Web Semantics: Science, Services and Agents on the World Wide Web 41 (2016) 9–29

Contents lists available at ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: www.elsevier.com/locate/websem

LDQL: A query language for the Web of Linked Data☆

Olaf Hartig [a,*], Jorge Pérez [b]

[a] Department of Computer and Information Science (IDA), Linköpings Universitet, SE-581 83 Linköping, Sweden

[b] Department of Computer Science, Universidad de Chile & Chilean Center for Semantic Web Research, Beauchef 851, Santiago - 8370456, Chile

ARTICLE INFO

Article history:
Received 31 March 2016
Received in revised form
15 August 2016
Accepted 15 October 2016
Available online 29 October 2016

keywords:
Linked Data
Query language
Foundations
SPARQL
Queries

ABSTRACT

The Web of Linked Data is composed of tons of RDF documents interlinked to each other forming a huge repository of distributed semantic data. Effectively querying this distributed data source is an important open problem in the Semantic Web area. In this paper, we propose LDQL, a declarative language to query Linked Data on the Web. One of the novelties of LDQL is that it expresses 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. We present a formal syntax and semantics, prove equivalence rules, and study the expressiveness of the language. In particular, we show that LDQL is strictly more expressive than all the query formalisms that have been proposed previously for Linked Data on the Web. We also study some computability issues regarding LDQL. We first prove that when considering the Web of Linked Data as a fully accessible graph, the evaluation problem for LDQL can be solved in polynomial time. Nevertheless, when the limited data access capabilities of Web clients are considered, the scenario changes drastically; there are LDQL queries for which a complete execution is not possible in practice. We formally study this issue and provide a sufficient syntactic condition to avoid this problem; queries satisfying this condition are ensured to have a procedure to be effectively evaluated over the Web of Linked Data.

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 [1]. These principles are based on standard Web technologies. In particular, (i) the Hypertext Transfer Protocol (HTTP) [2] is used to access data, (ii) HTTP-based Uniform Resource Identifiers (URIs) [3] are used as identifiers for entities described in the data, and (iii) the Resource Description Framework (RDF) [4] 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*.

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 the Web of Linked Data as if it was a single (distributed) database. For an overview on techniques proposed to execute queries over Linked Data on the WWW refer to [5].

While there does not exist a standard language for expressing such queries, a few options have been proposed in the research literature. In particular, a first strand of research focuses on extending the scope of the RDF query language SPARQL [6] such that an evaluation of SPARQL queries over Linked Data on the WWW has a well-defined semantics [7–11]. A second strand of research focuses on navigational languages [12,13]. Although these approaches have different motivations, a commonality of all these proposals is that the definition of query-relevant regions of the Web of Linked Data and the definition of query-relevant data within the specified regions are mixed ; as a result, in their queries, users cannot specify one without affecting the other.

The first main contribution of this paper is the proposal of LDQL, a novel query language for the Web of Linked Data. The most important feature of LDQL is that it clearly separates query components for selecting query-relevant regions of the Web of Linked Data, from components for specifying the query result that has to be constructed from the data in the selected regions. The most basic construction in LDQL are tuples of the form $\langle L, Q \rangle$ where L is an expression used to select a set of relevant documents,

☆ This paper is an extended and revised version of Hartig and Pérez (2015).

* Corresponding author.
E-mail address: olaf.hartig@liu.se (O. Hartig).

http://dx.doi.org/10.1016/j.websem.2016.10.001
1570-8268/

and Q is a query intended to be executed over the data in these documents as if they were a single RDF repository. In an abstract setting one can use several formalisms to express L and Q. In our proposal, for the former part we introduce the notion of *link path expressions* that are a form of nested regular expressions (with some other important features) used to navigate the link graph of the Web. For the latter, we use standard SPARQL graph patterns. Such basic LDQL queries can be combined by using conjunctions, disjunctions, and projection. To begin evaluating these queries one needs to specify a set of seed URIs. The language also possesses features to dynamically (at query time) identify new seed URIs to evaluate portions of a query. In this paper, we present a formal syntax and semantics for LDQL and propose some rewrite rules.

As our second main contribution we compare LDQL with four previously proposed formalisms for querying the Web of Linked Data: *SPARQL under reachability-based query semantics* [8], *SPARQL Property Path patterns under context-based semantics* [10], *SPARQL under full-Web query semantics* [8,10], and *NautiLOD* [13]. We formally prove that LDQL is strictly more expressive than every one of these. That is, we show that for every query Q in any of the previous languages, one can effectively construct an LDQL query that is equivalent to Q. Moreover, for every one of the previous languages, there exists an LDQL query that cannot be expressed in that language. These results show that LDQL presents an interesting expressive power.

Our third contribution is a study of computability issues regarding LDQL. We first study the *classical complexity* of the query language; we show that, in a setting in which the Web of Linked Data is considered as a fully accessible graph, every LDQL query can be evaluated in polynomial time. In contrast, when we consider the intrinsic limitations of data access as per the Linked Data principles, there exists queries for which a complete execution is not possible in practice. 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 LDQL queries that are Web-safe. Our last technical contribution is the identification of a sufficient syntactic condition for Web-safeness.

The rest of the paper is structured as follows. Section 2 provides an overview of related work. Section 3 introduces a data model that provides the basis for defining the semantics of LDQL. In Section 4 we formally define the syntax and semantics of LDQL and show some simple algebraic properties. In Section 5 we compare LDQL with the three mentioned languages, and in Section 6 we focus on computability issues. Section 7 concludes the paper and sketches future work.

Preliminary versions of some of the results in this paper appeared in [14]. The new material added in this version includes a comprehensive discussion of related work, complete proofs for all the results (these proofs were not presented in [14]), detailed translation rules from previous query languages for Linked Data to LDQL, as well as the result on the polynomial classical complexity of the language (Theorem 9) that was presented only as a conjecture in [14].

2. Related work

Since its emergence the WWW has attracted research interest in adopting declarative query languages for retrieving information from the WWW. In this section we briefly review general (i.e., Linked Data independent) query languages for the WWW and, afterwards, discuss existing query formalisms and languages designed to query the Web of Linked Data.

We do not compare LDQL with more standard graph navigational languages [15] such as XPath [16], GraphLog [17], and nSPARQL [18], or the formalisms used in graph database systems like Neo4j [19] or Sparksee [20], as all of them are designed to navigate graph data in a centralized scenario in which the graph is stored locally. An interesting direction for future research is to explore more expressive ways of navigating graphs, for instance GraphLog [17], and adapt them as the navigational part of LDQL.

2.1. Early work on Web query languages

Initial work on querying the WWW emerged in the late 1990s. Florescu et al.'s survey provides an overview on early work in this area [21]. Most of this work is based on an understanding of the WWW as a distributed hypertext system consisting of Web pages that are interconnected by hypertext links.

Query languages proposed and studied in this context can be grouped into languages to retrieve either specific Web pages (e.g., W3QL [22,23]), particular attributes of specific Web pages (e.g., WebSQL [24,25], F-logic [26], Web Calculus [27]), or particular content within specific Web pages (e.g., WebLog [28], WebOQL [29], NetQL [30], NALG [31], Squeal [32], HTML-QL [33], WQL [34]). Common to these languages is the navigational nature of the queries. That is, each of these languages is based on some form of path expression that allows users to specify navigation paths to relevant Web pages. Additionally, the query languages that belong to the third group possess features to select content within the relevant pages; hence, these languages are similar in spirit to LDQL.

However, by using these earlier Web query languages, Web data can be retrieved only in an unstructured or, at best, semi-structured form. In contrast, the data considered by LDQL (and by the other Linked Data related query languages that we discuss in the following) is structured and query results may combine such data from multiple separate sources. Another distinctive novelty of some Linked Data query languages, including LDQL, is that navigation paths can be specified in terms of data links (as opposed to ordinary hypertext links).

2.2. SPARQL-based query formalisms for linked data

Live execution of declarative queries directly over the Web of Linked Data has attracted much attention recently (e.g., [5,11, 35–37]). The majority of existing work on query execution and optimization approaches proposed in this context assumes that the queries to be executed are expressed by using the conjunctive fragment of SPARQL (i.e., SPARQL basic graph patterns). However, the SPARQL standards do not provide a formal foundation to apply SPARQL in this context. Nonetheless, SPARQL seems to be a natural first choice given that Linked Data is based on the RDF data model and SPARQL is the standard query language for RDF data. Consequently, multiple proposals exist for adapting the standard query semantics of SPARQL to provide for well-defined queries over data that can be accessed as per the Linked Data principles.

Bouquet et al. were the first to provide a formalization for using SPARQL basic graph patterns (BGPs) as a language for Linked Data queries [7]. We went a step further and considered a more expressive fragment of SPARQL [8]. Other BGP-focused proposals have been published by Umbrich et al. [11] and by Harth and Speiser [9]. In the following, we describe these proposals informally.

Bouquet et al. formalized three *"query methods"* for BGPs [7]: First, the *"bounded method"* assumes that queries contain a specification that enumerates a particular set of documents. The evaluation of such a query is then restricted to the data in these documents. Informally, this method corresponds to a restricted form of the most basic LDQL construction $\langle L, Q \rangle$ in which L is restricted to simply contain a list of pointers to documents and Q is some BGP. Bouquet et al.'s second method, the *"navigational method"*, is based on a notion of reachability that assumes a

recursive traversal of all data links in a queried Web. The result of a query must be computed by taking into account all data that can be discovered by starting such a traversal from a designated document. This method also corresponds to a restricted form of the most basic LDQL construction $\langle L, Q \rangle$; in this case, L is restricted to be an expression that specifies an exhaustive, recursive traversal, and Q is some BGP again. For their third method, called *"direct access method"*, Bouquet et al. assume an oracle that, for any given query, selects a set of *"relevant"* documents from the queried Web. Without providing an idea of their notion of relevance in this context, the authors define an expected query result based on such a set of relevant documents. Due to the undefined basis of this definition, this third query method cannot be meaningfully compared to LDQL (or to any other query formalism).

Instead of focusing on BGPs only, in our earlier work we considered a more expressive fragment of SPARQL (including the operators AND, OPT, UNION, and FILTER) for which we introduced a *full-Web* query semantics and a family of *reachability-based* query semantics [8]. Informally, under the full-Web semantics, the scope of evaluating SPARQL expressions is all Linked Data on the queried Web. Based on a formal analysis, we showed that it is impossible in practice to compute complete query results under this semantics. The reachability-based semantics address this limitation by restricting the scope of the evaluation to data that is reachable by traversing a particular, well-defined set of data links. The most restrictive version of these reachability-based semantics resembles Bouquet et al.'s bounded method, and the least restrictive version resembles the navigational method. For a comparison between (selected) reachability-based semantics and LDQL we refer to Section 5.1 in which we show that LDQL is strictly more expressive than SPARQL under these semantics. Additionally, in Section 5.3 we show that the same holds for LDQL versus SPARQL under full-Web semantics.

Umbrich et al. focus on BGPs and define five different query semantics for conjunctive Linked Data queries [11]. The first of these semantics resembles one of the aforementioned reachability-based semantics; namely, the c_{Match}-semantics (cf. Section 5.1). Umbrich et al.'s other query semantics extend this c_{Match}-semantics to *"benefit [from] inferable knowledge"* [11]. Thus, these extensions take into account additional RDF triples that can be inferred from data available on the queried Web. In particular, these query semantics integrate (i) lightweight RDFS reasoning [38] (restricted to a fixed, a-priori defined set of vocabularies), and (ii) inference rules for RDF triples with the predicate owl:sameAs [39]. While LDQL, as presented in this paper, does not provide features for leveraging inferable knowledge, we consider possible extensions in this direction as a very interesting topic for future research.

Harth and Speiser also focus on BGPs only and propose several Linked Data related query semantics for them [9]. These semantics use authoritativeness of data sources to restrict the evaluation of queries to particular subsets of all data in a queried Web. Unfortunately, the proposal lacks a proper formal definition of one of the key concepts for specifying authority restrictions (that is, the concept of an *"authoritative lookup"* [9, Definition 10]). Therefore, it is impossible to discuss Harth and Speiser's query semantics in detail or to provide an informed comparison with other query formalisms or languages such as LDQL.

A common characteristic of all these Linked Data specific adaptations of SPARQL is that query results are described in terms of SPARQL patterns that have to be matched against the (virtual) union of all RDF data from a particular subset of the data sources on the Web of Linked Data. However, none of these adaptations provides a means to explicitly specify this subset of data sources to be considered. LDQL addresses this limitation.

2.3. Navigational languages for the Web of Linked Data

Instead of trying to adapt SPARQL to express queries over the Web of Linked Data, some research groups have started to work on new query languages for Linked Data. To the best of our knowledge, two such languages have been proposed in the literature: LDPath [12] and NautiLOD [13]. Both of these languages are navigational languages tailored to query Linked Data on the Web. That is, they introduce some form of path expressions based on which a user may specify navigation paths over the graph that emerges from the existence of data links between Linked Data documents on the Web. Hence, these languages are similar in nature to the first group of the early Web query languages mentioned in Section 2.1. In the following we briefly describe both languages.

In LDPath [12], the basic type of path expressions is a *"property selection"* that is represented by a URI. Such an expression selects the object of any RDF triple whose subject is the current *"context resource"* and whose predicate is the given URI. More complex LDPath path expressions can be built recursively by concatenating subexpressions or combining them via a union or an intersection operator. Additionally, each subexpression may be associated with a *"path test"* that represents a condition for filtering the result of the subexpression. To our knowledge, there does not exist a formally defined semantics for LDPath. However, according to Schaffert et al. [12], *"LDPath [...] allows traversal over the conceptual RDF graph represented by interlinked Linked Data servers"*. Unfortunately, a precise definition of this graph structure is missing, and so is a definition of the particular graph that needs to be considered for evaluating a given LDPath expression. Instead, the authors informally suggest that *"path traversal transparently "hops over" to other Linked Data servers when needed"* [12]. Due to the lack of a formal semantics, we ignore LDPath in the rest of this paper.

NautiLOD expressions, in contrast, come with a formal semantics [13]. The result of evaluating such an expression is a set of URIs whose lookup yields a Linked Data document that is the end vertex of some path specified by the expression. The basic building blocks of NautiLOD expressions are very similar to LDPath. However, test expressions are more powerful because, in NautiLOD, those tests are represented using existential, SPARQL-based subqueries and, thus, provide the full expressive power of the SPARQL query language. Informally, a URI in the tested result of the corresponding NautiLOD subexpression passes the test, if the existential test query evaluates to true over the data that can be retrieved by looking up this URI. Another interesting feature of NautiLOD are action subexpressions that can be embedded into a NautiLOD path expression. Represented actions are then performed as side-effects of navigating along the specified paths. Such an action may be the retrieval of data into a local store or the sending of a notification message [13]. Our proposed language, LDQL, does not provide such an actions feature (but it would be trivial to add such a feature for applications designed to leverage it). If we ignore actions and analyze the expressive power of the navigational core of NautiLOD, we shall see that it is strictly less expressive than LDQL (cf. Section 5.4).

As an alternative to defining a new language for navigation over Linked Data, we have recently investigated an approach to use the property paths feature of SPARQL 1.1 [6, Section 9] as a navigational language for the Web of Linked Data [10]. To this end, we have defined a so-called context-based semantics for property path expressions that is inspired by the semantics of NautiLOD. Similar to the navigational core of NautiLOD, the resulting language is strictly less expressive than LDQL as we show in Section 5.2.

While LDPath, NautiLOD, and property paths expressions focus on navigation, our goal with LDQL is to provide a language that enables users to combine NautiLOD-style navigation with SPARQL-style RDF data matching.

3. Data model

In this section we introduce a structural data model that captures the concept of a Web of Linked Data formally. As usual [8–11,13], 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 [4] 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). An *RDF triple* is a tuple $\langle s, p, o\rangle \in \mathcal{T}$ with $\mathcal{T} = (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$. For any RDF triple $t = \langle s, p, o\rangle$ we write $\text{uris}(t)$ 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 *documents* and use them to represent the concept of Web documents from which Linked Data can be extracted. Hence, we assume a function, say data, that maps each document $d \in \mathcal{D}$ to a finite set of RDF triples $\text{data}(d) \subseteq \mathcal{T}$ such that the data of each document uses a unique set of blank nodes.

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

Definition 1. Assume a special symbol $\perp$ such that $\perp \notin \mathcal{D}$. A **Web of Linked Data** is a tuple $W = \langle D, adoc\rangle$ that consists of the following two elements:

- $D \subseteq \mathcal{D}$ is a set of documents; and
- $adoc$ is a function that maps every URI either to a document in D or to the symbol $\perp$ (i.e., $adoc : \mathcal{U} \to D \cup \{\perp\}$) such that for every document $d \in D$, there exists a URI $u \in \mathcal{U}$ with $adoc(u) = d$.

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. URIs that cannot be looked up, or whose look up does not result in retrieving a document (even after following HTTP-based redirection pointers) are mapped to the special symbol $\perp$. Hereafter, we write $\text{dom}^{\not\perp}(adoc)$ to denote the set of URIs that function $adoc$ maps to a document (instead of $\perp$); i.e., $\text{dom}^{\not\perp}(adoc) = \{u \in \mathcal{U} | adoc(u) \neq \perp\}$. For any URI $u \in \mathcal{U}$ with $u \in \text{dom}^{\not\perp}(adoc)$ (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 every document $d \in D$ to be in the image of function $adoc$ because we conceive documents as irrelevant for a Web of Linked Data if they cannot be retrieved by any URI lookup in this Web.

Let $W = \langle D, adoc\rangle$ be a Web of Linked Data. W is said to be finite if the set $\text{dom}^{\not\perp}(adoc)$ is finite. In this paper we assume that every Web of Linked Data is finite. Given documents $d, d' \in D$ and a triple $t \in \text{data}(d)$, we say that a URI $u \in \text{uris}(t)$ establishes a *data link* from d to d', if $adoc(u) = d'$. As a final concept, we formalize the notion of a *link graph* associated to W. This graph has documents in D as nodes, and directed edges representing data links between documents. 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.

Definition 2. The **link graph** of a Web of Linked Data $W = \langle D, adoc\rangle$, denoted by $\mathcal{G}_W$, is a directed, edge-labeled multigraph, $\mathcal{G}_W = \langle D, E_W\rangle$, whose set of labeled edges is defined as follows:

$$E_W = \Big\{\langle d_{\text{src}}, (t, u), d_{\text{tgt}}\rangle \subseteq D \times (\mathcal{T} \times \mathcal{U}) \times D \;\Big|\; t \in \text{data}(d_{\text{src}}) \text{ and } u \in \text{uris}(t) \text{ and } d_{\text{tgt}} = adoc(u)\Big\}.$$

For a link graph edge $e = \langle d_{\text{src}}, (t, u), d_{\text{tgt}}\rangle$, tuple (t, u) is the label of e. Moreover, we sometimes write $e \in \mathcal{G}_W$ to denote that e is an edge in the link graph $\mathcal{G}_W$.

