal definition of this notion of Web-safeness we make the following observations. While the mathematical structures introduced by our data model capture the notion of Linked Data on the WWW formally (and, thus, allow us to provide a formal semantics for LDQL queries), in practice, these structures are not available completely for the WWW. For instance, given that an infinite number of strings can be used as HTTP URIs [7], we cannot assume complete information about which URIs are in the domain of the partial function $adoc$ (i.e., can be looked up to retrieve some document) and which are not; in fact, disclosing this information would require a process that systematically tries to look up every possible HTTP URI and, thus, would never terminate because of the aforementioned infiniteness. Therefore, it is also impossible to guarantee the discovery of every document in the set $D$ (without looking up an infinite number of URIs). Consequently, any query whose execution requires a complete enumeration of this set is not feasible in practice. Based on these observations, we define *Web-safeness* of LDQL queries as follows.

**Definition 7.** An LDQL query $q$ is **Web-safe** if there exists an algorithm that, for any finite Web of Linked Data $W = \langle D, adoc\rangle$, computes $[\![q]\!]_W$ by looking up only a finite number of URIs without assuming an a priori availability of any information about the sets $D$ and $\mathrm{dom}(adoc)$.

*Example 9.* Recall our example queries $q_{\mathsf{ex}}$, $q'_{\mathsf{ex}}$, and $q''_{\mathsf{ex}}$ (cf. Example 8). For query $q_{\mathsf{ex}} = \langle ?x, \varepsilon, \langle ?x, p_1, ?z\rangle\rangle$, any URI $u \in \mathcal{U}$ may be used to obtain a nonempty subset of the query result as long as a lookup of $u$ retrieves a document whose data includes RDF triples that match $\langle u, p_1, ?z\rangle$. Therefore, without access to $D$ or $\mathrm{dom}(adoc)$ of the queried Web $W = \langle D, adoc\rangle$, the completeness of the computed query result can be guaranteed only by checking each of the infinitely many possible HTTP URIs. Hence, query $q_{\mathsf{ex}}$ is *not* Web-safe. In contrast, although it contains $q_{\mathsf{ex}}$ as a subquery, query $q''_{\mathsf{ex}} = (q_{\mathsf{ex}}\ \mathsf{AND}\ q'_{\mathsf{ex}})$ is Web-safe, and so is $q'_{\mathsf{ex}} = \langle u_{\mathsf{A}}, lpe'_{\mathsf{ex}}, B_{\mathsf{ex}}\rangle$. A possible execution algorithm for $q'_{\mathsf{ex}}$ may first compute $[\![lpe'_{\mathsf{ex}}]\!]_W^{u_{\mathsf{A}}}$ by traversing the queried Web $W$ based on the given LPE $lpe'_{\mathsf{ex}}$. Thereafter, the algorithm retrieves documents by looking up all URIs $u \in [\![lpe'_{\mathsf{ex}}]\!]_W^{u_{\mathsf{A}}}$ (or simply keeps these documents after the traversal); and, finally, the algorithm evaluates pattern $B_{\mathsf{ex}}$ over the union of the RDF triples in the retrieved documents. If $W$ is finite (i.e., contains a finite number of documents) the traversal process requires a finite number of URI lookups only, and so does the retrieval of documents in the second step; the final step does not look up any URI. To see that $q''_{\mathsf{ex}}$ is also Web-safe we note that after executing subquery $q'_{\mathsf{ex}}$ (e.g., by using the algorithm as outlined before), the execution of the other (non-Web-safe) subquery $q_{\mathsf{ex}}$ can be reduced to a finite number of URI lookups, namely the URIs bound to variable $?x$ in solution mappings obtained for subquery $q'_{\mathsf{ex}}$. Although any other URI may also be used to obtain solution mappings for $q_{\mathsf{ex}}$, such solution mappings cannot be joined with any of the solution mappings for $q'_{\mathsf{ex}}$ and, thus, are irrelevant for the result of $q''_{\mathsf{ex}}$.