Example 1. As a running example for this paper we assume a Web of Linked Data $W_{\text{ex}} = \langle D_{\text{ex}}, adoc_{\text{ex}}\rangle$ that consists of three documents, $D_{\text{ex}} = \{d_{\text{M1}}, d_{\text{M2}}, d_{\text{M3}}\}$, The data in these documents are the following sets of RDF triples:

$$\text{data}(d_{\text{M3}}) = \big\{\langle u_{\text{Revolutions}}, u_{\text{sequelOf}}, u_{\text{Reloaded}}\rangle, \langle u_{\text{Reloaded}}, u_{\text{influencedBy}}, u_{\text{Matrix1}}\rangle\big\},$$
$$\text{data}(d_{\text{M2}}) = \big\{\langle u_{\text{Reloaded}}, u_{\text{sequelOf}}, u_{\text{Matrix1}}\rangle\big\},$$
$$\text{data}(d_{\text{M1}}) = \big\{\langle u_{\text{Revolutions}}, u_{\text{influencedBy}}, u_{\text{Matrix1}}\rangle\big\}.$$

Moreover, for function $adoc_{\text{ex}}$ we have $\text{dom}^{\not\perp}(adoc_{\text{ex}}) = \{u_{\text{Matrix1}}, u_{\text{Reloaded}}, u_{\text{Revolutions}}, u_{\text{sequelOf}}\}$ such that

$$adoc_{\text{ex}}(u_{\text{Matrix1}}) = d_{\text{M1}}, \quad adoc_{\text{ex}}(u_{\text{Revolutions}}) = d_{\text{M3}},$$
$$adoc_{\text{ex}}(u_{\text{Reloaded}}) = d_{\text{M2}}, \quad adoc_{\text{ex}}(u_{\text{sequelOf}}) = d_{\text{M3}}.$$

This Web contains 10 data links. For instance, the RDF triple $\langle u_{\text{Revolutions}}, u_{\text{influencedBy}}, u_{\text{Matrix1}}\rangle \in \text{data}(d_{\text{M1}})$ with the URI $u_{\text{Revolutions}}$ establishes a data link to document d_{M3}. Hence, the corresponding edge in the link graph of W_{ex} is $\langle d_{\text{M1}}, (\langle u_{\text{Revolutions}}, u_{\text{influencedBy}}, u_{\text{Matrix1}}\rangle, u_{\text{Revolutions}}), d_{\text{M3}}\rangle$. Fig. 1 illustrates the link graph $\mathcal{G}_{W_{\text{ex}}}$ with all 10 edges.

4. 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 of components: *Link path expressions* (*LPEs*) for selecting query-relevant documents of the queried Web of Linked Data; and 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 [6], including the algebraic formalization introduced in [40,41]. In particular, for SPARQL graph patterns we closely follow the formalization in [41] considering operators AND, OPT, UNION, FILTER, and GRAPH, plus the operator BIND defined in [6].

We begin this section by introducing the most basic concept of our language, the notion of link patterns. We use link patterns as the basis for navigating the link graph of a Web of Linked Data.

4.1. Link patterns

A link pattern is a tuple in

$$\big(\mathcal{U} \cup \{\_, +\}\big) \times \big(\mathcal{U} \cup \{\_, +\}\big) \times \big(\mathcal{U} \cup \mathcal{L} \cup \{\_, +\}\big)$$

with _ and + special symbols not in $\mathcal{U}$, $\mathcal{L}$, or $\mathcal{B}$. Link patterns are used to match link graph edges in the context of a designated *context* URI. The special symbol + denotes a placeholder for the context URI. The special symbol _ denotes a wildcard that will drive the direction of the navigation. Before formalizing how link graph edges actually match link patterns, we show some intuition. Consider the link graph of Web W_{ex} in Example 1 (see Fig. 1), and the link pattern $\langle +, p_1, \_\rangle$. Intuitively, in the context of URI u_{A}, the edge with label $(\langle u_{\text{A}}, p_1, u_{\text{B}}\rangle, u_{\text{B}})$ from document d_{A} to document d_{B}, matches the link pattern $\langle +, p_1, \_\rangle$. Notice that in the matching, the context URI u_{A} takes the place of symbol +, and u_{B} takes the place of the wildcard symbol _. Notice that u_{B} also denotes the direction of the edge that matches the link pattern. On the other hand, the edge with label $(\langle u_{\text{A}}, p_1, u_{\text{B}}\rangle, u_{\text{A}})$ from d_{A} to d_{A}, does not match $\langle +, p_1, \_\rangle$; although u_{B} can take the place of the wildcard symbol _, the direction of the edge is not to u_{B}. That is, when matching an edge labeled by (t, u) we require URI u to be taking the place of a wildcard in the link pattern. When more than one wildcard symbol is used, the link pattern can be

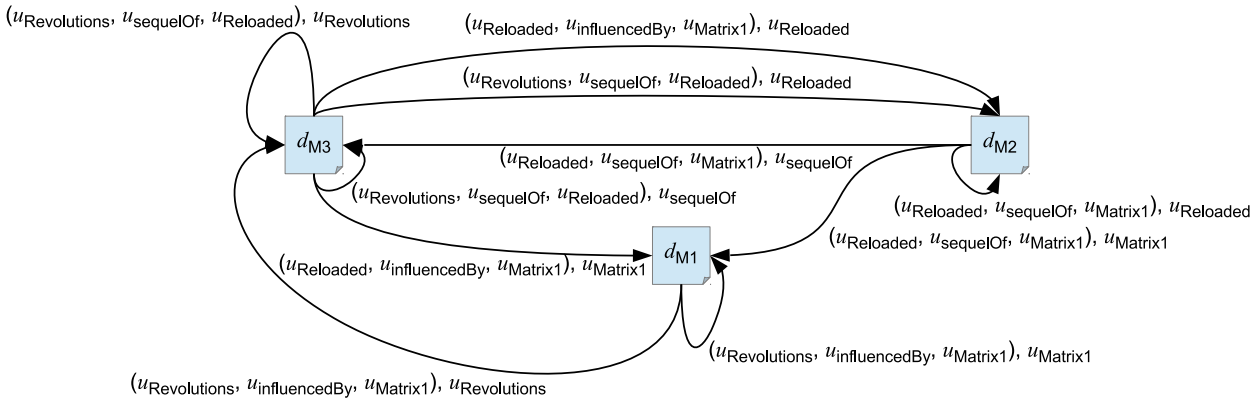

Fig. 1. The link graph $\mathcal{G}_{W_{ex}}$ of our example Web of Linked Data W_{ex}.

matched by edges pointing to the direction of any of the URIs taking the place of a wildcard. For instance, in the context of u_A, the link pattern $\langle \_, p_2, \_ \rangle$ is matched by edges $\langle d_A, (\langle u_B, p_2, u_C \rangle, u_B), d_B \rangle$ and $\langle d_A, (\langle u_B, p_2, u_C \rangle, u_C), d_C \rangle$. The next definition formalizes this notion of matching.

Definition 3. A link graph edge with label $(\langle x_1, x_2, x_3 \rangle, u)$ **matches** a link pattern $\langle y_1, y_2, y_3 \rangle$ in the context of a URI u_{ctx} if the following two properties hold:

1. there exists $i \in \{1, 2, 3\}$ such that $y_i = \_$ and $x_i = u$, and
2. for every $i \in \{1, 2, 3\}$ either $y_i = +$ and $x_i = u_{ctx}$, or $y_i = x_i$, or $y_i = \_$.

One of the rationales for adopting the notion of a context URI and the $+$ symbol 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* in the following sense: A data link is authoritative if it is established by a triple in the source document of the link such that this triple is a statement that uses a URI for which the source document is the authoritative source of data. More formally, a data link represented by link graph edge $\langle d_{src}, (t, u), d_{tgt} \rangle \in \mathcal{G}_W$ is called authoritative in a Web of Linked Data $W = \langle D, adoc \rangle$ if $d_{src} = adoc(u')$ for some URI $u' \in \text{uris}(t)$. For instance, in our example Web (cf. Example 1 and Fig. 1) all data links are authoritative except for the links established by the triple $\langle u_{Reloaded}, u_{influencedBy}, u_{Matrix1} \rangle$ in document d_{M3}. By using the symbol $+$ in a link pattern, the navigation can be restricted to follow only authoritative data links from document $d_{ctx} = adoc(u_{ctx})$, whereas, with the wildcard $\_$, every data link from d_{ctx} would be followed.

4.2. LDQL queries

The most basic construction in LDQL queries are tuples of the from $\langle L, P \rangle$ where L is an expression used to select a set of documents from the Web of Linked Data, and P is a SPARQL graph pattern to query these documents as if they were a single RDF dataset. In an abstract setting, one can use any formalism to specify L as long as L defines sets of RDF documents. In our proposal we use what we call *link path expressions* (LPEs) that are a form of nested regular expressions [18] over the alphabet of link patterns. Every link path expression begins its navigation in a context URI, traverses the Web, and returns a set of URIs; these URIs are used to construct an RDF dataset with all the documents to be retrieved by looking up the URIs. This dataset is passed to the SPARQL graph pattern to obtain the final evaluation of the whole query. Besides the basic constructions of the form $\langle L, P \rangle$, in LDQL one can also use AND, UNION and projection, to combine them. We also introduce an operator SEED that is used to dynamically change, at query time, the seed URI from which the navigation begins. The next definition formalizes the syntax of LDQL queries and LPEs.

Definition 4. The syntax of LDQL is given by the following production rules in which lp is an arbitrary link pattern, $?v$ is a variable, P is a SPARQL graph pattern (as per [41]), V is a finite set of variables, and U is a finite set of URIs:

$$q := \langle lpe, P \rangle \mid (\text{SEED } U\ q) \mid (\text{SEED } ?v\ q) \mid (q \text{AND} q) \mid (q \text{UNION} q) \mid \pi_V q$$
$$lpe := \varepsilon \mid lp \mid lpe/lpe \mid lpe|lpe \mid lpe^* \mid [lpe] \mid \langle ?v, q \rangle.$$

Any expression that satisfies the production q is an **LDQL query**, any expression that satisfies the production lpe is a **link path expression** (**LPE**), and any LDQL query of the form $\langle lpe, P \rangle$ is a **basic LDQL query**.

Before going into the formal semantics of LDQL and LPEs, we give some more intuition about how these expressions are evaluated in a Web of Linked Data W. As mentioned before, the most basic expression in LDQL is of the form $\langle lpe, P \rangle$. To evaluate this expression over W we will need a set S of *seed* URIs. When evaluating $\langle lpe, P \rangle$, every one of the seed URIs in S will trigger a navigation of link graph $\mathcal{G}_W$ via the link path expression lpe starting on that seed. That is, the seed URIs are passed to lpe as *context* URIs in which the LPE should be evaluated. These evaluations of lpe will result in a set of URIs that are used to construct a dataset over which P is finally evaluated.

Regarding the navigation of link graph $\mathcal{G}_W$, the most basic form of navigation is to follow a single link graph edge that matches a link pattern lp. When a navigation via a link pattern lp is triggered from a context URI u, we proceed as follows. We first go to the authoritative document for u, that is $adoc(u)$, and try to find outgoing link graph edges that match lp in the context of u (as explained in Section 4.1). Every one of these matches defines a new context URI u' from which the navigation can continue. More complex forms of navigation are obtained by combining link patterns via classical regular expression operators such as concatenation $/$, disjunction $|$, and recursive concatenation $(\cdot)^*$. The nesting operator $[\cdot]$ is used to test for existence of paths. When a context URI u is passed to an expression $[lpe]$, it checks whether $\mathcal{G}_W$ contains a path from $d_{ctx} = adoc(u)$ that matches lpe. If such a path exists, the navigation can continue from the same context URI u. The most involved form of navigation is by using the expression $\langle ?v, q \rangle$ with q an LDQL query. To evaluate this expression from context URI u one first has to pass u as a seed URI for q and recursively evaluate q from that seed. This evaluation generates a set of solution mappings, and for every one of these

mappings its value on variable $?v$ is used as the new context URI from which the navigation continues. Finally, 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.

To formally define the semantics of LDQL we need to introduce some terminology. We first define a function $\text{dataset}_W(\cdot)$ that from a set of URIs constructs an RDF dataset with all the documents pointed to by those URIs in W. Formally, given a Web of Linked Data $W = \langle D, adoc\rangle$ and a set U of URIs, $\text{dataset}_W(U)$ is an RDF dataset (as per [6,41]) that has the set of triples $\{t \in \text{data}(adoc(u)) \mid u \in (U \cap \text{dom}^{\not\perp}(adoc))\}$ as default graph. Moreover, for every URI $u \in (U \cap \text{dom}^{\not\perp}(adoc))$, $\text{dataset}_W(U)$ contains the named graph $\langle u, \text{data}(adoc(u))\rangle$.

Example 2. Consider the Web W_{ex} in Example 1 and the set $U = \{u_{\text{Revolutions}}, u_{\text{Matrix1}}\}$ of URIs. Then, $\text{dataset}_{W_{\text{ex}}}(U)$ is the set

$$\text{dataset}_{W_{\text{ex}}}(U) = \{G_0, \langle u_{\text{Revolutions}}, G_1\rangle, \langle u_{\text{Matrix1}}, G_2\rangle\}$$

with two named graphs, $\langle u_{\text{Revolutions}}, G_1\rangle$ and $\langle u_{\text{Matrix1}}, G_2\rangle$, such that

$$G_1 = \big\{\langle u_{\text{Revolutions}}, u_{\text{sequelOf}}, u_{\text{Reloaded}}\rangle, \langle u_{\text{Reloaded}}, u_{\text{influencedBy}}, u_{\text{Matrix1}}\rangle\big\}, \quad \text{and}$$

$$G_2 = \big\{\langle u_{\text{Revolutions}}, u_{\text{influencedBy}}, u_{\text{Matrix1}}\rangle\big\},$$

and its default graph is $G_0 = G_1 \cup G_2$.

In the formalization of the semantics of LDQL, we use the standard join operator $\bowtie$ over sets of solution mappings [6,40]. We also make use of the semantics of SPARQL graph patterns over datasets as defined in [41]. In particular, given an RDF dataset $\mathfrak{D}$, and a SPARQL graph pattern P, we denote by $[\![P]\!]^{\mathfrak{D}}$ the evaluation of P over dataset $\mathfrak{D}$ [41, Definition 13.3].

We are now ready to formally define the semantics of LDQL and LPEs. Given a Web of Linked Data W and a set S of URIs, we formalize the evaluation of LDQL queries over W from the seed URIs S, as a function $[\![\cdot]\!]^S_W$ that given an LDQL query, produces a set of solution mappings. Similarly, the evaluation of LPEs over W from a context URI u, is formalized as a function $[\![\cdot]\!]^u_W$ that given an LPE, produces a set of URIs.

Definition 5. Let $W = \langle D, adoc\rangle$ be a Web of Linked Data. Given a finite set $S \subseteq \mathcal{U}$, the S-**based evaluation** of LDQL queries over W, denoted by $[\![\cdot]\!]^S_W$, is a set of solution mappings that is defined recursively as follows:

$$[\![\langle lpe, P\rangle]\!]^S_W = [\![P]\!]^{\mathfrak{D}} \quad \text{where } \mathfrak{D} = \text{dataset}_W\left(\bigcup_{u\in S} [\![lpe]\!]^u_W\right),$$

$$[\![(\text{SEED } U\ q)]\!]^S_W = [\![q]\!]^U_W,$$

$$[\![(\text{SEED } ?v\ q)]\!]^S_W = \bigcup_{u\in \text{dom}^{\not\perp}(adoc)} \left([\![q]\!]^{\{u\}}_W \bowtie \{\mu_u\}\right)$$

where $\mu_u = \{?v \mapsto u\}$,

$$[\![(q_1 \text{UNION} q_2)]\!]^S_W = [\![q_1]\!]^S_W \cup [\![q_2]\!]^S_W,$$

$$[\![(q_1 \text{AND} q_2)]\!]^S_W = [\![q_1]\!]^S_W \bowtie [\![q_2]\!]^S_W,$$

$$[\![\pi_V q]\!]^S_W = \{\mu \mid \text{there exists } \mu' \in [\![q]\!]^S_W \text{ such that } \mu \text{ and } \mu' \text{ are compatible and } \text{dom}(\mu) = \text{dom}(\mu') \cap V\}.$$

For the semantics of LPEs, given a context URI $u_{\text{ctx}} \in \mathcal{U}$, if $u_{\text{ctx}} \in \text{dom}^{\not\perp}(adoc)$, then the u_{ctx}-**based evaluation** of LPEs over W, denoted by $[\![\cdot]\!]^{u_{\text{ctx}}}_W$, is defined recursively as follows:

$$[\![\varepsilon]\!]^{u_{\text{ctx}}}_W = \{u_{\text{ctx}}\},$$

$$[\![lp]\!]^{u_{\text{ctx}}}_W = \{u \in \mathcal{U} \mid \text{there exist a link graph edge } \langle d_{\text{src}}, (t, u), d_{\text{tgt}}\rangle \in \mathcal{G}_W, \text{ with } d_{\text{src}} = adoc(u_{\text{ctx}}), \text{ that matches } lp \text{ in the context of } u_{\text{ctx}}\},$$

$$[\![lpe_1/lpe_2]\!]^{u_{\text{ctx}}}_W = \{u \in [\![lpe_2]\!]^{u'}_W \mid u' \in [\![lpe_1]\!]^{u_{\text{ctx}}}_W\},$$

$$[\![lpe_1|lpe_2]\!]^{u_{\text{ctx}}}_W = [\![lpe_1]\!]^{u_{\text{ctx}}}_W \cup [\![lpe_2]\!]^{u_{\text{ctx}}}_W,$$

$$[\![lpe^*]\!]^{u_{\text{ctx}}}_W = \{u_{\text{ctx}}\} \cup [\![lpe]\!]^{u_{\text{ctx}}}_W \cup [\![lpe/lpe]\!]^{u_{\text{ctx}}}_W \cup [\![lpe/lpe/lpe]\!]^{u_{\text{ctx}}}_W \cup \cdots,$$

$$[\![[lpe]]\!]^{u_{\text{ctx}}}_W = \{u_{\text{ctx}} \mid [\![lpe]\!]^{u_{\text{ctx}}}_W \neq \emptyset\},$$

$$[\![\langle ?v, q\rangle]\!]^{u_{\text{ctx}}}_W = \{u \in \mathcal{U} \mid \text{there exists } \mu \in [\![q]\!]^{\{u_{\text{ctx}}\}}_W \text{ such that } \mu(?v) = u\}.$$

Moreover, if $u_{\text{ctx}} \notin \text{dom}^{\not\perp}(adoc)$, then $[\![lpe]\!]^{u_{\text{ctx}}}_W = \emptyset$ for every LPE lpe.

Example 3. Let lpe_{ex} be the LPE $\langle \_, u_{\text{sequelOf}}, \_\rangle^*/[\langle \_, u_{\text{influencedBy}}, \_\rangle]$. This LPE selects documents that can be reached via arbitrarily long paths of data links with predicate u_{sequelOf} and, additionally, have some outgoing data link with predicate $u_{\text{influencedBy}}$. For our example Web W_{ex} and context URI $u_{\text{Revolutions}}$, the LPE selects documents $d_{\text{M3}} = adoc_{\text{ex}}(u_{\text{Revolutions}})$ and $d_{\text{M1}} = adoc_{\text{ex}}(u_{\text{Matrix1}})$. More precisely, we have $[\![lpe_{\text{ex}}]\!]^{u_{\text{Revolutions}}}_{W_{\text{ex}}} = \{u_{\text{Revolutions}}, u_{\text{Matrix1}}\}$. Note that document d_{M2} can also be reached via a u_{sequelOf} – path, but it does not pass the $u_{\text{influencedBy}}$ – related test.

Example 4. Consider a set of URIs $S_{\text{ex}} = \{u_{\text{Revolutions}}\}$ and a basic LDQL query $\langle lpe_{\text{ex}}, B_{\text{ex}}\rangle$ whose LPE is lpe_{ex} as introduced in Example 3 and whose SPARQL graph pattern is a basic graph pattern that contains two triple patterns,

$$B_{\text{ex}} = \big\{\langle ?x, u_{\text{sequelOf}}, ?y\rangle, \langle ?x, u_{\text{influencedBy}}, ?z\rangle\big\}.$$

Given that $[\![lpe_{\text{ex}}]\!]^{u_{\text{Revolutions}}}_{W_{\text{ex}}} = \{u_{\text{Revolutions}}, u_{\text{Matrix1}}\}$ (cf. Example 3), the default graph of $\text{dataset}_{W_{\text{ex}}}([\![lpe_{\text{ex}}]\!]^{u_{\text{Revolutions}}}_{W_{\text{ex}}})$ is (cf. Example 2):

$$\big\{\langle u_{\text{Revolutions}}, u_{\text{sequelOf}}, u_{\text{Reloaded}}\rangle, \langle u_{\text{Reloaded}}, u_{\text{influencedBy}}, u_{\text{Matrix1}}\rangle, \langle u_{\text{Revolutions}}, u_{\text{influencedBy}}, u_{\text{Matrix1}}\rangle\big\}.$$

Then, according to the query semantics, the result of query $\langle lpe_{\text{ex}}, B_{\text{ex}}\rangle$ over W_{ex} using seeds S_{ex} consists of a single solution mapping, namely $\mu = \{?x \mapsto u_{\text{Revolutions}}, ?y \mapsto u_{\text{Reloaded}}, ?z \mapsto u_{\text{Matrix1}}\}$.

Example 5. Consider an LDQL query $q_{\text{ex}} = (\text{SEED } ?x\ \langle \varepsilon, \langle ?x, u_{\text{sequelOf}}, ?w\rangle\rangle)$ whose subquery is a basic LDQL query that has a single triple pattern as its SPARQL graph pattern. Additionally, let $q'_{\text{ex}} = \langle lpe_{\text{ex}}, B_{\text{ex}}\rangle$ be the basic LDQL query introduced in Example 4, and let q''_{ex} be the conjunction of these two queries; i.e., $q''_{\text{ex}} = (q_{\text{ex}} \text{AND} q'_{\text{ex}})$. By Example 4 we know that $[\![q'_{\text{ex}}]\!]^{S_{\text{ex}}}_{W_{\text{ex}}} = \{\mu\}$ with $\mu = \{?x \mapsto u_{\text{Revolutions}}, ?y \mapsto u_{\text{Reloaded}}, ?z \mapsto u_{\text{Matrix1}}\}$. Furthermore, based on the data given in Example 1, it is easy to see that $[\![q_{\text{ex}}]\!]^{S_{\text{ex}}}_{W_{\text{ex}}} = \{\mu_1, \mu_2\}$ with $\mu_1 = \{?x \mapsto u_{\text{Revolutions}}, ?w \mapsto u_{\text{Reloaded}}\}$ and $\mu_2 = \{?x \mapsto u_{\text{Reloaded}}, ?w \mapsto u_{\text{Matrix1}}\}$. For the S_{ex}-based evaluation of q''_{ex} over W_{ex}, the result sets $[\![q_{\text{ex}}]\!]^{S_{\text{ex}}}_{W_{\text{ex}}}$ and $[\![q'_{\text{ex}}]\!]^{S_{\text{ex}}}_{W_{\text{ex}}}$ have to be joined. Thus, we need to compute $\{\mu_1, \mu_2\} \bowtie \{\mu\}$, which results in a single mapping

$$\mu' = \mu_1 \cup \mu = \{?x \mapsto u_{\text{Revolutions}}, ?w \mapsto u_{\text{Reloaded}}, ?y \mapsto u_{\text{Reloaded}}, ?z \mapsto u_{\text{Matrix1}}\}.$$

4.3. Algebraic properties of LDQL queries

As a basis for the discussion in the next sections, we show some simple algebraic properties. We say that LDQL queries q and q' are semantically equivalent, denoted by $q \equiv q'$, if $[\![q]\!]^S_W = [\![q']\!]^S_W$ holds for every Web of Linked Data W and every finite set $S \subseteq \mathcal{U}$. The following two lemmas follow easily from the definition of the semantics of LDQL.

Lemma 1. *The operators* AND *and* UNION *are associative and commutative.*

Lemma 2. *Let q_1, q_2, and q_3 be LDQL queries, the following equivalences hold:*

$$(q_1 \text{AND} (q_2 \text{UNION} q_3)) \equiv ((q_1 \text{AND} q_2) \text{UNION} (q_1 \text{AND} q_3)) \quad (1)$$

$$\pi_V(q_1 \text{UNION} q_2) \equiv (\pi_V q_1 \text{UNION} \pi_V q_2) \quad (2)$$

$$(\text{SEED } U\ (q_1 \text{UNION} q_2)) \equiv ((\text{SEED } U\ q_1) \text{UNION} (\text{SEED } U\ q_2)) \quad (3)$$

$$(\text{SEED } ?v\ (q_1 \text{UNION} q_2)) \equiv ((\text{SEED } ?v\ q_1) \text{UNION} (\text{SEED } ?v\ q_2)). \quad (4)$$

Lemma 1 allows us to write sequences of either ANDor UNIONwithout parentheses. Our next result shows the power of the construction $\langle ?v, q\rangle$. In particular, it shows that link patterns *lp*, concatenation /, disjunction |, and the test $[\cdot]$, are just *syntactic sugar* as they can be simulated by using ε, $\langle ?v, q\rangle$ and $(\cdot)^*$.

Lemma 3. *There exists a polynomial time procedure* $\text{trans}_L(\cdot)$ *such that for every link path expression lpe, we have that* $\text{trans}_L(lpe)$ *is a link path expression that only uses ε, the construction $\langle ?v, q\rangle$, and operator $(\cdot)^*$, and such that for every URI u and Web of Linked Data W it holds that* $[\![lpe]\!]^u_W = [\![\text{trans}_L(lpe)]\!]^u_W$.

Proof. The proof is based on a recursive translation of link path expressions beginning with link patterns. Let $\langle y_1, y_2, y_3\rangle$ be a link pattern. We construct an LPE $\text{trans}_L(\langle y_1, y_2, y_3\rangle)$ as follows. Assume that $y_1 = \_$, then we construct the LDQL query

$$q_1 = \big\langle \varepsilon, (\text{GRAPH} ?u\ (?out, Y_2, Y_3))\big\rangle$$

where (i) if $y_2 = +$, then $Y_2 = ?u$, (ii) if $y_2 \in \mathcal{U}$, then $Y_2 = y_2$ and (iii) if $y_2 = \_$, then $Y_2 = ?y_2$. And similarly, if (i) $y_3 = +$, then $Y_3 = ?u$, (ii) if $y_3 \in \mathcal{U}$, then $Y_3 = y_3$ and (iii) if $y_3 = \_$, then $Y_3 = ?y_3$. By following a similar process, we construct the LDQL query $q_2 = \langle \varepsilon, (\text{GRAPH} ?u\ (Y_1, ?out, Y_3))\rangle$ if $y_2 = \_$, and the query $q_3 = \langle \varepsilon, (\text{GRAPH} ?u\ (Y_1, Y_2, ?out))\rangle$ if $y_3 = \_$. Now consider an LDQL query q that is the UNION of the above queries for every $y_i = \_$. Then, the LPE $\text{trans}_L(\langle y_1, y_2, y_3\rangle)$ is constructed as

$$\text{trans}_L\big(\langle y_1, y_2, y_3\rangle\big) = \langle ?out, q\rangle.$$

As an example, consider the link pattern $\langle +, p, \_\rangle$ for which we obtain:

$$\text{trans}_L\big(\langle +, p, \_\rangle\big) = \langle ?out, \langle \varepsilon, (\text{GRAPH} ?u\ (?u, p, ?out))\rangle\rangle.$$

Notice that $[\![\langle +, p, \_\rangle]\!]^u_W$ is retrieving all the URIs v such that in the document pointed by u (which is $adoc(u)$), there is a triple of the form $\langle u, p, v\rangle$. Now, in order to evaluate $[\![\langle ?out, \langle \varepsilon, (\text{GRAPH} ?u\ (?u, p, ?out))\rangle\rangle]\!]^u_W$ we first have to compute

$$[\![\langle \varepsilon, (\text{GRAPH} ?u\ (?u, p, ?out))\rangle]\!]^{\{u\}}_W.$$

Notice that since ε is used as the LPE in the expression, the URI that has to be used to construct the dataset to pose the query, is just u. Thus, we have to compute $[\![(\text{GRAPH} ?u\ (?u, p, ?out))]\!]^{\mathfrak{D}}$ where $\mathfrak{D} = \{adoc(u), \langle u, adoc(u)\rangle\}$, from which we obtain all the mappings $\mu = \{?u \mapsto u, ?out \mapsto v\}$ such that $\langle u, p, v\rangle$ is in $adoc(u)$. Thus finally, from $[\![\langle ?out, \langle \varepsilon, (\text{GRAPH} ?u\ (?u, p, ?out))\rangle\rangle]\!]^u_W$ we obtain all the mappings $\{?out \mapsto v\}$ such that $\langle u, p, v\rangle$ is in $adoc(u)$. Which is the same as what we obtain from $[\![\langle +, p, \_\rangle]\!]^u_W$. Along these same lines, it is not difficult to prove that in general $[\![\text{trans}_L(\langle y_1, y_2, y_3\rangle)]\!]^u_W = [\![\langle y_1, y_2, y_3\rangle]\!]^u_W$.

Before defining the translation in general, we make the following observation about SPARQL patterns that we use in the translation. Consider a dataset $\mathfrak{D} = \{G_0, \langle u_1, G_1\rangle, \ldots, \langle u_k, G_k\rangle\}$, and the graph pattern $P = (\text{GRAPH } ?u\ \{\ \})$. According to the semantics of SPARQL [6,41] the evaluation of P over $\mathfrak{D}$ is the set of mapping $\{\mu_1, \ldots, \mu_k\}$ such that $\mu_i = \{?u \mapsto u_i\}$. That is, P retrieves the names (URIs) of the named graphs in the dataset $\mathfrak{D}$.

We can now define the translation in general:

- For the case of LPE $r = \varepsilon$, we have $\text{trans}_L(r) = \varepsilon$.
- For the case of LPE $r = r_1/r_2$, we have $\text{trans}_L(r) = \langle ?v, q\rangle$ where q is:
  $\big(\langle \text{trans}_L(r_1), (\text{GRAPH} ?x\ \{\ \})\rangle \text{ AND } \big(\text{SEED} ?x\ \langle \text{trans}_L(r_2), (\text{GRAPH} ?v\ \{\ \})\rangle\big)\big)$.
- For the case of LPE $r = r_1|r_2$, we have that $\text{trans}_L(r) = \langle ?v, q\rangle$ where q is:
  $\big(\langle \text{trans}_L(r_1), (\text{GRAPH} ?v\ \{\ \})\rangle \text{ UNION } \langle \text{trans}_L(r_2), (\text{GRAPH} ?v\ \{\ \})\rangle\big)$.
- For the case of LPE $r = [r_1]$, we have that $\text{trans}_L(r) = \langle ?v, q\rangle$ where q is:
  $\big(\langle \varepsilon, (\text{GRAPH} ?v\ \{\ \})\rangle \text{ AND } \pi_{\{?v\}}\big(\text{SEED} ?v\ \langle \text{trans}_L(r_1), (\text{GRAPH} ?x\ \{\ \})\rangle\big)\big)$.
- For the case of LPE $r = (r_1)^*$, we have that $\text{trans}_L(r) = (\text{trans}_L(r_1))^*$.

The general proof proceeds by induction. In the following, we focus on proving that $[\![\text{trans}_L(r_1|r_2)]\!]^u_W = [\![r_1|r_2]\!]^u_W$. The proofs for the other cases are similar.

Assume that $u' \in [\![r_1|r_2]\!]^u_W$, then we know that $u' \in [\![r_1]\!]^u_W \cup [\![r_2]\!]^u_W$. If $u' \in [\![r_1]\!]^u_W$ then by induction hypothesis we know that $u' \in [\![\text{trans}_L(r_1)]\!]^u_W$. Now notice that

$$[\![\langle \text{trans}_L(r_1), (\text{GRAPH} ?v\ \{\ \})\rangle]\!]^{\{u\}}_W = [\![(\text{GRAPH} ?v\ \{\ \})]\!]^{\mathcal{D}},$$

where $\mathcal{D} = \text{dataset}_W([\![\text{trans}_L(r_1)]\!]^u_W)$. Thus, given that $u' \in [\![\text{trans}_L(r_1)]\!]^u_W$, we know that $\mathcal{D}$ has a named graph $\langle u', \text{data}(adoc(u'))\rangle$, which implies that the solution mapping $\{?v \to u'\}$ is a solution for $[\![(\text{GRAPH} ?v\ \{\ \})]\!]^{\mathcal{D}}$, and thus $\{?v \to u'\} \in [\![\langle \text{trans}_L(r_1), (\text{GRAPH} ?v\ \{\ \})\rangle]\!]^{\{u\}}_W$. From this it is straightforward to conclude that $u' \in [\![\text{trans}_L(r_1|r_2)]\!]^u_W$. The other direction is similar.

It is clear that the translation procedure can be implemented in polynomial time. Just notice that one can do a single bottom-up pass over the parse tree of the input LPE expression labeling every node with its corresponding translation. After we finish this process, the label of the root is the complete translation of the LPE expression. Moreover, to construct the label of a particular node in the parse tree we use a single copy of the label of every child node plus a constant number of symbols, thus, the label of the root is of linear size w.r.t. the size of the input expression. □

Although not strictly necessary, we decided to keep link patterns and operators /, |, and $[\cdot]$ because they represent a natural and intuitive way of expressing navigation paths. We will use this result later when we analyze the complexity of the language. From the Lemma 3 we directly obtain the following result.

Proposition 1. *For every LDQL query q, there exists an LDQL query q' s.t. $q \equiv q'$ and every LPE in q' consists only of the symbol ε, the construction $\langle ?v, q\rangle$, and operator $(\cdot)^*$. Moreover, q' can be constructed in polynomial time from q.*

5. Comparison with previous Linked Data query formalisms

In this section, we formally compare the expressive power of LDQL with previously proposed formalisms to query Linked Data on the WWW. We focus on the following four approaches as described informally in Section 2: SPARQL under reachability-based semantics [8], SPARQL property path patterns under a context-based semantics [10], SPARQL under full-Web semantics [8,10], and NautiLOD [13]. We prove that LDQL is strictly more expressive than every one of them in the following sense: On one hand, for every query Q in any of these approaches, one can construct an LDQL query that is equivalent to Q, and on the other hand, for each of these approaches, there exists an LDQL query that cannot be expressed using that approach.

5.1. Comparison with SPARQL under reachability-based query semantics

In [8] the author introduces a family of reachability-based query semantics. Based on these semantics, SPARQL graph patterns can be used as a query language for Linked Data on the WWW. Similar to how the scope of evaluating the SPARQL part of a basic LDQL query is restricted to the data of particular documents, reachability-based semantics restrict the scope of SPARQL queries to documents that can be reached by traversing a well-defined set of data links. To specify what data links belong to such a set, the notion of a *reachability criterion* is used; that is, a function $c : \mathcal{T} \times \mathcal{U} \times \mathcal{P} \to \{true, false\}$ where $\mathcal{P}$ denotes the set of all SPARQL graph patterns (recall from Section 3 that $\mathcal{U}$ is the set of all URIs and $\mathcal{T}$ is the set of all RDF triples). Then, given such a reachability criterion c, a finite set S of URIs, and a SPARQL graph pattern P, a document $d \in \mathcal{D}$ is *(c, S, P)-reachable* in a Web of Linked Data $W = \langle D, adoc \rangle$ if at least one of the following two conditions holds:

1. There exists a URI $u \in S$ such that $adoc(u) = d$; or
2. there exists a link graph edge $\langle d_{\text{src}}, (t, u), d_{\text{tgt}} \rangle \in \mathcal{G}_W$ such that (i) d_{src} is (c, S, P)-reachable in W, (ii) $c(t, u, P) = true$, and (iii) $d_{\text{tgt}} = d$.

Notice how the second condition restricts the notion of reachability by ignoring all data links that do not satisfy the given reachability criterion c. Concrete examples of reachability criteria are c_{All}, c_{None}, and c_{Match} [8], where c_{All} selects all data links, and c_{None} ignores all data links; i.e., $c_{\text{All}}(t, u, P) = true$ and $c_{\text{None}}(t, u, P) = false$ for all tuples $\langle t, u, P \rangle \in \mathcal{T} \times \mathcal{U} \times \mathcal{P}$. In contrast to such an all-or-nothing strategy, criterion c_{Match} returns *true* for every data link whose triple matches a triple pattern of the given graph pattern; formally, $c_{\text{Match}}(t, u, P) = true$ if and only if there exists some solution mapping μ such that $\mu[tp] = t$ for an arbitrary triple pattern tp that is contained in P.

Given the notion of a reachability criterion, it is possible to define a family of (reachability-based) query semantics for SPARQL. To this end, let c be a reachability criterion, let S be a finite set of URIs, and let P be a SPARQL graph pattern. Then, for any Web of Linked Data $W = \langle D, adoc \rangle$, the *$S$-based evaluation* of P over W *under c-semantics*, denoted by $[\![P]\!]_W^{\mathsf{R}(c,S)}$, is a set of solution mappings that is equivalent to $[\![P]\!]_G$ where G is the RDF graph that consists of all triples from all documents that are (c, S, P)-reachable in W.

While there exist an infinite number of possible reachability criteria, in this paper we focus on c_{All}, c_{None}, and c_{Match}. The following two results show that LDQL is strictly more expressive than SPARQL graph patterns under any of these three query semantics.