The example illustrates that there exists an LDQL query that is not Web-safe. In fact, it is not difficult to see that the argument for the non-Web-safeness of query $q_{\mathsf{ex}}$ as made in the example can be applied to any LDQL query of the form $\langle ?v, lpe, P\rangle$; that is, none of these queries is Web-safe. However, the example also shows that more complex queries that contain such non-Web-safe subqueries may still be Web-safe. Therefore, we now introduce properties to identify LDQL queries that are Web-safe (even if some of their subqueries are not). As a basis for the discussion, we first observe that the Web-safeness of an LDQL query carries over to any semantically equivalent LDQL query.

**Fact 1.** An LDQL query $q$ is Web-safe if there exists an LDQL query $q'$ such that $q$ and $q'$ are semantically equivalent and $q'$ is Web-safe.

In conjunction with Corollary 1, Fact 1 allows us to focus our discussion on LDQL queries in UNION normal form without losing generality. It is easy to show that such queries are Web-safe if all of their (top-level) subqueries are Web-safe:

**Proposition 1.** *An LDQL query of the form* $(q_1\ \mathsf{UNION}\ q_2\ \mathsf{UNION}\ \ldots\ \mathsf{UNION}\ q_n)$ *is Web-safe if each subquery* $q_i$ $(1 \leq i \leq n)$ *is Web-safe.*

*Proof.* Assume each subquery $q_i$ is Web-safe. Hence, for each $q_i$, there exists an algorithm that satisfies the conditions in Definition 7. Then, the Web-safeness of LDQL query $(q_1\ \mathsf{UNION}\ q_2\ \mathsf{UNION}\ \ldots\ \mathsf{UNION}\ q_n)$ is easily shown by specifying another algorithm that calls the algorithms of the subqueries sequentially and unions their results.

By Proposition 1, we can show that an LDQL query in UNION normal form is Web-safe by showing that each of its UNION-free subqueries is Web-safe. Therefore, in the remainder of this section we focus the Web-safeness of UNION-free LDQL queries.

Our discussion of the (UNION-free) query $q''_{\mathsf{ex}} = (q_{\mathsf{ex}}$ AND $q'_{\mathsf{ex}})$ in Example 9 suggests that UNION -free LDQL queries can be shown to be Web-safe if it is possible to execute any non-Web-safe subquery by using variable bindings obtained from other subqueries. A necessary condition for such an execution strategy is that the variable in question (i.e., variable $?x$ in the case of non-Web-safe subquery $q_{\mathsf{ex}} = \langle ?x, \varepsilon, \langle ?x, p_1, ?z\rangle\rangle$) is guaranteed to be bound in every possible solution mapping obtained from the other subqueries.

To allow for an automated verification of this condition we adopt Buil-Aranda et al.'s notion of strongly bound variables [5].[1] To this end, for any SPARQL graph pattern $P$, let $\mathrm{sbvars}(P)$ denote the set of strongly bound variables in $P$ as defined by Buil-Aranda et al. [5]. For the sake of space, we do not repeat the definition here. However, we emphasize that $\mathrm{sbvars}(P)$ can be constructed recursively, and each variable in $\mathrm{sbvars}(P)$ is guaranteed to be bound in every possible solution for $P$ [5, Proposition 1]. To carry over these properties to LDQL queries, we use the notion of strongly bound variables in SPARQL patterns to define the following notion of strongly bound variables in LDQL queries; thereafter, in Lemma 2, we show the desired boundedness guarantee.

**Definition 8.** The set of **strongly bound variables** in a LDQL query $q$, denoted by $\mathrm{sbvars}(q)$, is defined recursively as follows:

1. If $q$ is of the form $\langle u, lpe, P\rangle$, then $\mathrm{sbvars}(q) = \mathrm{sbvars}(P)$.
2. If $q$ is of the form $\langle ?v, lpe, P\rangle$, then $\mathrm{sbvars}(q) = \mathrm{sbvars}(P) \cup \{?v\}$.
3. If $q$ is of the form $(q_1$ AND $q_2)$, then $\mathrm{sbvars}(q) = \mathrm{sbvars}(q_1) \cup \mathrm{sbvars}(q_2)$.
4. If $q$ is of the form $(q_1$ UNION $q_2)$, then $\mathrm{sbvars}(q) = \mathrm{sbvars}(q_1) \cap \mathrm{sbvars}(q_2)$.