Theorem 1. *Let $c \in \{c_{\text{All}}, c_{\text{None}}, c_{\text{Match}}\}$. There exists an LDQL query q for which there does not exist a SPARQL pattern P such that $[\![P]\!]_W^{\mathsf{R}(c,S)} = [\![q]\!]_W^S$ for every Web of Linked Data W and every finite set $S \subseteq \mathcal{U}$.*

Proof. In the proof we use the following basic LDQL query $Q(?x)$ given by

$$\big\langle \langle +, p, \_ \rangle, (?x, ?x, ?x) \big\rangle.$$

We prove first that the reachability criterion c_{None} cannot be used to express $Q(?x)$. On the contrary, assume that there exists a SPARQL pattern P such that

$$[\![P]\!]_W^{\mathsf{R}(c_{\text{None}},S)} = [\![Q(?x)]\!]_W^S$$

for every S and W. Let u, u', a, b be different elements in $\mathcal{U}$ that are not mentioned in P. Consider now a Web of Linked Data $W_1 = \langle D_1, adoc_1 \rangle$ that consists of two documents, d_1 and d_2, such that $\text{data}(d_1) = \{(u, p, u')\}$ and $\text{data}(d_2) = \{(a, a, a)\}$, and such that $adoc_1(u) = d_1$ and $adoc_1(u') = d_2$. Moreover, consider another Web of Linked Data, $W_2 = \langle D_2, adoc_2 \rangle$, that also contains document d_1, and another document, d_3, such that $\text{data}(d_3) = \{(b, b, b)\}$, and such that $adoc_2(u) = d_1$ and $adoc_2(u') = d_3$. First notice that

$$[\![Q(?x)]\!]_{W_1}^{\{u\}} = \{\{?x \to a\}\} \neq [\![Q(?x)]\!]_{W_2}^{\{u\}} = \{\{?x \to b\}\}.$$

It is easy to see that $[\![P]\!]_{W_1}^{\mathsf{R}(c_{\text{None}},\{u\})} = [\![P]\!]_{W_2}^{\mathsf{R}(c_{\text{None}},\{u\})}$. Just notice that from $\{u\}$, by using the c_{None} criterion, the set of $(c_{\text{None}}, \{u\}, P)$-reachable documents is the same set $\{d_1\}$ in both W_1 and W_2. As a consequence, we have that $[\![Q(?x)]\!]_{W_1}^{\{u\}} \neq [\![Q(?x)]\!]_{W_2}^{\{u\}}$ but $[\![P]\!]_{W_1}^{\mathsf{R}(c_{\text{None}},\{u\})} = [\![P]\!]_{W_2}^{\mathsf{R}(c_{\text{None}},\{u\})}$, which is a contradiction.

To continue with the proof, we now show that the reachability criterion c_{All} cannot be used to express $Q(?x)$. To obtain a contradiction, assume that there exists a pattern P such that

$$[\![P]\!]_W^{\mathsf{R}(c_{\text{All}},S)} = [\![Q(?x)]\!]_W^S$$

for every S and W. Let u, u', a, b be different URIs that are not mentioned in P. Consider now $W_1 = (\{d_1, d_2, d_3\}, adoc_1)$ having three documents with $\text{data}(d_1) = \{(u, p, u')\}$, $\text{data}(d_2) = \{(a, a, a)\}$ and $\text{data}(d_3) = \{(b, b, b)\}$, and such that $adoc_1(u) = d_1$, $adoc_1(u') = d_2$ and $adoc_1(a) = d_3$. Moreover, consider $W_2 = (\{d_1, d_2, d_3\}, adoc_2)$ having exactly the same documents as W_1, and $adoc_2(u) = d_1$, $adoc_2(u') = d_3$ and $adoc_2(b) = d_2$. First notice that

$$[\![Q(?x)]\!]_{W_1}^{\{u\}} = \{\{?x \to a\}\} \neq [\![Q(?x)]\!]_{W_2}^{\{u\}} = \{\{?x \to b\}\}.$$

Now notice that from $\{u\}$, the set of $(c_{\text{All}}, \{u\}, P)$-reachable documents in W_1 is the set $\{d_1, d_2, d_3\}$; d_1 is the document associated to u, d_2 is reachable from d_1 via the URI u', and d_3 is reachable from d_2 via the URI a. Moreover, the set of $(c_{\text{All}}, \{u\}, P)$-reachable in W_2 is also $\{d_1, d_2, d_3\}$; d_1 is the document associated to u, d_3 is reachable from d_1 via the URI u', and d_2 is reachable from d_3 via URI b. Given that the set of $(c_{\text{All}}, \{u\}, P)$-reachable documents is the same in both W_1 and W_2, we have $[\![P]\!]_{W_1}^{\mathsf{R}(c_{\text{All}},\{u\})} = [\![P]\!]_{W_2}^{\mathsf{R}(c_{\text{All}},\{u\})}$. Given that $[\![Q(?x)]\!]_{W_1}^{\{u\}} \neq [\![Q(?x)]\!]_{W_2}^{\{u\}}$, we obtain our desired contradiction.

We now consider the case of c_{Match}, and prove that it cannot be used to express $Q(?x)$. To obtain a contradiction, assume that there exists a SPARQL pattern P such that

$$[\![P]\!]_W^{\mathsf{R}(c_{\text{Match}},S)} = [\![Q(?x)]\!]_W^S$$

for every S and W. Let u, u', u'', a be different URIs that are not mentioned in P. Consider now $W_1 = (\{d_1, d_2\}, adoc_1)$ with $\text{data}(d_1) = \{(u, p, u')\}$ and $\text{data}(d_2) = \{(a, a, a)\}$, and $adoc(u) = d_1$ and $adoc(u') = d_2$. Moreover, consider $W_2 = (\{d_1', d_2'\}, adoc_2)$ with $\text{data}(d_1') = \{(u'', p, u')\}$ and $\text{data}(d_2') = \{(a, a, a)\}$, and $adoc(u) = d_1'$ and $adoc(u') = d_2'$. First notice that

$$[\![Q(?x)]\!]_{W_1}^{\{u\}} = \{\{?x \to a\}\} \neq [\![Q(?x)]\!]_{W_2}^{\{u\}} = \emptyset.$$

We now prove that $[\![P]\!]_{W_1}^{\mathsf{R}(c_{\text{Match}},\{u\})} = [\![P]\!]_{W_2}^{\mathsf{R}(c_{\text{Match}},\{u\})}$. Given that d_1 is the document associated to u in W_1, we have that d_1 is

$(c_{\text{Match}}, \{u\}, P)$-reachable in W_1. Similarly, we know that d'_1 is $(c_{\text{Match}}, \{u\}, P)$-reachable in W_2. Moreover, given that P does not mention u, u' and u'' we have that (u, p, u') matches a triple pattern in P if and only if (u'', p, u') matches a triple pattern in P. Thus we have that d_2 is $(c_{\text{Match}}, \{u\}, P)$-reachable in W_1 if and only if d'_2 is $(c_{\text{Match}}, \{u\}, P)$-reachable in W_2. Thus we have only two cases, either

- $\{d_1\}$ is the set of $(c_{\text{Match}}, \{u\}, P)$-reachable documents in W_1, and $\{d'_1\}$ is the set of $(c_{\text{Match}}, \{u\}, P)$-reachable documents in W_2, or
- $\{d_1, d_2\}$ is the set of $(c_{\text{Match}}, \{u\}, P)$-reachable documents in W_1, and $\{d'_1, d'_2\}$ is the set of $(c_{\text{Match}}, \{u\}, P)$-reachable documents in W_2.

In the first case we have that $[\![P]\!]_{W_1}^{\text{R}(c_{\text{Match}},\{u\})}$ is obtained by evaluating P over $G_1 = \{(u, p, u')\}$, and that $[\![P]\!]_{W_2}^{\text{R}(c_{\text{Match}},\{u\})}$ is obtained by evaluating P over graph $G_2 = \{(u'', p, u')\}$. Given that P does not mention u, u' and u'', we obtain that the evaluation of P over G_1 is the same as the evaluation of P over G_2, which implies that $[\![P]\!]_{W_1}^{\text{R}(c_{\text{Match}},\{u\})} = [\![P]\!]_{W_2}^{\text{R}(c_{\text{Match}},\{u\})}$.

In the second case, $[\![P]\!]_{W_1}^{\text{R}(c_{\text{Match}},\{u\})}$ is obtained by evaluating P over graph $G_1 = \{(u, p, u'), (a, a, a)\}$, and $[\![P]\!]_{W_2}^{\text{R}(c_{\text{Match}},\{u\})}$ is obtained by evaluating P over graph $G_2 = \{(u'', p, u'), (a, a, a)\}$. Then, for the same reason as above, we have that the evaluation of P is the same over G_1 and over G_2, which implies that $[\![P]\!]_{W_1}^{\text{R}(c_{\text{Match}},\{u\})} = [\![P]\!]_{W_2}^{\text{R}(c_{\text{Match}},\{u\})}$. As a consequence, we have proven that $[\![P]\!]_{W_1}^{\text{R}(c_{\text{Match}},\{u\})} = [\![P]\!]_{W_2}^{\text{R}(c_{\text{Match}},\{u\})}$, while $[\![Q(?x)]\!]_{W_1}^{\{u\}} \neq [\![Q(?x)]\!]_{W_2}^{\{u\}}$, which is our desired contradiction. □

Theorem 2. *Let $c \in \{c_{\text{All}}, c_{\text{None}}, c_{\text{Match}}\}$. For every SPARQL graph pattern P there exists an LDQL query q such that $[\![P]\!]_W^{\text{R}(c,S)} = [\![q]\!]_W^S$ for every Web of Linked Data W and every finite set $S \subseteq \mathcal{U}$.*

Proof. Let P be an arbitrary SPARQL graph pattern, let $W = \langle D, adoc \rangle$ be an arbitrary Web of Linked Data, and let S be some finite set of URIs. We prove the theorem by constructing, for each $c \in \{c_{\text{All}}, c_{\text{None}}, c_{\text{Match}}\}$, an LPE lpe^c that allows us to reach all the URIs representing the documents that are (c, S, P)-reachable in W. Then, the LDQL query associated that simulates the S-based evaluation of P is simply $\langle lpe^c, P \rangle$.

The definition of lpe^c for each $c \in \{c_{\text{All}}, c_{\text{None}}, c_{\text{Match}}\}$ is as follows.

$lpe^{c_{\text{All}}}$ is $\langle \_, \_, \_ \rangle^*$,
$lpe^{c_{\text{None}}}$ is ε, and
$lpe^{c_{\text{Match}}}$ is
$\big(\langle ?s, q_1 \rangle \mid \langle ?p, q_1 \rangle \mid \langle ?o, q_1 \rangle \mid \ldots \mid \langle ?s, q_m \rangle \mid \langle ?p, q_m \rangle \mid \langle ?o, q_m \rangle\big)^*$ where $?s$, $?p$ and $?o$ are fresh variables (not used in P), m is the number of triple patterns in P, and for each such triple pattern tp_k $(1 \leq k \leq m)$ there exists a subquery q_k of the form $\langle \varepsilon, P_k \rangle$ with a SPARQL pattern P_k that is constructed as follows: P_k contains the triple pattern $\langle ?s, ?p, ?o \rangle$ and – depending on the form of the corresponding triple pattern $tp_k = \langle s_k, p_k, o_k \rangle$ – may contain additional FILTER operators; in particular, if $s_k \notin \mathcal{V}$, then P_k contains FILTER$?s = s_k$; if $p_k \notin \mathcal{V}$, then P_k contains FILTER$?p = p_k$; and if $o_k \notin \mathcal{V}$, then P_k contains FILTER$?o = o_k$.
For instance, if $P = \{(a, b, ?x)\}$ then $lpe^{c_{\text{Match}}}$ is the expression

$$\big(\langle ?s, \langle \varepsilon, (?s, ?p, ?o)\text{FILTER}(?s = a \wedge ?p = b)\rangle\rangle \mid \langle ?p, \langle \varepsilon, (?s, ?p, ?o)\text{FILTER}(?s = a \wedge ?p = b)\rangle\rangle \mid \langle ?o, \langle \varepsilon, (?s, ?p, ?o)\text{FILTER}(?s = a \wedge ?p = b)\rangle\rangle\big)^*.$$

Then, for each reachability criterion $c \in \{c_{\text{All}}, c_{\text{None}}, c_{\text{Match}}\}$ with its corresponding LPE lpe^c as specified above, we have to show the following equivalence:

$$[\![P]\!]_W^{\text{R}(c,S)} = [\![\langle lpe^c, P \rangle]\!]_W^S . \tag{5}$$

As we have discussed before, and by the definition of the reachability-based query semantics and the definition of LDQL query semantics, in order to prove (5) it is sufficient to prove the following claim.

Claim 1. *For each $c \in \{c_{\text{All}}, c_{\text{None}}, c_{\text{Match}}\}$, the set of all documents that are (c, S, P)-reachable in W is equivalent to the following set of documents:*

$$D^c_{\text{LPE}} = \{adoc(u) \mid u \in [\![lpe^c]\!]_W^{u_{\text{ctx}}} \text{ for some } u_{\text{ctx}} \in S\}.$$

The complete proof of this claim can be found in the Appendix. We just give here some intuition on why the construction works.

Consider the LPE $\langle \_, \_, \_ \rangle$ and a set S of seed URIs. Notice that from S the LPE $\langle \_, \_, \_ \rangle$ allows us to navigate to all the URIs that are mentioned in the documents pointed by the URIs in S. Thus, the LPE $\langle \_, \_, \_ \rangle^* = lpe^{c_{\text{All}}}$ allows one to go from S to the set, say S_1, of all the URIs mentioned in the document pointed by S, and from there to the set, say S_2, of all the URIs mentioned in the document pointed by S_1, and so on. This is exactly the intuition behind the definition of the (c_{All}, S, P)-reachable documents, independent of the pattern P. Similarly, if we consider the LPE ε and a set S of seed URIs, from S the LPE ε allows us to navigate only to the same URIs mentioned in S, and thus we do not reach any document besides the documents pointed by URIs in S. This is exactly the intuition behind the definition of the (c_{None}, S, P)-reachable documents, independent of the pattern P.

For the case of c_{Match}, let lpe_m be the following expression

$$\big(\langle ?s, q_1 \rangle \mid \langle ?p, q_1 \rangle \mid \langle ?o, q_1 \rangle \mid \ldots \mid \langle ?s, q_m \rangle \mid \langle ?p, q_m \rangle \mid \langle ?o, q_m \rangle\big)$$

where the q_i's are defined as stated in the definition of $lpe^{c_{\text{Match}}}$. If there is a triple pattern in P, say for example $(?x, b, u_1)$, then we know that there exists $i \in \{1, \ldots, m\}$ such that $\langle ?o, q_i \rangle$ is one of the disjuncts in lpe_m where q_i is

$$q_i = \langle \varepsilon, \big((?s, ?p, ?o)\text{FILTER}(?p = b \wedge ?o = u_1)\big)\rangle.$$

Now lets focus on q_i. If we begin navigating this LDQL expression from a URI u in S, then, since we stay in u (q_i navigates using ε) we just evaluate the pattern $\big((?s, ?p, ?o)\text{FILTER}(?p = b \wedge ?o = u_1)\big)$ in $adoc(u)$, which produces a mapping result if and only if $(?x, b, u_1)$ matches a triple in $adoc(u)$. Moreover, every such mapping will assign value u_1 to variable $?o$. Thus the exported value in expression $\langle ?o, q_i \rangle$ would be exactly u_1. Generalizing this example one can show how lpe_m works: if there is a triple pattern in P that matches a triple, say t, in any of the documents pointed by URIs in S, then lpe_m allows us to navigate to any URI that is mentioned in t. This is the intuition behind the base case of the definition of a (c_{Match}, S, P)-reachable document. Given that $lpe^{c_{\text{Match}}} = lpe_m^*$ we obtain that $lpe^{c_{\text{Match}}}$ defines exactly the set of (URIs pointing to) documents that are (c_{Match}, S, P)-reachable. The complete formal proof can be found in the Appendix. □

5.2. Comparison with property paths under context-based query semantics

Property paths (PPs for short) were introduced in SPARQL 1.1 as a way of adding navigational power to the language [6]. PPs are a form of regular expressions that are evaluated over a single (local) RDF graph; a PP expression is used to retrieve pairs $\langle a, b \rangle$ of nodes in the graph such that there is a path from a to b whose sequence

of edge labels belongs (as a string) to the regular language defined by the expression. The syntax of PP expressions is given by the following grammar,[1] where $p, u_1, u_2, \ldots, u_k$ are URIs.

$$pe := p \mid !(u_1|u_2|\cdots|u_k) \mid pe/pe \mid pe|pe \mid pe^*.$$

A PP pattern is defined as a tuple of the form $\langle \alpha, pe, \beta \rangle$ where pe is a PP expression, and α and β are in $\mathcal{U} \cup \mathcal{L} \cup \mathcal{V}$.

In [10] the authors adapted the semantics of PP patterns so that they can be used to query the Web of Linked Data. The proposed query semantics is called *context-based semantics* [10]. To define this semantics, the authors first introduce the notion of a *context selector* for a Web of Linked Data W. This context selector is a function $C^W(\cdot)$ that given a URI $u \in \text{dom}^{\not L}(adoc)$ returns the RDF triples in $\text{data}(adoc(u))$ that have u in the subject position. Formally, for every URI $u \in \text{dom}^{\not L}(adoc)$ we have $C^W(u) = \{\langle s, p, o\rangle \in \text{data}(adoc(u)) \mid s = u\}$. To simplify the exposition, the authors extended the definition of $C^W(\cdot)$ to also handle URIs not in $\text{dom}^{\not L}(adoc)$, and literals and blank nodes. For any such RDF term a they define $C^W(a)$ as the empty set.

The context-based semantics for PPs over the Web of Linked Data in [10] is a bag semantics that follows closely the semantics for PPs defined in the normative semantics of SPARQL 1.1 [6]. Hence, both semantics use a procedure, the *ArbitraryLengthPath* procedure [6], to define the semantics of the $(\cdot)^*$ operator. It was shown in [42] that for sets semantics, the normative semantics of PPs can be defined by using standard techniques for regular expressions. To make the comparison with LDQL, in this paper we adapt the context-based semantics for PPs presented in [10] by following the techniques in [42], and consider only sets of mappings. To this end, we define a function $[\![\cdot]\!]^{\text{ctxt}}_W$ that, given a PP-pattern, returns its evaluation under context-based semantics over the Web of Linked Data W. In the definition, for a solution mapping μ and an RDF term α, we use the notation $\mu[\alpha]$ with the following meaning: $\mu[\alpha] = \mu(\alpha)$ if $\alpha \in \text{dom}(\mu)$, and $\mu[\alpha] = \alpha$ in the other case. Similarly, $\mu[\langle s, p, o\rangle] = \langle \mu[s], \mu[p], \mu[o]\rangle$. The recursive definition is as follows.

$$[\![(\alpha, p, \beta)]\!]^{\text{ctxt}}_W = \{\mu \mid \text{dom}(\mu) = \{\alpha, \beta\} \cap \mathcal{V} \text{ and } \mu[\langle \alpha, p, \beta\rangle] \in C^W(\mu[\alpha])\}$$

$$[\![(\alpha, !(u_1|\cdots|u_k), \beta)]\!]^{\text{ctxt}}_W = \{\mu \mid \text{dom}(\mu) = \{\alpha, \beta\} \cap \mathcal{V} \text{ and there exists a URI } p \text{ such that } \mu[\langle \alpha, p, \beta\rangle] \in C^W(\mu[\alpha]) \text{ and } p \notin \{u_1, \ldots, u_k\}\}$$

$$[\![(\alpha, pe_1/pe_2, \beta)]\!]^{\text{ctxt}}_W = \pi_{\{\alpha,\beta\}\cap\mathcal{V}}\big([\![(\alpha, pe_1, ?v)]\!]^{\text{ctxt}}_W \bowtie [\![(?v, pe_2, \beta)]\!]^{\text{ctxt}}_W\big)$$

$$[\![(\alpha, pe_1|pe_2, \beta)]\!]^{\text{ctxt}}_W = [\![(\alpha, pe_1, \beta)]\!]^{\text{ctxt}}_W \cup [\![(\alpha, pe_2, \beta)]\!]^{\text{ctxt}}_W$$

$$[\![(\alpha, pe^*, \beta)]\!]^{\text{ctxt}}_W = \{\mu \mid \text{dom}(\mu) = \{\alpha, \beta\} \cap \mathcal{V} \text{ and } \mu[\alpha] = \mu[\beta] \in \mathit{terms}(W)\} \cup [\![(\alpha, pe, \beta)]\!]^{\text{ctxt}}_W \cup [\![(\alpha, pe/pe, \beta)]\!]^{\text{ctxt}}_W \cup [\![(\alpha, pe/pe/pe, \beta)]\!]^{\text{ctxt}}_W \cup \cdots.$$

A *PP-based SPARQL query* [10] is an expression formed by combining PP-patterns using the standard SPARQL operators AND, UNION, OPT, FILTER and so on, following the standard semantics for these operators [41].

We next show that there exists a simple LDQL query that cannot be expressed by using the full expressive power of PP-based SPARQL queries under context-based semantics. We also show that every PP pattern can be simulated by an LDQL query, which essentially shows that PP-based SPARQL queries can be captured by LDQL queries combined with standard SPARQL operators.

Theorem 3. *There exists an LDQL query that cannot be expressed as a PP-based SPARQL query under context-based semantics. That is, there exists an LDQL query q for which there does not exist a PP-based SPARQL query P and set of URIs S such that $[\![P]\!]^{\text{ctxt}}_W = [\![q]\!]^S_W$ for every Web of Linked Data W.*

Proof. We will show that the LDQL query Q given by

$$\big(\text{SEED } \{u\}\ \big\langle\langle +, p, \_\rangle, (?x, ?x, ?x)\big\rangle\big),$$

with $u, p \in \mathcal{U}$, cannot be expressed by PPs under context-based semantics. On the contrary, assume that there exists a PP-based SPARQL query P and a set of URIs S such that for every Web of Linked Data W, we have:

$$[\![P]\!]^{\text{ctxt}}_W = [\![Q]\!]^S_W.$$

Let $u' \in \mathcal{U}$ be an arbitrary URI such that $u' \neq u$. Consider now a Web of Linked Data $W_1 = \langle D_1, adoc_1\rangle$ that consists of two documents, d_1 and d_2, such that $\text{data}(d_1) = \{(u, p, u')\}$ and $\text{data}(d_2) = \{(a, a, a)\}$, and such that $adoc(u) = d_1$ and $adoc(u') = d_2$. Moreover, consider a Web of Linked Data $W_2 = \langle D_2, adoc_2\rangle$ that also contains document d_1, and another document, d_3, such that $\text{data}(d_3) = \{(b, b, b)\}$, and such that $adoc_2(u) = d_1$ and $adoc_2(u') = d_3$. First notice that for every finite set $S \subseteq \mathcal{U}$ we have that

$$[\![Q]\!]^S_{W_1} = \{\{?x \to a\}\} \neq [\![Q]\!]^S_{W_2} = \{\{?x \to b\}\}.$$

Notice that $C^{W_1}(u) = C^{W_2}(u) = \{(u, p, u')\}$ and $C^{W_1}(u') = C^{W_2}(u') = \emptyset$. In general, we have that for every term $v \neq u$ it holds that $C^{W_1}(v) = C^{W_2}(v) = \emptyset$. This essentially shows that the context selectors C^{W_1} and C^{W_2} are equivalent. Given that the context-based semantics is based on context selectors, it is easy to prove that for every PP-based SPARQL query R we have that $[\![R]\!]^{\text{ctxt}}_{W_1} = [\![R]\!]^{\text{ctxt}}_{W_2}$. This can be done by induction on the construction of PP-based SPARQL queries. For example, the evaluation of a base PP-pattern of the form (v, p, β), with $v \in \mathcal{U}$ and $\beta \in \mathcal{U} \cup \mathcal{V}$, over W_1 is given by

$$[\![(v, p, \beta)]\!]^{\text{ctxt}}_{W_1} = \{\mu \mid \text{dom}(\mu) = \{\beta\} \cap \mathcal{V} \text{ and } \mu[\langle v, p, \beta\rangle] \in C^{W_1}(v)\},$$

which is equal to $[\![(v, p, \beta)]\!]^{\text{ctxt}}_{W_2}$ since $C^{W_1}(v) = C^{W_2}(v)$. All the other cases for the construction of property paths are equivalent. Moreover, since for the case of property path patterns the evaluation is the same over W_1 and over W_2, we have that for a general PP-based SPARQL query (using operators AND, UNION, OPT, and so on), the evaluation is also the same. Thus, we have that $[\![P]\!]^{\text{ctxt}}_{W_1} = [\![P]\!]^{\text{ctxt}}_{W_2}$ but also that $[\![Q]\!]^S_{W_1} \neq [\![Q]\!]^S_{W_2}$, which contradicts the fact that $[\![P]\!]^{\text{ctxt}}_W = [\![Q]\!]^S_W$ for every W. □

Theorem 4. *For every PP-pattern $\langle \alpha, pe, \beta\rangle$, there exists an LDQL query q such that for every Web of Linked Data W we have $[\![(\alpha, pe, \beta)]\!]^{\text{ctxt}}_W = [\![q]\!]^{\emptyset}_W$.*

Proof. We provide a translation scheme from PPs to LDQL. One major complication is that PPs can retrieve literals and, in general, values that are not in $\text{dom}^{\not L}(adoc)$, which are difficult to handle by LPEs that can only traverse URIs in $\text{dom}^{\not L}(adoc)$. This complication will become clear when presenting the details of the translation.

We begin by translating PPs of the form $(?x, pe, ?y)$ for which both subject and object are variables. Later we explain how to adapt this translation to the other cases. In the translation we associate to every PP expression r an LDQL query $Q_r(?x, ?y)$ with $?x$

[1] In [10] the reverse path construction $\hat{}pe$ is also considered. We do not consider it here as the form of navigation of these reverse paths does not represent a traversal of the link graph.

1. If $r \in \mathcal{U}$ then $Q_r(?x, ?y) = (\text{SEED } ?x\ \langle \varepsilon, (?x, r, ?y)\rangle)$.
2. If $r = !(u_1 \mid \cdots \mid u_k)$ with $u_i \in \mathcal{U}$ then $Q_r(?x, ?y)$ is defined as
$$\pi_{\{?x,?y\}}\Big(\text{SEED } ?x\ \langle \varepsilon, ((?x, ?p, ?y)\ \text{FILTER}\ (?p \neq u_1 \wedge \cdots \wedge ?p \neq u_k))\rangle\Big).$$
3. If $r = r_1/r_2$ then $Q_r(?x, ?y)$ is defined as
$$\pi_{\{?x,?y\}}(Q_{r_1}(?x, ?z)\ \text{AND}\ Q_{r_2}(?z, ?y)).$$
4. If $r = r_1|r_2$ then $Q_r(?x, ?y)$ is defined as
$$(Q_{r_1}(?x, ?y)\ \text{UNION}\ Q_{r_2}(?x, ?y)).$$
5. If $r = r_1^*$ then $Q_r(?x, ?y)$ is defined as follows. First consider the LDQL query
$$Q_\varepsilon(?x, ?y) = \pi_{\{?x,?y\}}(\text{SEED } ?f\ \langle \varepsilon, P\rangle)$$
where P is the following pattern
$$\begin{aligned} P = &((?x, ?p, ?o)\ \text{AND}\ (?y, ?p, ?o)\ \text{FILTER}\ (?x = ?y))\ \text{UNION}\\ &((?s, ?x, ?o)\ \text{AND}\ (?s, ?y, ?o)\ \text{FILTER}\ (?x = ?y))\ \text{UNION}\\ &((?s, ?p, ?x)\ \text{AND}\ (?s, ?p, ?y)\ \text{FILTER}\ (?x = ?y)).\end{aligned}$$
Now consider the LDQL query $Q_s(?v)$ defined as
$$Q_s(?v) = (\langle \varepsilon, (\text{GRAPH } ?u\ \{\ \})\rangle\ \text{AND}\ Q_{r_1}(?u, ?v)).$$
Then, query $Q_r(?x, ?y)$ is defined as
$$Q_\varepsilon(?x, ?y)\ \text{UNION}\ ((\text{SEED } ?x\ \langle (?v, Q_s(?v))^*, (\text{GRAPH } ?z\ \{\ \})\rangle)\ \text{AND}\ Q_{r_1}(?z, ?y)).$$

Fig. 2. Rules for translating a PP expression r into an LDQL query $Q_r(?x, ?y)$.

and $?y$ as free variables. The definition of $Q_r(?x, ?y)$ is by induction on the construction of PP expressions. In the construction, all the variables mentioned, besides $?x$ and $?y$, are considered as fresh variables. The rules for constructing Q_r are shown in Fig. 2.

Claim 2. *For every PP pattern of the form $\langle ?x, r, ?y\rangle$ it holds that $[\![\langle ?x, r, ?y\rangle]\!]_W^{\text{ctxt}} = [\![Q_r(?x, ?y)]\!]_W^{\emptyset}$.*

The proof of this claim can be done by induction on the construction of $Q_r(?x, ?y)$. All the details of the induction can be found in the Appendix. We just mention here some cases to give enough intuition on why the construction works. Consider the PP pattern $\langle ?x, !(u_1|\cdots|u_k), ?y\rangle$. In this case we use rule 2 in Fig. 2 and the translation is

$$\pi_{\{?x,?y\}}\big(\text{SEED}?x\,\langle \varepsilon, \big((?x, ?p, ?y)\text{FILTER} \times (?p \neq u_1 \wedge \cdots \wedge ?p \neq u_k)\big)\rangle\big).$$

In this LDQL query we are setting variable $?x$ to the seed URI from which we start our navigation. Suppose that this URI is u. We then navigate from u using LPE ε, which means that we stay at the document pointed by u, that is $adoc(u)$. Finally, with the expression $\big((?x, ?p, ?y)\text{FILTER}(?p \neq u_1 \wedge \cdots \wedge ?p \neq u_k)\big)$, we extract the triples of the from (u, a, b) in $adoc(u)$ such that a is different from all the URIs $u_1, \ldots, u_k$. Thus, a mapping $\mu = \{?x \to u, ?y \to b\}$ is a solution if there is a triple (u, a, b) in $adoc(u)$ such that $a \notin \{u_1, \ldots, u_k\}$, which is exactly the context-based semantics of $\langle ?x, !(u_1|\cdots|u_k), ?y\rangle$.

The other interesting case is the PP pattern $\langle ?x, r_1^*, ?y\rangle$, where r_1 is an arbitrary PP-expression. In this case we use rule 5 in Fig. 2. The expression r_1^* can be written as $\varepsilon|r_1^+$ and the query $Q_{r_1^*}(?x, ?y)$ handles ε and r_1^+ separately. For the case of ε we use $Q_\varepsilon(?x, ?y)$, which essentially matches when the values assigned to $?x$ and $?y$ are the same (arbitrary) value. More interesting is the case of r_1^+. For this case, we first define query $Q_s(?v)$ in Fig. 2 given by $\big(\langle \varepsilon, (\text{GRAPH}?u\ \{\ \})\rangle \text{AND} Q_{r_1}(?u, ?v)\big)$. If we assume that $Q_{r_1}(?u, ?v)$ is correct, then $Q_s(?v)$, when evaluated from a seed URI u, gives as result all the values (which can be URIs or literals) that are *reachable* from u by following expression r_1 according to the context-based semantics of PPs. The portion of the query given by $\langle \varepsilon, (\text{GRAPH}?u\ \{\ \})\rangle$ is only ensuring that $?u$ is always bound to a URI which is in $\text{dom}^{\not\perp}(adoc)$. Now consider the expression $\langle ?v, Q_s(?v)\rangle^*$. This expression is essentially *repeating* several times $Q_s(?v)$; if we start with a seed URI u and we evaluate $\langle ?v, Q_s(?v)\rangle$, we obtain in $?v$ a URI in $\text{dom}^{\not\perp}(adoc)$, say u', that is reachable from u by following r_1, and by the semantics of the construction $\langle ?v, q\rangle$ in LDQL, this URI u' is the one used to continue the navigation afterwards. Thus, $\langle ?v, Q_s(?v)\rangle^*$, when evaluated from a seed URI u, gives the set of all URIs $\text{dom}^{\not\perp}(adoc)$ that are reachable from u following 0 or more copies of r_1. Now consider the part of $Q_{r_1^*}(?x, ?y)$ given by

$$(\text{SEED}?x\ \langle (?v, Q_s(?v))^*, (\text{GRAPH}?z\ \{\ \}).$$

From the discussion above, we note that this query is setting variable $?x$ as the seed URI, and variable $?z$ as the URI reached after following 0 or more copies of r_1 from $?x$. Finally, the last part of $Q_{r_1^*}(?x, ?y)$ is a join with $Q_{r_1}(?z, ?y)$, which essentially performs the last step and retrieves (and stores in $?y$) all the values that can be reached from $?z$ by following r_1. Notice that in this last case the value assigned to $?y$ can be an arbitrary URI (not necessarily in $\text{dom}^{\not\perp}(adoc)$) or even a literal. The detailed proof by induction can be found in the Appendix.

We have shown how to construct an equivalent LDQL query for every PP pattern of the form $\langle ?x, r, ?y\rangle$. For PP patterns that do not have two variables we need a slightly different construction, in particular for the case in which $(\cdot)^*$ is used. Consider a PP pattern $\langle \alpha, r, \beta\rangle$ where α is a URI or variable, and β is a URI, variable, or literal. Then, for the cases (i) $r = p \in \mathcal{U}$, (ii) $r = !(u_1|\cdots|u_k)$, (iii) $r = r_1/r_2$, and (iv) $r = r_1|r_2$, we construct an LDQL query $Q_r(\alpha, \beta)$ that is obtained from $Q_r(?x, ?y)$ by replacing all occurrences of $?x$ by α and all occurrences of $?y$ by β.

We now consider the case of $r = r_1^*$. First, without loss of generality, we assume that r_1 is not of the form $(a^*)^*$. This is

possible given that an expression of the form $(a^*)^*$ is equivalent to a^*. In this case we also need to use the BIND operator of SPARQL. The semantics of BIND is very simple and is as follows: for a URI u the query BIND(uAS?y) binds variable ?y to u, that is the result consists of a single solution, namely: $\{?y \to u\}$. To continue with our construction, for a PP pattern $\langle u, r, ?y\rangle$ we construct a query $P_r^u(?y)$ as

$$\langle \varepsilon, \mathsf{BIND}(u\mathsf{AS}?y)\rangle \mathsf{UNION}\big((\mathsf{SEED}\{u\}\ \langle (?v, Q_s(?v))^*, (\mathsf{GRAPH}?z\ \{\ \}))) \mathsf{AND} Q_{r_1}(?z, ?y)\big).$$

The part $\langle \varepsilon, \mathsf{BIND}(u\mathsf{AS}?y)\rangle$ handles the ε case (0 repetitions of r_1). The other part is similar to the case of two variables but fixing the initial URI to u. For a PP pattern $\langle ?x, r, v\rangle$ we construct a similar LDQL query $S_r^v(?x)$ as

$$\langle \varepsilon, \mathsf{BIND}(v\mathsf{AS}?x)\rangle \mathsf{UNION}\big((\mathsf{SEED}?x\ \langle (?v, Q_s(?v))^*, (\mathsf{GRAPH}?z\ \{\ \}))) \mathsf{AND} Q_{r_1}(?z, v)\big).$$

Finally, for a PP pattern $\langle u, r, v\rangle$ we construct an LDQL query $U_r^{u,v}$ as

$$\pi_\emptyset\langle \varepsilon, (\mathsf{BIND}(u\mathsf{AS}?x)\mathsf{ANDBIND}(v\mathsf{AS}?y))\mathsf{FILTER}(?x = ?y)\rangle \mathsf{UNION}\big((\mathsf{SEED}\{u\}\ \langle (?v, Q_s(?v))^*, (\mathsf{GRAPH}?z\ \{\ \}))) \mathsf{AND} Q_{r_1}(?z, v)\big).$$

Finally consider a PP pattern (ℓ, r, β), where ℓ is a literal. For the cases $r = p \in \mathcal{U}$ and $r = !(u_1|\cdots|u_k)$ we should translate it into an unsatisfiable query. One way of obtaining that query is, for example, with an expression

$$\langle \varepsilon, (\mathsf{BIND}(\ell\mathsf{AS}?x)\mathsf{ANDBIND}(\ell\mathsf{AS}?y))\mathsf{FILTER}(?x \neq ?y)\rangle. \tag{6}$$

For the cases $r = r_1/r_2$ and $r = r_1|r_2$ we follow the same construction as if ℓ was a URI but with (6) as base case. For the case of $r = r_1^*$, we only have to consider the ε case, as PPs cannot actually navigate from literal values. Thus, if β is a variable ?y, we consider the following query

$$\langle \varepsilon, \mathsf{BIND}(\ell\mathsf{AS}?y)\rangle,$$

and if β is a URI or literal the query is

$$\pi_\emptyset\langle \varepsilon, (\mathsf{BIND}(\ell\mathsf{AS}?x)\mathsf{ANDBIND}(\beta\mathsf{AS}?y))\mathsf{FILTER}(?x = ?y)\rangle.$$

The correctness of the complete translation for PP patterns of the form $\langle \alpha, r, \beta\rangle$ can be proved along the same lines as for the case of PP pattern $\langle ?x, r, ?y\rangle$. □

5.3. Comparison with SPARQL under full-web query semantics

In addition to the aforementioned reachability-based semantics and the context-based semantics, the authors of [8] and of [10] define a *full-Web semantics* for answering SPARQL queries over a Web of Linked Data. By this semantics, a SPARQL pattern has to be evaluated over the union of all the triples in all the documents in a Web of Linked Data. As shown by the authors in [8], this semantics is mostly of theoretical interests as there cannot exist a system that guarantees to compute it using an algorithm that both terminates and returns complete query results. Nonetheless, in this section we show that LDQL is powerful enough to capture this full-Web semantics.

Let $W = \langle D, adoc\rangle$ be a Web of Linked Data and consider the set G_{All} of RDF triples constructed as

$$G_{\text{All}}^W = \bigcup_{d \in D} \text{data}(d).$$

The evaluation of a SPARQL graph pattern P over W under full-Web semantics is defined as the evaluation of P over G_{All} according to the normative SPARQL semantics [8,10]. Formally, let $\mathfrak{D}_{\text{All}}^W$ be the RDF dataset that contains G_{All}^W as the default graph and no named graph. Recall that given an RDF dataset $\mathfrak{D}$, and a SPARQL graph pattern P, we denote by $[\![P]\!]^{\mathfrak{D}}$ the evaluation of P over dataset $\mathfrak{D}$. Thus, the evaluation of P over W under full-Web semantics is just $[\![P]\!]^{\mathfrak{D}_{\text{All}}^W}$.

The next two results show that LDQL is strictly more expressive than SPARQL graph patterns under full-Web semantics. That is, not only is LDQL powerful enough to capture full-Web semantics of SPARQL queries (Theorem 5), there also exists an LDQL query that cannot be expressed as a SPARQL pattern under full-Web semantics (Theorem 6).