**Lemma 2.** *Let $q$ be an LDQL query. For every Web of Linked Data $W$ and every solution mapping $\mu \in [\![q]\!]_W$, it holds that* $\mathrm{sbvars}(q) \subseteq \mathrm{dom}(\mu)$.

*Proof.* Lemma 2 follows trivially from Definition 8 and [5, Proposition 1].

We are now ready to show the following result.

**Theorem 1.** *A* UNION*-free LDQL query $(q_1$ AND $q_2$ AND ... AND $q_m)$ is Web-safe if there exists a total order $\prec$ over the set of subqueries $\{q_1, q_2, \ldots, q_m\}$ such that for each subquery $q_i$ ($1 \leq i \leq m$), it holds that either (i) $q_i$ is a basic LDQL query or (ii) $q_i$ is of the form $\langle ?v, lpe, P\rangle$ and $?v \in \bigcup_{q_j \prec q_i} \mathrm{sbvars}(q_j)$.*

*Proof (Sketch).* We prove Theorem 1 based on an iterative algorithm that generalizes the execution strategy outlined for query $q''_{\mathsf{ex}}$ in Example 9. That is, the algorithm executes the subqueries $q_1, q_2, \ldots, q_m$ sequentially in the order $\prec$ such that each iteration step executes one of the subqueries by using the solution mappings computed during the previous step. By the conditions in Theorem 1, the first subquery (according to $\prec$)

[1] While we may also adopt Buil-Aranda et al.'s notion of bound variables (not to be confused with their notion of strongly bound variables), a definition of bound variables in LDQL queries would be based directly on the boundedness of variables in SPARQL patterns. Then, it is not difficult to see that the undecidability of verifying whether a given variable is bound in a given SPARQL pattern [5] would also carry over to LDQL queries. Due to space limitations, we omit discussing boundedness and use directly the decidable alternative (i.e., strong boundedness).

cannot be of the form $\langle ?v, lpe, P\rangle$; hence, the first step of the iteration is guaranteed to execute a basic LDQL query. This execution resembles the execution strategy as outlined for the (basic) query $q'_{\mathsf{ex}}$ in Example 9. For any subsequent iteration step, if the subquery executed by that step is of the form $\langle ?v, lpe, P\rangle$, then the corresponding condition in Theorem 1 (in conjunction with Lemma 2) guarantees that $?v \in \mathrm{dom}(\mu)$ for every solution mapping $\mu$ obtained from the previous step. These mappings are finitely many (for a finite Web of Linked Data). Then, the algorithm uses the URIs bound to variable $?v$ in these mappings as the only possible context URIs for the execution of the subquery (instead of using all URIs), which is sufficient because solution mappings computed for the subquery by using some other context URI would not be join compatible with any of the solution mappings obtained from the previous step. For a full formal proof and the complete algorithm we refer to the extended version of this paper [13].

To recapitulate, we summarize our proposed procedure to test a given LDQL query $q$ for Web-safeness based on our results in this paper: First, by using the algebraic properties in Lemma 1, the query has to be rewritten into a semantically equivalent LDQL query $q_{\mathsf{nf}} = (q_1 \text{ UNION } q_2 \text{ UNION } \ldots \text{ UNION } q_n)$ that is in UNION normal form, which is possible for any LDQL query (cf. Corollary 1). Thereafter, the following Web-safeness test has to repeated for every subquery $q_i$ $(1 \leq i \leq n)$; recall that each of these subqueries is a UNION-free LDQL query $q_i = (q_1^i \text{ AND } q_2^i \text{ AND } \ldots \text{ AND } q_{m_i}^i)$. The test is to find an order for their subqueries $q_1^i, q_2^i, \ldots, q_{m_i}^i$ that satisfies the conditions in Theorem 1. Every top-level subquery $q_i$ $(1 \leq i \leq n)$ for which such an order exists, is Web-safe (cf. Theorem 1). If all top-level subqueries are identified to be Web-safe by this test, then $q_{\mathsf{nf}}$ is Web-safe (cf. Proposition 1), and so is $q$ (cf. Fact 1).