Theorem 5. *For every SPARQL graph pattern P there exists an LDQL query q such that for every Web of Linked Data W, it holds that $[\![P]\!]^{\mathfrak{D}_{All}^W} = [\![q]\!]_W^{\{u_{\text{ctx}}\}}$, where u_{ctx} is an arbitrary URI.*

Proof. We begin by constructing an LPE $lpe_{\texttt{All}}$ that provides access to all documents in an arbitrary Web of Linked Data $W = \langle D, adoc\rangle$. To this end, first consider the LDQL query $Q_{\text{All}}(?v)$ defined as

$$Q_{\texttt{All}}(?v) = (\mathsf{SEED}?v\ \langle \varepsilon, \{\ \}\rangle).$$

Essentially, this query considers all possible URIs in $\text{dom}^{\not\perp}(adoc)$, and binds each of them to variable ?v as follows. Operator SEED first defines a navigation from an arbitrary URI, say u. Then, the subquery $\langle \varepsilon, \{\ \}\rangle$ navigates from u by using LPE ε; thus, it stays in the document $d = adoc(u)$, and evaluates the empty pattern $\{\ \}$ over data(d). Therefore, the mapping $\{?v \to u\}$ is in $[\![Q_{\text{All}}(?v)]\!]_W^{\{u_{\text{ctx}}\}}$ for every possible URI $u \in \text{dom}^{\not\perp}(adoc)$, where u_{ctx} is an arbitrary URI.

Then, we define LPE lpe_{All} as $lpe_{\texttt{All}} = \langle ?v, Q_{\text{all}}(?v)\rangle$. Now let u_{ctx} be an arbitrary URI not mentioned in P. By the semantics of LPEs, we have that

$$\begin{aligned}[\![lpe_{\texttt{All}}]\!]_W^{u_{\text{ctx}}} &= [\![\langle ?v, Q_{\text{All}}(?v)\rangle]\!]_W^{u_{\text{ctx}}} \\ &= \big\{u \in \mathcal{U} \mid \text{there exists } \mu \in [\![Q_{\text{All}}(?v)]\!]_W^{\{u_{\text{ctx}}\}} \\ &\qquad \text{such that } \mu(?v) = u\big\} \\ &= \text{dom}^{\not\perp}(adoc).\end{aligned}$$

Recall that for every document $d \in D$, there exists a URI $u \in \mathcal{U}$ such that $adoc(u) = d$. Consequently, $lpe_{\texttt{All}}$ provides access to all documents in a Web of Linked Data. Formally we have that $\mathfrak{D}' = \text{dataset}_W\left([\![lpe]\!]_W^{u_{\text{ctx}}}\right)$ is a dataset that has G_{All}^W as default graph and $\langle u_{\text{ctx}}, G_{\text{All}}^W\rangle$ as named graph. Notice that the only difference between $\mathfrak{D}'$ and $\mathfrak{D}_{\text{All}}^W$ is the named graph $\langle u_{\text{ctx}}, G_{\text{All}}^W\rangle$.

Consider now the LDQL query $q = \langle lpe_{\texttt{All}}, P'\rangle$ where P' is obtained from P by replacing every sub pattern of the form (GRAPH?x R) by an unsatisfiable pattern P_{un}. Notice that by the semantics of LDQL we have $[\![q]\!]_W^{\{u_{\text{ctx}}\}} = [\![P']\!]^{\mathfrak{D}'}$. It is not difficult to show that $[\![P']\!]^{\mathfrak{D}'} = [\![P]\!]^{\mathfrak{D}_{\text{All}}^W}$. In particular, it is clear that every sub pattern of P that is not of the form (GRAPHu R) or (GRAPH?x R) has the same evaluation under datasets $\mathfrak{D}'$ and $\mathfrak{D}_{\text{All}}^W$ as both datasets have the same default graph. For the case of a pattern of the form (GRAPHu R), in both datasets the evaluation of the pattern is the empty set because the only named graph has u_{ctx} as name (where u_{ctx} can be selected to be different from u). Finally, for the case of a pattern of the form (GRAPH?x R), in dataset $\mathfrak{D}_{\text{All}}^W$ the evaluation is clearly empty (as there is no named graph in $\mathfrak{D}_{\text{All}}^W$), which is exactly the evaluation of P_{un} in dataset $\mathfrak{D}'$. Thus, we have shown that $[\![P']\!]^{\mathfrak{D}'} = [\![P]\!]^{\mathfrak{D}_{\text{All}}^W}$ and thus $[\![q]\!]_W^{\{u_{\text{ctx}}\}} = [\![P]\!]^{\mathfrak{D}_{\text{All}}^W}$. □

Theorem 6. *There exists an LDQL query q for which there does not exist a SPARQL graph pattern P and set S of URIs such that $[\![q]\!]_W^S = [\![P]\!]^{\mathfrak{D}_{All}^W}$ for every Web of Linked Data W.*

Proof. Let $W = \langle D, adoc \rangle$ be a Web of Linked Data and let u and u' be two URIs such that $u \in \text{dom}^{\not\perp}(adoc)$ and $u' \notin \text{dom}^{\not\perp}(adoc)$. Consider now another Web of Linked Data $W' = (D', adoc')$ such that $D = D'$, and $adoc'$ is almost exactly $adoc$, with the only difference that $u' \in \text{dom}^{\not\perp}(adoc')$, $u \notin \text{dom}^{\not\perp}(adoc')$, and $adoc'(u') = adoc(u)$. Notice that $\mathfrak{D}^{W}_{\text{All}} = \mathfrak{D}^{W'}_{\text{All}}$ as in both Webs the documents (and thus the data) are the same. This implies that for every SPARQL pattern P we have that the evaluation of P under full-Web semantics over W is the same as over W'.

Now consider the LDQL query $q = (\text{SEED}u' \ \langle \varepsilon, \{\ \} \rangle)$. It is clear that under W' the evaluation of q gives as result a set containing the empty mapping. On the other hand, the evaluation of q over W gives the empty set as there is no document associated with u'. Therefore, q is not expressible in SPARQL under full-Web semantics, as every pattern P gives the same result when evaluated under full-Web semantics over W and W', respectively, while q gives different results. □

A final comment is in order. In our definition of LDQL we have considered as a base case the construction $\langle lpe, P \rangle$ where P is a SPARQL 1.0 graph pattern. In the full-Web semantics mentioned in [10], the authors consider PP-based SPARQL queries, that is, SPARQL 1.1 graph patterns constructed by combining PP-patterns using the standard SPARQL operators AND, UNION, OPT, and FILTER (cf. Section 5.2). Thus, to completely capture the full-Web semantics in [10] we should also allow SPARQL 1.1 graph patterns in the base case of LDQL. Similarly, in [43] (which is an extended version of [10]) a reachability-based semantics for PP-based SPARQL 1.1 patterns is proposed. This semantics can also be captured in our framework by just considering SPARQL 1.1 patterns in the base case of LDQL expressions and using the construction shown in Section 5.1. We have decided to use only SPARQL 1.0 patterns in our core language as it already has the necessary features that we needed in the expressiveness results, namely the operators AND, UNION, GRAPH, and so on. Nevertheless, we stress that, in practice, any version of SPARQL patterns (either 1.0, 1.1, or even subsequent versions) can be plugged into the LDQL base case.

5.4. Comparison with NautiLOD

NautiLOD is a navigation language to traverse Linked Data on the WWW and to perform actions (such as sending emails) during the traversal [13]. We compare LDQL with the navigational core of NautiLOD, which excludes action rules and represents the output of a navigation as a set of URIs, which is called "NautiLOD semantics returning set of nodes" in [13]. It should be noticed that [13] introduces alternative semantics, called "NautiLOD semantics returning Web fragments", in which queries can essentially output the portion of the Web that was traversed while evaluating the expression, including URIs and links. The idea of retrieving Web fragments is very interesting and we think that it can also be adapted to LDQL. We leave such an adaptation, as well as a full comparison with the Web-fragment semantics of NautiLOD, for future work.

The syntax of NautiLOD expressions (without actions) is given by the following grammar (where $p \in \mathcal{U}$ and P is a SPARQL graph pattern).

$$ne := p \mid p\hat{} \mid \langle \_ \rangle \mid ne/ne \mid ne|ne \mid ne^* \mid ne[(\text{ASK}P)]$$

In terms of our data model,[2] the semantics of NautiLOD expressions that returns sets of URIs over a Web of Linked Data $W = \langle D, adoc \rangle$ from URI $u \in \text{dom}^{\not\perp}(adoc)$ is defined recursively as follows.

$$[\![p]\!]^u_W = \{u' \mid \langle u, p, u' \rangle \in \text{data}(adoc(u))\}$$
$$[\![p\hat{}]\!]^u_W = \{u' \mid \langle u', p, u \rangle \in \text{data}(adoc(u))\}$$
$$[\![\langle \_ \rangle]\!]^u_W = \{u' \mid \langle u, p, u' \rangle \in \text{data}(adoc(u)) \text{ for some } p \in \mathcal{U}\}$$
$$[\![ne_1/ne_2]\!]^u_W = \{u'' \mid u'' \in [\![ne_2]\!]^{u'}_W \text{ for some } u' \in [\![ne_1]\!]^u_W \cap \text{dom}^{\not\perp}(adoc)\}$$
$$[\![ne_1|ne_2]\!]^u_W = [\![ne_1]\!]^u_W \cup [\![ne_2]\!]^u_W$$
$$[\![ne^*]\!]^u_W = \{u\} \cup [\![ne]\!]^u_W \cup [\![ne/ne]\!]^u_W \cup [\![ne/ne/ne]\!]^u_W \cup \cdots$$
$$[\![ne[(\text{ASK}P)]]\!]^u_W = \{u' \mid u' \in [\![ne]\!]^u_W \cap \text{dom}^{\not\perp}(adoc) \text{ and } [\![P]\!]_{\text{data}(adoc(u'))} \neq \emptyset\}$$

We next compare the expressive power of LDQL and NautiLOD. Notice that the evaluation of a NautiLOD expression is a set of URIs, whereas the evaluation of an LDQL query is a set of mappings. Thus, to state our results formally we compare NautiLOD with LDQL queries that have a single *free variable*. Let $q(?x)$ be an LDQL query with $?x$ as free variable. We say that $q(?x)$ and a NautiLOD expression ne are equivalent if for every Web of Linked Data $W = \langle D, adoc \rangle$ and every pair of URIs u, u' such that $u \in \text{dom}^{\not\perp}(adoc)$, it holds that $u' \in [\![ne]\!]^u_W$ if and only if $\{?x \mapsto u'\} \in [\![q(?x)]\!]^{\{u\}}_W$.

We first prove that LDQL is strictly more expressive than NautiLOD. Recall that NautiLOD can only express paths; a combination of such paths via SPARQL operators is not allowed. Thus, it is easy to prove that NautiLOD cannot express operators such as SEED, AND, or UNION, which are allowed natively in LDQL. However, in this paper we make a stronger claim: Instead of using the mentioned operators, we will prove that there exists a basic LDQL query that cannot be represented using NautiLOD expressions.

Theorem 7. *There exists a basic LDQL query $Q(?x)$ with $?x$ as free variable that does not use SPARQL operators (*AND, OPT, UNION, *and so on) nor the operator* SEED, *and for which there does not exist a NautiLOD expression ne such that $[\![n]\!]^u_W = [\![Q(?x)]\!]^{\{u\}}_W$ for every Web of Linked Data $W = \langle D, adoc \rangle$ and URI $u \in \text{dom}^{\not\perp}(adoc)$.*

Proof. Consider the LDQL query $Q(?x)$ given by

$$\big\langle \langle +, p, \_ \rangle, (?x, ?x, ?x) \big\rangle$$

with $p \in \mathcal{U}$. Now assume that there exists a NautiLOD expression n such that

$$[\![n]\!]^v_W = [\![Q(?x)]\!]^{\{v\}}_W$$

for every Web of Linked Data W and $v \in \text{dom}^{\not\perp}(adoc)$. Let u, u', a, b be different URIs in $\mathcal{U}$ that are not mentioned in n. Consider now a Web of Linked Data $W_1 = \langle D_1, adoc_1 \rangle$ that consists of two documents, d_1 and d_2, such that $\text{data}(d_1) = \{(u, p, u')\}$ and $\text{data}(d_2) = \{(a, a, a)\}$, and such that $adoc(u) = d_1$ and $adoc(u') = d_2$. Moreover, consider another Web of Linked Data, $W_2 = \langle D_2, adoc_2 \rangle$, that also contains document d_1, and another document, d_3, such that $\text{data}(d_3) = \{(b, b, b)\}$, and such that $adoc(u) = d_1$ and $adoc(u') = d_3$. First notice that

$$[\![Q(?x)]\!]^{\{u\}}_{W_1} = \{\{?x \to a\}\} \neq [\![Q(?x)]\!]^{\{u\}}_{W_2} = \{\{?x \to b\}\}.$$

We now prove that $[\![n]\!]^u_{W_1} = [\![n]\!]^u_{W_2}$, which is a contradiction. To prove this, we show that for every subexpression e of n, and for every possible URI v, it holds that $[\![e]\!]^v_{W_1} = [\![e]\!]^v_{W_2}$. First notice that for both Webs, $\text{dom}^{\not\perp}(adoc_1)$ and $\text{dom}^{\not\perp}(adoc_2)$ contain only two URIs, namely, u and u'. Thus, we only have two reason for the cases in which $v = u$ or $v = u'$. We proceed by induction.

[2] In [13], all URIs have an assigned set of RDF triples (which may be empty). Hence, the authors implicitly assume that every URI is in $\text{dom}^{\not\perp}(adoc)$. In our data model one can have URIs not in $\text{dom}^{\not\perp}(adoc)$. Hence, to properly capture the semantics of NautiLOD in terms of our data model we have to introduce conditions of the form "$u' \in \text{dom}^{\not\perp}(adoc)$".

- Assume that $e = r \in \mathcal{U}$. Given that in W_1 and W_2 the URI u is associated with the same document (document d_1), then $[\![r]\!]^u_{W_1} = [\![r]\!]^u_{W_2}$. Moreover, given that $r \neq a$ and $r \neq b$ (recall that n does not mention a or b), we have that $[\![r]\!]^{u'}_{W_1} = [\![r]\!]^{u'}_{W_2} = \emptyset$.
- Assume that $e = r^\wedge$ with $r \in \mathcal{U}$. Exactly the same argument as the above case applies.
- Assume that $e = \langle \_ \rangle$. For the same reason as in the above two cases we have that $[\![r]\!]^u_{W_1} = [\![r]\!]^u_{W_2}$. Now consider $[\![\langle \_ \rangle]\!]^{u'}_{W_1}$. Then, we have that URI v is in $[\![\langle \_ \rangle]\!]^{u'}_{W_1}$ if and only if there exists some p such that $(u', p, v) \in \text{data}(adoc(u'))$, but the only triple in $\text{data}(adoc(u'))$ is (a, a, a) and since $a \neq u'$ we have that $[\![\langle \_ \rangle]\!]^{u'}_{W_1} = \emptyset$. For a similar reason we obtain that $[\![\langle \_ \rangle]\!]^{u'}_{W_2} = \emptyset$, completing this part of the proof.
- The cases (i) $e = r_1/r_2$, (ii) $e = r_1|r_2$, and (iii) $e = r^*$ follow from the base cases proved above.
- Assume $e = r[(\text{ASK}P)]$. By definition we have that

  $[\![r[(\text{ASK}P)]]\!]^v_W = \{v' \mid v' \in [\![r]\!]^v_W \cap \text{dom}^{\not\perp}(adoc)$ and $[\![P]\!]_{\text{data}(adoc(v'))} \neq \emptyset\}$.

  By the induction hypothesis we have that $[\![r]\!]^v_{W_1} = [\![r]\!]^v_{W_2}$ for $v = u, u'$. Thus, we only need to prove that the evaluation of P is always the same. Given that $\text{data}(adoc(u))$ is the same document in W_1 and W_2, we have that for u the property holds. Now consider $[\![P]\!]_{\text{data}(d_2)}$ and $[\![P]\!]_{\text{data}(d_3)}$ with $\text{data}(d_2) = \{(a, a, a)\}$ and $\text{data}(d_3) = \{(b, b, b)\}$. Recall that P does not mention a or b, thus we have that if $\mu \in [\![P]\!]_{d_2}$ then the mapping μ' obtained from μ by replacing every occurrence of a by b, is in $[\![P]\!]_{d_3}$, and vice versa. Thus, we have that $[\![P]\!]_{d_2} = \emptyset$ if and only if $[\![P]\!]_{d_3} = \emptyset$. This proves that $[\![r[(\text{ASK}P)]]\!]^v_{W_1} = [\![r[(\text{ASK}P)]]\!]^v_{W_2}$ for $v = u, u'$.

We have finished the proof that $[\![n]\!]^u_{W_1} = [\![n]\!]^u_{W_2}$, which contradicts the fact that n is equivalent to $Q(?x)$. □

Theorem 8. *For every NautiLOD expression ne, there exists an LDQL query $Q(?x)$, with ?x a free variable, that is equivalent to ne; that is, for every Web of Linked Data $W = \langle D, adoc \rangle$ and every $u \in \text{dom}^{\not\perp}(adoc)$, we have $[\![n]\!]^u_W = [\![q]\!]^{\{u\}}_W$.*

Proof. The outlook of the proof is as follows. The proof begins with a simple translation that replaces every $p \in \mathcal{U}$ in a NautiLOD expression by a link pattern $\langle +, p, \_ \rangle$. For instance, the expression p_1/p_2^* is translated into $\langle +, p_1, \_ \rangle / \langle +, p_2, \_ \rangle^*$. The translation of $\langle \_ \rangle$ and of $[(\text{ASK}P)]$ needs the LPE construction $\langle ?v, q \rangle$. The complete translation poses several other complications. In particular, the last step of NautiLOD expressions must be translated by using a SPARQL pattern and not an LPE. For this we use the following property. Given a regular expression r that does not generate the empty word, one can always write r as $r_1/a_1| \cdots |r_k/a_k$ where the a_i's are base symbols of the alphabet. Thus, we can translate r by using LPEs to translate the r_i's as outlined above; next, translate the a_i's by using a method similar to the proof of Theorem 4, and finally use UNION for |.

For the complete proof we proceed by an induction that shows how to translate every possible NautiLOD expression. The translation consists of two parts. We first define the following function $\text{trans}_N(\cdot)$ that, given a NautiLOD expression, produces an LPE.

$$\begin{aligned}
\text{trans}_N(p) &= \langle +, p, _ \rangle \\
\text{trans}_N(p^\wedge) &= \langle _, p, + \rangle \\
\text{trans}_N(\langle _ \rangle) &= \big(?x, \langle \varepsilon, (\text{GRAPH}?u\ (?u, ?p, ?x)) \rangle\big) \\
\text{trans}_N(n_1/n_2) &= \text{trans}_N(n_1)/\text{trans}_N(n_2) \\
\text{trans}_N(n_1|n_2) &= \text{trans}_N(n_1)|\text{trans}_N(n_2) \\
\text{trans}_N(n^*) &= \text{trans}_N(n)^* \\
\text{trans}_N(n[(\text{ASK}P)]) &= \text{trans}_N(n)/[(?x, \langle \varepsilon, (\text{GRAPH}?x\ P) \rangle]
\end{aligned}$$

Before presenting the complete translations, we prove the following result. Let n be a NautiLOD expression, then for every Web of Linked Data $W = \langle D, adoc \rangle$ and URIs $u, v \in \text{dom}^{\not\perp}(adoc)$ we have that

$$v \in [\![n]\!]^u_W \quad \text{if and only if} \quad v \in [\![\text{trans}_N(n)]\!]^{\{u\}}_W .$$

The proof is by induction on the construction of the NautiLOD expression.

- For the case of $p \in \mathcal{U}$ we have that

  $[\![p]\!]^u_W = \{u' \mid (u, p, u') \in \text{data}(adoc(u))\}$.

  Notice that $v \in \text{dom}^{\not\perp}(adoc)$ and $v \in [\![p]\!]^u_W$, if and only if there is a link from document $adoc(u)$ to document $adoc(v)$ that matches $\langle +, p, \_ \rangle$. This happens, if and only if $v \in [\![\langle +, p, \_ \rangle]\!]^{\{u\}}_W$, which is what we wanted to prove.
- The case for $p^\wedge$ is similar but using $\langle \_, p, + \rangle$.
- The case for $\langle \_ \rangle$. Note that $v \in [\![\langle \_ \rangle]\!]^u_W$ if and only if there exists a $p \in \mathcal{U}$ such that $(u, p, v) \in \text{data}(adoc(u))$. On the other hand, we have that $v \in [\![\text{trans}_N(\langle \_ \rangle)]\!]^{\{u\}}_W = [\![\big(?x, \langle \varepsilon, (\text{GRAPH}?u\ (?u, ?p, ?x)) \rangle\big)]\!]^{\{u\}}_W$ if and only if $v \in [\![\pi_{?x}(\text{GRAPH}?u\ (?u, ?p, ?x))]\!]^{\mathcal{D}}$ where $\mathcal{D}$ is the dataset that is given as $\mathcal{D} = \{\text{data}(adoc(u)), \langle u, \text{data}(adoc(u)) \rangle\}$. Thus, $v \in [\![\text{trans}_N(\langle \_ \rangle)]\!]^{\{u\}}_W$ if and only if there exists p such that $(u, p, v) \in \text{data}(adoc(u))$. This proves the desired property.
- For the case of an expression n_1/n_2, we have that URI v in $\text{dom}^{\not\perp}(adoc)$ is in $[\![n_1/n_2]\!]^u_W$ if and only if there exists a URI $v' \in \text{dom}^{\not\perp}(adoc)$ such that $v' \in [\![n_1]\!]^u_W$ and $v \in [\![n_2]\!]^{v'}_W$. Then, we can apply or induction hypothesis and obtain that $v \in [\![n_1/n_2]\!]^u_W$ if and only if $v' \in [\![\text{trans}_N(n_1)]\!]^{\{u\}}_W$ and $v \in [\![\text{trans}_N(n_2)]\!]^{\{v'\}}_W$, and thus $v \in [\![\text{trans}_N(n_1/n_2)]\!]^{\{u\}}_W$.
- Cases $n_1|n_2$ and n^* are direct from the definition of NautiLOD and LDQL.
- For the case of expression $n[(\text{ASK}P)]$ we have that $v \in [\![n[(\text{ASK}P)]]\!]^u_W$ if and only if $v \in [\![n]\!]^u_W$, $v \in \text{dom}^{\not\perp}(adoc)$, and $[\![P]\!]_{\text{data}(adoc(v))} \neq \emptyset$. On the other hand, we have that $v \in [\![\text{trans}_N(n[(\text{ASK}P)])]\!]^{\{u\}}_W$ if and only if

  $v \in [\![\text{trans}_N(n)/[(?x, \langle \varepsilon, (\text{GRAPH}?x\ P) \rangle]]\!]^{\{u\}}_W$.

  This happens if and only if there exists a $v' \in [\![\text{trans}_N(n)]\!]^{\{u\}}_W$ such that $v \in [\![[(?x, \langle \varepsilon, (\text{GRAPH}?x\ P) \rangle]]\!]^{\{v'\}}_W$. From this property and the semantics of $[\cdot]$ in LDQL, we have that $v = v'$ and $[\![(?x, \langle \varepsilon, (\text{GRAPH}?x\ P) \rangle]\!]^{\{v\}}_W \neq \emptyset$. The last holds if and only if $[\![\pi_{?x}(\text{GRAPH}?x\ P)]\!]^{\mathcal{D}} \neq \emptyset$, with $\mathcal{D}$ the RDF dataset $\{\text{data}(adoc(v)), \langle v, \text{data}(adoc(v)) \rangle\}$. As a consequence, we have that $v \in [\![\text{trans}_N(n[(\text{ASK}P)])]\!]^{\{u\}}_W$ if and only if $v \in [\![\text{trans}_N(n)]\!]^{\{u\}}_W$ and $[\![P]\!]_{\text{data}(adoc(u))} \neq \emptyset$. Applying our induction hypothesis, we have $v \in [\![n]\!]^u_W$ and $[\![P]\!]_{\text{data}(adoc(u))} \neq \emptyset$, which is exactly what we needed to prove.

Notice that the hypothesis that $v \in \text{dom}^{\not\perp}(adoc)$ was fundamental to prove the previous result. Nevertheless, the output of a NautiLOD expression can be a URI not in $\text{dom}^{\not\perp}(adoc)$ or even a literal. So, we need to do a different translation in general. We now use $\text{trans}_N(\cdot)$ to translate a general NautiLOD expression. Given a NautiLOD expression n, we have two cases.

Assume first that n, as a regular expression, does not produce the empty string. Then, by using regular language results, we know that we can write an equivalent expression n' of the form

$$n_1/e_1 \mid \cdots \mid n_k/e_k \mid m_1[(\text{ASK}P_1)] \mid \cdots \mid m_\ell[(\text{ASK}P_\ell)]$$

where every n_i and m_j is a NautiLOD expression, and every e_i is either of the form p, or $p^\wedge$, or $\langle \_ \rangle$. We now are ready to produce an

LDQL query $Q_n(?x)$ that is equivalent to n. The query is constructed as follows.

$$Q_n(?x) = \pi_{\{?x\}}\Big(\langle \text{trans}_N(n_1), Q_1\rangle \text{UNION} \cdots \text{UNION}$$
$$\langle \text{trans}_N(n_k), Q_k\rangle \text{UNION}$$
$$\langle \text{trans}_N(m_1), (\text{GRAPH} ?x\ P_1)\rangle \text{UNION} \cdots$$
$$\text{UNION} \langle \text{trans}_N(m_\ell), (\text{GRAPH} ?x\ P_\ell)\rangle\Big),$$

where every graph pattern Q_i depends on the form of e_i $(1 \leq i \leq k)$; that is,

- $Q_i = (\text{GRAPH} ?u\ (?u, p, ?x))$ if $e_i = p$,
- $Q_i = (\text{GRAPH} ?u\ (?x, p, ?u))$ if $e_i = p^{\wedge}$, and
- $Q_i = (\text{GRAPH} ?u\ (?u, ?p, ?x))$ if $e_i = \langle \_ \rangle$.

To prove the correctness of our construction, assume that $v \in [\![n]\!]^u_W$. Then, we know that $v \in [\![n_i/e_i]\!]^u_W$ for some $i \in \{1, \ldots, k\}$ or $v \in [\![m_i[(\text{ASK}P_i)]]\!]^u_W$ for some $i \in \{1, \ldots, \ell\}$. If $v \in [\![n_i/e_i]\!]^u_W$, we know that there exists a URI v' such that $v' \in [\![n_i]\!]^u_W$ and $v \in [\![e_i]\!]^{v'}_W$. Notice that, since $v \in [\![e_i]\!]^{v'}_W$, and e_i is either p, or $p^{\wedge}$, or $\langle \_ \rangle$ then we know that v' is in $\text{dom}^{\not\perp}(adoc)$. Thus, we can use our previous result to conclude from $v' \in [\![n_i]\!]^u_W$ that $v' \in [\![\text{trans}_N(n_i)]\!]^{\{u\}}_W$. Now, if $e_i = p$, then from $v \in [\![e_i]\!]^{v'}_W$ we conclude that $(v', p, v) \in \text{data}(adoc(v'))$. Therefore, $[\![(?u, p, ?x)]\!]_{\text{data}(adoc(v'))}$ contains the mapping $\mu = \{?u \to v', ?x \to v\}$ and, thus, $\mu \in [\![(\text{GRAPH} ?u\ (?u, p, ?x))]\!]^{\mathcal{D}}$ where $\mathcal{D} = \{\text{data}(adoc(v')), \langle v', \text{data}(adoc(v'))\rangle\}$. Given that $v' \in [\![\text{trans}_N(n_i)]\!]^{\{u\}}_W$, we have that

$$\mu = \{?u \to v', ?x \to v\} \in [\![\langle \text{trans}_N(n_i), Q_i\rangle]\!]^{\{u\}}_W .$$

Finally, given that $Q_n(?x)$ only keeps the $?x$ variable, we have that $\{?x \to v\}$ is in $[\![Q_n(?x)]\!]^{\{u\}}_W$, which is what we wanted to show. For the cases of $e_i = p^{\wedge}$ and $e_i = \langle \_ \rangle$, the proof is the essentially the same.

Now assume that $v \in [\![m_i[(\text{ASK}P_i)]]\!]^u_W$ for some $i \in \{1, \ldots, \ell\}$, which implies that $v \in [\![m_i]\!]^u_W$ and $[\![P_i]\!]_{\text{data}(adoc(v))} \neq \emptyset$. By the semantics of NautiLOD, we have that v is in $\text{dom}^{\not\perp}(adoc)$ (otherwise we could not have been able to evaluate P), and thus we can apply our result above to obtain that $v \in [\![\text{trans}_N(m_i)]\!]^{\{u\}}_W$. Now, given that $[\![P_i]\!]_{\text{data}(adoc(v))} \neq \emptyset$ we have that $[\![(\text{GRAPH} ?x\ P_i)]\!]^{\mathcal{D}} \neq \emptyset$ where $\mathcal{D} = \{\text{data}(adoc(v)), \langle v, \text{data}(adoc(v))\rangle\}$. Moreover, for every mapping μ in $[\![(\text{GRAPH} ?x\ P_i)]\!]^{\mathcal{D}}$ we have that $\mu(?x) = v$. All these facts imply that mapping $\mu' = \{?x \to v\}$ is in $[\![\langle \text{trans}_N(m_\ell), (\text{GRAPH} ?x\ P_\ell)\rangle]\!]^{\{u\}}_W$ and, thus, μ' is in $[\![Q_n(?x)]\!]^{\{u\}}_W$, which is exactly what we wanted to prove.

If we start by assuming that $\mu = \{?x \to v\}$ is in $[\![Q_n(?x)]\!]^{\{u\}}_W$, then, by following a similar reasoning as above, one concludes that $v \in [\![n]\!]^u_W$.

To complete the proof we have to cover the case in which n, as a regular expression, can produce the empty string. If this is the case, by applying some classical regular languages properties, one can rewrite n as $\varepsilon|n'$ with n' an expression that does not produce the empty string ε. Thus, we can translate n into the LDQL query $Q_n(?x)$ that is given as follows.

$$\langle \varepsilon, (\text{GRAPH } ?x\ \{\ \})\rangle \text{UNION} Q_{n'}(?x).$$

Notice that for every $u \in \text{data}(adoc(v))$ we have that $[\![\langle \varepsilon, (\text{GRAPH } ?x\ \{\ \})\rangle]\!]^{\{u\}}_W$ results in a single mapping $\mu = \{?x \to u\}$, which is enough to conclude that n and $Q_n(?x)$ are equivalent. This completes the proof. □

6. Computability

In this section we consider several computability issues regarding LDQL. We first perform a classical analysis of the complexity of the evaluation problem for the language. In particular, we show that, in a setting in which a complete Web of Linked Data is considered as input, every LDQL query can be evaluated in polynomial time. Although it is not realistic to have complete access to the Web of Linked Data in practice, the theoretical analysis shows that LDQL is comparable in terms of complexity with classical query languages for graph databases.

We then drop the assumption that one has complete access to the Web of Linked Data and show that in this more realistic setting, there exists LDQL queries for which a complete execution is impossible. We formally study this issue proposing the notion of Web-safeness for LDQL queries that ensures that a complete execution of the queries can be performed over the WWW. We finally provide a syntactic sufficient condition that ensures Web-safeness.

6.1. Classical complexity analysis

For our classical analysis we consider the following decision problem that we call *the classical evaluation problem for* LDQL. Given a fixed LDQL query q, the input for the problem is a mapping μ, a finite set S of seed URIs, and a finite Web of Linked Data $W = \langle D, adoc\rangle$. The output is the answer to the following question: is μ in $[\![q]\!]^S_W$? Notice that we consider the *data complexity* [44] of the problem since query q is not considered as part of the input. In this classical scenario, we assume that we have full access to W, in particular, that we have access to the elements in the set $\text{dom}^{\not\perp}(adoc)$. The main result is the following.

Theorem 9. *The classical evaluation problem for LDQL can be solved in polynomial time.*

Proof. The proof is based on two algorithms Eval (Algorithm 1) and Get (Algorithm 2) that we describe next. In both algorithms we make use of a special set $\text{Pos}(q, W, S)$ that contains all the possible solution mappings that are candidates to be in the set $[\![q]\!]^S_W$. Formally, for a given LDQL query q, a Web of Linked Data $W = \langle D, adoc\rangle$, and a set S of URIs, we have:

$$\text{Pos}(q, W, S) = \big\{\mu \mid \text{dom}(\mu) \subseteq \text{var}(q) \text{ and for every } ?X \in \text{dom}(\mu)$$
$$\text{it holds that } \mu(?X) \in \text{dom}^{\not\perp}(adoc) \cup S \cup \text{terms}(W)\big\},$$

where $\text{var}(q)$ is the set of all variables that are mentioned in q, and $\text{terms}(W)$ is the set of all the elements in $\mathcal{U} \cup \mathcal{B} \cup \mathcal{L}$ that are mentioned in some documents in W. It is straightforward to see that to construct all the possible mappings we have to select for every variable in $\text{var}(q)$ either a value in $\text{dom}^{\not\perp}(adoc) \cup S \cup \text{terms}(W)$ or no value at all. Hence, we have:

$$|\text{Pos}(q, W, S)| = (|\text{dom}^{\not\perp}(adoc)| + |S| + |\text{terms}(W)| + 1)^{|\text{var}(q)|}.$$

Notice that the size of $\text{Pos}(q, W, S)$ is polynomial in the size of W and S when q is considered to be fixed, and can be constructed in polynomial time by a simple enumeration. We next prove the correctness of the algorithms, and the polynomial-time complexity.

Correcteness

Procedure Eval (Algorithm 1) is responsible for checking if a mapping μ is in $[\![q]\!]^S_W$, and proceeds by cases according to the form of query q. Procedure Get (Algorithm 2) is responsible for returning all URIs that can be reached from a given URI u following a given LPE *lpe* in the Web of Linked Data W. The formal proof of correctness is by mutual induction between these two procedures and on the construction of query q (in Eval) and on *lpe* (in Get).

Procedure Eval is essentially implementing the semantics of LDQL queries presented in Definition 5. Thus, assuming (as inductive hypothesis) that Eval is correct for subexpressions of query q, a straightforward induction argument shows that Eval is correct for the first five cases (lines 1–19 in Algorithm 1). The only case that deserves a bit more attention is the case in which q is of the form $\langle lpe, P\rangle$ (lines 20–27). For this case we

Algorithm 1 Eval(μ, q, W, S).

1: **if** q is of the form $(q_1 \text{UNION} q_2)$ **then**
2: **if** Eval$(\mu, q_1, W, S) = true$ **or** Eval$(\mu, q_2, W, S) = true$ **then**
3: **return** *true*
4: **else if** q is of the form $(q_1 \text{AND} q_2)$ **then**
5: **for all** pairs of mappings $\mu_1 \in \text{Pos}(q_1, W, S)$ and $\mu_2 \in \text{Pos}(q_2, W, S)$ **do**
6: **if** Eval$(\mu_1, q_1, W, S) = true$ **and** Eval$(\mu_2, q_2, W, S) = true$ **and** $\mu = \mu_1 \cup \mu_2$ **then**
7: **return** *true*
8: **else if** q is of the form $\pi_V q_1$ **then**
9: **for all** mappings $\mu_1 \in \text{Pos}(q_1, W, S)$ **do**
10: **if** Eval$(\mu_1, q_1, W, S) = true$ **and** μ and μ_1 are compatible **and** $\text{dom}(\mu) = \text{dom}(\mu_1) \cap V$ **then**
11: **return** *true*
12: **else if** q is of the form (SEED U q_1) **then**
13: **if** Eval$(\mu, q_1, W, U) = true$ **then**
14: **return** *true*
15: **else if** q is of the form (SEED $?v$ q_1) **then**
16: **for all** URIs $u \in \text{dom}^{\not\perp}(adoc)$ **do**
17: **for all** mappings $\mu_1 \in \text{Pos}(q_1, W, \{u\})$ **do**
18: **if** Eval$(\mu_1, q_1, W, S) = true$ **and** $\mu = \mu_1 \cup \{?v \to u\}$ **then**
19: **return** *true*
20: **else if** q is of the form $\langle lpe, P \rangle$ **then**
21: rewrite *lpe* into *lpe*′ using only ε, $(\ )^*$ and the construction $\langle ?v, p \rangle$
22: $U := \emptyset$
23: **for all** $u \in S$ **do**
24: $U := U \cup \text{Get}(u, lpe', W)$
25: $D := \text{dataset}(U)$
26: **if** $\text{Eval}_{\text{SPARQL}}(\mu, P, D) = true$ **then**
27: **return** *true*
28: **return** *false*

Algorithm 2 Get(u, lpe, W).

1: **if** $u \notin \text{dom}^{\not\perp}(adoc)$ **then return** $\emptyset$
2: **if** *lpe* is ε **then**
3: **return** $\{u\}$
4: **else if** *lpe* is of the form $\langle ?v, q \rangle$ **then**
5: $X := \emptyset$
6: **for all** solution mapping $\mu \in \text{Pos}(q, W, \{u\})$ **do**
7: **if** Eval$(\mu, q, W, \{u\}) = true$ **and** $\mu(?v) \in \mathcal{U}$ **then** $X := X \cup \{\mu(?v)\}$
8: **return** X
9: **else if** *lpe* is of the form r^* **then**
10: $X := \emptyset$
11: let Q be an empty queue
12: Q.enqueue(u)
13: **while** Q is not empty **do**
14: $v := Q$.dequeue()
15: $X := X \cup \{v\}$
16: **for all** w in Get(v, r, W) **do**
17: **if** w is not in X **then** Q.enqueue(w)
18: **return** X

begin by constructing a new expression *lpe*′ that is equivalent to *lpe* but that only uses operators ε, $(\ )^*$ and the construction $\langle ?v, p \rangle$. By Proposition 1 we know that this is always possible. The algorithm proceeds by accumulating in the set U all the URIs that are reachable from S via *lpe*′ by using procedure Get(v, r, W). Then, from U it constructs the corresponding dataset D and finally uses $\text{Eval}_{\text{SPARQL}}(\mu, P, D)$ to perform a standard SPARQL evaluation to check if μ is in the evaluation of pattern P over dataset D. Thus, assuming (as inductive hypothesis) that procedure Get is correct for subexpressions of q of the form $\langle lpe, P \rangle$, we obtain that Eval is also correct in this case as it is essentially implementing the semantics for expressions $\langle lpe, P \rangle$ presented in Definition 5.