We conclude the section by pointing out the following limitation of our results: Even if the given conditions are sufficient to show that an LDQL query is Web-safe, they are not sufficient for showing the opposite. It remains an open question whether there exists a (decidable) property of all Web-safe LDQL queries that is sufficient *and* necessary.

## 5 Concluding Remarks and Future Work

LDQL, the query language that we introduce in this paper, allows users to express queries over Linked Data on the WWW. We defined LDQL such that navigational features for selecting the query-relevant documents on the Web are separate from patterns that are meant to be evaluated over the data in the selected documents. This separation distinguishes LDQL from other approaches to express queries over Linked Data. For instance, neither any of the existing SPARQL-based approaches [4,10,11,14,16] nor NautiLOD [8] can be used to express queries such as our example queries $q'_{\mathsf{ex}}$ and $q''_{\mathsf{ex}}$.

Given that our analysis of LDQL in this paper focuses primarily on Web-safeness, in future work the language may be studied with respect to the complexity of query evaluation and the problem of query containment. A formal study of the expressive power of LDQL would also be interesting. A more practical direction for future research on LDQL is the development of approaches to execute LDQL queries efficiently.

## References


1. M. Arenas, C. Gutierrez, and J. Pérez. On the Semantics of SPARQL. In *Semantic Web Information Management - A Model-Based Perspective*, chapter 13. Springer, 2009.
2. T. Berners-Lee. Linked Data. Online at http://www.w3.org/DesignIssues/LinkedData.html, 2006.
3. C. Bizer, T. Heath, and T. Berners-Lee. Linked Data – The Story So Far. *Semantic Web and Information Systems*, 5(3):1–22, 2009.
4. P. Bouquet, C. Ghidini, and L. Serafini. Querying The Web Of Data: A Formal Approach. In *Proceedings of the 4th Asian Semantic Web Conference*, 2009.
5. C. Buil-Aranda, M. Arenas, and O. Corcho. Semantics and Optimization of the SPARQL 1.1 Federation Extension. In *Proceedings of the 8th Extended Semantic Web Conference*, 2011.
6. R. Cyganiak, D. Wood, M. Lanthaler, G. Klyne, J. J. Carroll, and B. McBride. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, Online at http://www.w3.org/TR/rdf11-concepts/, Feb. 2014.
7. R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. J. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, Online at http://tools.ietf.org/html/rfc2616, June 1999.
8. V. Fionda, G. Pirrò, and C. Gutierrez. NautiLOD: A Formal Language for the Web of Data Graph. *ACM Transactions on the Web*, 9(1):1–43, 2015.
9. S. Harris, A. Seaborne, and E. Prud'hommeaux. SPARQL 1.1 Query Language. W3C Recommendation, Online at http://www.w3.org/TR/sparql11-query/, Mar. 2013.
10. A. Harth and S. Speiser. On Completeness Classes for Query Evaluation on Linked Data. In *Proceedings of the 26th AAAI Conference*, 2012.
11. O. Hartig. SPARQL for a Web of Linked Data: Semantics and Computability. In *Proceedings of the 9th Extended Semantic Web Conference*, 2012.
12. O. Hartig. An Overview on Execution Strategies for Linked Data Queries. *Datenbank-Spektrum*, 13(2), 2013.
13. O. Hartig. LDQL: A Language for Linked Data Queries (Extended Version). Online at http://olafhartig.de/files/LDQL-ext.pdf, 2015.
14. O. Hartig and G. Pirrò. A Context-Based Semantics for SPARQL Property Paths over the Web. In *Proceedings of the 12th Extended Semantic Web Conference*, 2015.
15. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and Complexity of SPARQL. *ACM Transactions on Database Systems*, 34, 2009.
16. J. Umbrich, A. Hogan, A. Polleres, and S. Decker. Link Traversal Querying for a Diverse Web of Data. *Semantic Web Journal*, 2014.