To argue about the correctness of Get, first notice that we only have a few cases. The cases in which *lpe* is ε or in which $u \notin \text{dom}^{\not\perp}(adoc)$ are trivially correct (these are the base cases of the mutual induction). For the case in which *lpe* is of the form $\langle ?v, q \rangle$, we use the (inductive hypothesis) assumption that Eval is correct for subexpressions of q. In this case Get is essentially implementing the definition of $[\![\langle ?v, q \rangle]\!]^u_W$ (Definition 5). For the last case, in which $lpe = r^*$, first notice that by, Definition 5, we have that $w \in [\![r^*]\!]^u_W$ if and only if there exists an integer k such that $w \in [\![r^k]\!]^u_W$, where r^k is the expression formed by concatenating k copies of expression r. Let X be the output of Get(u, r^*, W). We prove next that $w \in X$ if and only if there exists an integer k such that $w \in [\![r^k]\!]^u_W$, which is enough to prove the correctness of Get.

Let $G_r = (V, E_r)$ be the graph such that $V = \text{dom}^{\not\perp}(adoc)$ and $(x, y) \in E_r$ if and only if $y \in \text{Get}(x, r, W)$. Notice that algorithm Get is performing a standard Breadth-First-Search (BFS) procedure over graph G_r starting from node u (lines 9–18 in Algorithm 2). Thus, when the procedure terminates, we know that set X stores all elements in V that are reachable from u following a path in G_r. Then, we have that $w \in X$ if and only if there exists a sequence $v_0, v_1, \ldots, v_k$ such that $v_0 = u$, $v_k = w$ and $v_i \in \text{Get}(v_{i-1}, r, W)$. Given that we can assume (by inductive hypothesis) that Get is correct for subexpressions of $lpe = r^*$, we know that $v_i \in \text{Get}(v_{i-1}, r, W)$ if and only if $v_i \in [\![r]\!]^{v_{i-1}}_W$. Summing up, we have that $w \in X$ if and only if there exists a sequence $v_0, v_1, \ldots, v_k$ such that $v_0 = u$, $v_k = w$ and $v_i \in [\![r]\!]^{v_{i-1}}_W$, which is equivalent to say that $w \in [\![r^k]\!]^u_W$. We finally conclude that $w \in X$ if and only if there exists a k such that $w \in [\![r^k]\!]^u_W$. This completes the proof of correctness of Get.

Complexity

We use an inductive argument to show that the complexity of Eval is polynomial. The induction is on the number of recursive calls during the execution of the complete procedure. Thus, assume that for less than N recursive calls, Eval performs in time proportional to a polynomial w.r.t. the size of W and S (recall that the query is not considered as part of the input to measure the complexity). Now, assume that the complete execution of Eval(μ, q, W, S) performs a total of N recursive calls. We proceed by cases.

For the case in which q is of the form $(q_1 \text{UNION} q_2)$, the total time is proportional to the sum of the time for Eval(μ_1, q_1, W, S) and Eval(μ_2, q_2, W, S), and since both perform less that N recursive calls, by the inductive hypothesis, both are of time polynomial (w.r.t. W and S). Thus, the complete execution is polynomial. A similar argument applies when q is of the form (SEED U q_1).

Now consider the case in which q is of the form $(q_1 \text{AND} q_2)$. In this case we have that the number of calls to Eval (line 6 in Algorithm 1) is $2 \times |\text{Pos}(q_1, W, S)| \times |\text{Pos}(q_2, W, S)|$. Notice that both, $|\text{Pos}(q_1, W, S)|$ and $|\text{Pos}(q_2, W, S)|$, are of polynomial size w.r.t. W and S, and thus, given that every call to Eval takes polynomial time, the total time is also polynomial in this case. A similar argument applies if q is of the form $\pi_V q_1$.

Consider now the case in which q is of the form (SEED $?v$ q_1). In this case the number of calls to Eval is $|\text{dom}^{\not\perp}(adoc)| \times |\text{Pos}(q_1, W, \{u\})|$, which is polynomial w.r.t. W and S, and thus the time is also polynomial in this case.

The only case that is left to be analyzed is the case in which q is of the form $\langle lpe, P \rangle$ (lines 20–27). By Lemma 3 we know that line 21 can be completed in polynomial time. Now notice that in line 25

the procedure constructs an RDF dataset D from the set U. Given that U is at most of size $|\text{dom}^{\curlyvee}(adoc)|$, then U is of polynomial size w.r.t. W, and one can construct D in polynomial time (provided that we have complete access to Web W). Moreover, notice that $\text{Eval}_{\text{SPARQL}}(\mu, P, D)$ is performing a standard SPARQL graph pattern evaluation, which we know can be done in polynomial time w.r.t. the size of D if the pattern P is considered fixed [40]. Hence, we only need to analyze the time complexity of the loop in line 23. The loop is executed $|S|$ times and the only important operation is the call to Get. Thus, to show the polynomial time complexity we only need to show that $\text{Get}(u, lpe, W)$ takes polynomial time.

To this end, we use a similar inductive argument for procedure Get as used for procedure Eval. Suppose that for less than K recursive calls, Get performs in polynomial time, and assume that K recursive calls are performed when executing $\text{Get}(u, lpe, W)$. We proceed by cases. The case in which lpe is ε is trivially polynomial. For the case in which lpe is of the form $\langle ?v, q\rangle$, the algorithm performs $|\text{Pos}(q_1, W, \{u\})|$ calls to Eval. We know that $|\text{Pos}(q_1, W, \{u\})|$ is polynomial and that the time to execute Eval is also polynomial, which gives us a polynomial total time. The last case is if lpe is of the form r^*. In this case, as we argued when proving the correctness of the procedure, Get is essentially performing a BFS over the graph $G_r = (V, E_r)$ with $V = \text{dom}^{\curlyvee}(adoc)$ and $E_r = \{(x, y) \mid y \in \text{Get}(x, r, W)\}$. Thus, the total time needed to complete this case is proportional to $|V| + |E|$ multiplied by the time needed to access the neighbor of every node, which in this case is the time needed to execute Get. Finally, since (i) $|V| = |\text{dom}^{\curlyvee}(adoc)|$, (ii) $|E| \leq |\text{dom}^{\curlyvee}(adoc)|^2$, and, (iii) by our induction hypothesis, the time needed to execute $\text{Get}(v, r, W)$ is polynomial (it performs less than K recursive calls), we have that the total time is also polynomial. This completes the proof of the complexity of Eval. □

6.2. Web-safeness

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 a Web of Linked Data such as 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 and study its expressiveness and classical computational complexity), 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 [2], we cannot assume complete information about which URIs are in the set $\text{dom}^{\curlyvee}(adoc)$ (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. 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 [43].

Definition 6. An LDQL query q is **Web-safe** if there exists an algorithm that, for any finite Web of Linked Data $W = \langle D, adoc\rangle$ and any finite set S of URIs, has the following three properties:

1. The algorithm computes $[\![q]\!]_W^S$.
2. During its execution, the algorithm looks up only a finite number of URIs (that is, conceptually, the algorithm invokes function *adoc* only a finite number of times).
3. Neither the set D nor the set $\text{dom}^{\curlyvee}(adoc)$ is required as input for the algorithm (hence, the algorithm does not require any a priori information about W).

Example 6. Recall the following three LDQL queries as introduced in Examples 4 and 5:

$$q_{\text{ex}} = \left(\text{SEED}?x \left\langle \varepsilon, \langle ?x, u_{\text{sequelOf}}, ?z\rangle\right\rangle\right), \qquad q'_{\text{ex}} = \langle lpe_{\text{ex}}, B_{\text{ex}}\rangle,$$
$$q''_{\text{ex}} = (q_{\text{ex}} \text{AND} q'_{\text{ex}}),$$

where $lpe_{\text{ex}} = \langle \_, u_{\text{sequelOf}}, \_\rangle^*/[\langle \_, u_{\text{influencedBy}}, \_\rangle]$ and $B_{\text{ex}} = \{\langle ?x, u_{\text{sequelOf}}, ?y\rangle, \langle ?x, u_{\text{influencedBy}}, ?z\rangle\}$.

For query q_{ex}, 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 $\langle u, u_{\text{sequelOf}}, ?z\rangle$. Therefore, without access to D or $\text{dom}^{\curlyvee}(adoc)$ 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_{ex} is *not* Web-safe. In contrast, although it contains q_{ex} as a subquery, query q''_{ex} is Web-safe, and so is q'_{ex}. Given $u_{\text{Revolutions}}$ as seed URI, a possible execution algorithm for q'_{ex} may first compute $[\![lpe_{\text{ex}}]\!]_W^{u_{\text{Revolutions}}}$ by traversing the queried Web W based on lpe_{ex}. Thereafter, the algorithm retrieves documents by looking up all URIs $u \in [\![lpe_{\text{ex}}]\!]_W^{u_{\text{Revolutions}}}$ (or simply keeps these documents after the traversal); and, finally, the algorithm evaluates pattern B_{ex} over the union of the RDF data 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''_{ex} is also Web-safe we note that after executing subquery q'_{ex} (e.g., by using the algorithm as outlined before), the execution of the other (non-Web-safe) subquery q_{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'_{ex}. Although any other URI may also be used to obtain solution mappings for q_{ex}, such solution mappings cannot be joined with any of the solution mappings for q'_{ex} and, thus, are irrelevant for the result of q''_{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_{ex} as made in the example can be applied to any LDQL query of the form $(\text{SEED}?x\ q)$ where subquery q is a (satisfiable) basic LDQL query; 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 show properties to identify LDQL queries that are Web-safe even if some of their subqueries are not. We begin with queries of the forms $\langle lpe, P\rangle$, $\pi_V q$, $(\text{SEED}\ U\ q)$, and $(q_1 \text{UNION} \ldots \text{UNION} q_n)$.

Proposition 2. *A basic LDQL query $\langle lpe, P\rangle$ is Web-safe if lpe is Web-safe, where we call an LPE Web-safe if either* (i) *it is of the form $\langle ?v, q'\rangle$ and the contained LDQL query q' is Web-safe, or* (ii) *it is of any form other than $\langle ?v, q'\rangle$ and all its subexpressions (if any) are Web-safe LPEs.*

Proposition 3. *An LDQL query of the form $\pi_V q'$ or the form $(\text{SEED}\ U\ q')$ is Web-safe if subquery q' is Web-safe.*

Proposition 4. *An LDQL query of the form $(q_1 \text{UNION} \ldots \text{UNION} q_n)$ is Web-safe if every subquery q_i $(1 \leq i \leq n)$ is Web-safe.*

Proof of Proposition 2. Let q be an arbitrary basic LDQL query $\langle lpe, P\rangle$ such that lpe is Web-safe. To show that q is Web-safe we provide Algorithm 3. In line 3 the algorithm calls a subroutine, ExecLPE (cf. Algorithm 4), that evaluates a given LPE in the context of a given URI. The correctness of the algorithm and its subroutine is easily checked. Moreover, a trivial proof by induction on the possible structure of LPEs can show that for any Web-safe LPE, the given subroutine looks up a finite number of URIs only. The crux of

Algorithm 3 Execution of a basic LDQL query $\langle lpe, P\rangle$ using a set S of URIs as seed.

1: Φ := a new empty set of URIs
2: **for all** $u \in S$ **do**
3: $\quad \Phi := \Phi \cup \text{ExecLPE}(lpe, u)$
4: G := a new empty set of RDF triples (i.e., an empty RDF graph)
5: $\mathcal{N}$:= a new empty set of pairs consisting of a URI and an RDF graph
6: **for all** $u \in \Phi$ **do**
7: $\quad$ **if** looking up URI u results in retrieving a document, say d **then**
8: $\quad\quad G := G \cup \text{data}(d)$
9: $\quad\quad \mathcal{N} := \mathcal{N} \cup \{\langle u, \text{data}(d)\rangle\}$
10: **return** $[\![P]\!]_G^{\langle G,\mathcal{N}\rangle}$ // $[\![P]\!]_G^{\langle G,\mathcal{N}\rangle}$ can be computed by using any algorithm that implements // the standard (set-based) SPARQL evaluation function[41]

such a proof is twofold: First, the evaluation of LPEs of the form lpe^* (lines 20–28 in Algorithm 4) is guaranteed to reach a fixed point for any *finite* Web of Linked Data. Second, the evaluation of LPEs of the form $\langle ?v, q\rangle$ (lines 32–36) uses an algorithm for subquery q that has the properties as required in Definition 6. Due to the Web-safeness of the given LPE and, thus, of q, such an algorithm exists. □

Proof of Proposition 3. First, let q be an LDQL query of the form $\pi_V q'$ such that subquery q' is Web-safe. Due to the Web-safeness of q', there exists an algorithm for q' that has the properties as required in Definition 6. We may use this algorithm to construct an algorithm for q; that is, our algorithm for q calls the algorithm for q', applies the projection operator to the result, and returns the set of solution mappings resulting from this projection. Since the application of the projection operator does not involve URI lookups, the constructed algorithm for q has the properties as required in Definition 6. Second, let q be an LDQL query of the form (SEED U q') such that q' is Web-safe. Hence, there exists an algorithm for q' that has the properties as required in Definition 6. Then, showing the Web-safeness of q is trivial because the algorithm for q' can also be used for q. □

Proof of Proposition 4. Let LDQL query q be of the form (q_1UNION ... UNIONq_n) such that every q_i ($1 \leq i \leq n$) is Web-safe. Hence, for every subquery q_i, there exists an algorithm that has the properties as required in Definition 6. Then, the Web-safeness of query q is easily shown by specifying another algorithm that calls the algorithms of the subqueries sequentially and unions their results. □

It remains to discuss LDQL queries of the form (q_1AND ... ANDq_m). Our discussion of query q''_{ex} in Example 6 suggests that such queries can be shown to be Web-safe if all non-Web-safe subqueries are of the form (SEED$?v$ q) and it is possible to execute these subqueries by using variable bindings obtained from other subqueries. A necessary condition for this execution strategy is that the variable in question (i.e., $?v$) 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 [45].[3]

[3] 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 [45] would also carry over to LDQL queries. Therefore, we omit discussing boundedness and use directly the decidable alternative (i.e., strong boundedness).

To this end, for any SPARQL graph pattern P, let sbvars(P) denote the set of strongly bound variables in P as defined by Buil-Aranda et al. [45]. For the sake of space, we do not repeat the definition here. However, we emphasize that sbvars(P) can be constructed recursively, and each variable in sbvars(P) is guaranteed to be bound in every possible solution for P [45, 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 4, we show the desired boundedness guarantee.

Definition 7. The set of **strongly bound variables** in an LDQL query q, denoted by sbvars(q), is defined recursively as follows:

Algorithm 4 ExecLPE(lpe, u_{ctx})

1: **if** looking up URI u_{ctx} results in retrieving a document, say d_{ctx} **then**
2: $\quad$ **if** lpe is ε **then**
3: $\quad\quad$ **return** a new singleton set $\{u_{\text{ctx}}\}$
4: $\quad$ **else if** lpe is a link pattern $lp = \langle y_1, y_2, y_3\rangle$ **then**
5: $\quad\quad lp'$:= $\langle y'_1, y'_2, y'_3\rangle$, where $\langle y'_1, y'_2, y'_3\rangle$ is a link pattern generated from lp such that any occurrence of symbol $+$ in lp is replaced by URI u_{ctx}
6: $\quad\quad \Phi$:= a new empty set of URIs
7: $\quad\quad$ **for all** $\langle x_1, x_2, x_3\rangle \in \text{data}(d_{\text{ctx}})$ **do**
8: $\quad\quad\quad$ **if** $(y'_1 = x_1$ **or** $y'_1 = \_)$ **and** $(y'_2 = x_2$ **or** $y'_2 = \_)$ **and** $(y'_3 = x_3$ **or** $y'_3 = \_)$ **then**
9: $\quad\quad\quad\quad$ **for all** $i \in \{1, 2, 3\}$ **do**
10: $\quad\quad\quad\quad\quad$ **if** $y'_i = \_$ **and** x_i is a URI whose lookup retrieves a document **then** $\Phi := \Phi \cup \{x_i\}$
11: $\quad\quad$ **return** Φ
12: $\quad$ **else if** lpe is of the form lpe_1/lpe_2 **then**
13: $\quad\quad \Phi'$:= ExecLPE(lpe_1, u_{ctx})
14: $\quad\quad \Phi$:= a new empty set of URIs
15: $\quad\quad$ **for all** $u' \in \Phi'$ **do** $\Phi := \Phi \cup \text{ExecLPE}(lpe_2, u')$ **end for**
16: $\quad\quad$ **return** Φ
17: $\quad$ **else if** lpe is of the form $lpe_1|lpe_2$ **then**
18: $\quad\quad \Phi_1$:= ExecLPE(lpe_1, u_{ctx}); Φ_2 := ExecLPE(lpe_2, u_{ctx})
19: $\quad\quad$ **return** $\Phi_1 \cup \Phi_2$
20: $\quad$ **else if** lpe is of the form l^* **then**
21: $\quad\quad \Phi_{\text{cur}}$:= ExecLPE(ε, u_{ctx})
22: $\quad\quad lpe'$:= l
23: $\quad\quad$ **repeat**
24: $\quad\quad\quad \Phi_{\text{prev}} := \Phi_{\text{cur}}$
25: $\quad\quad\quad \Phi_{\text{cur}} := \Phi_{\text{cur}} \cup \text{ExecLPE}(lpe', u_{\text{ctx}})$
26: $\quad\quad\quad lpe'$:= an LPE of the form lpe'/l
27: $\quad\quad$ **until** $\Phi_{\text{cur}} = \Phi_{\text{prev}}$
28: $\quad\quad$ **return** Φ_{cur}
29: $\quad$ **else if** lpe is of the form $[lpe']$ **then**
30: $\quad\quad \Phi$:= ExecLPE(lpe', u_{ctx})
31: $\quad\quad$ **if** $\Phi \neq \emptyset$ **then return** a new singleton set $\{u_{\text{ctx}}\}$ **else return** a new empty set **end if**
32: $\quad$ **else if** lpe is of the form $\langle ?v, q\rangle$ **then**
33: $\quad\quad \Omega$:= Exec(q, $\{u_{\text{ctx}}\}$) // where Exec is an arbitrary algorithm that can be used to compute the // $\{u_{\text{ctx}}\}$-based evaluation of q over the queried Web of Linked Data
34: $\quad\quad \Phi$:= a new empty set of URIs
35: $\quad\quad$ **for all** $\mu \in \Omega$ for which $?v \in \text{dom}(\mu)$ and $\mu(?v) \in \mathcal{U}$ **do** $\Phi := \Phi \cup \{\mu(?v)\}$ **end for**
36: $\quad\quad$ **return** Φ
37: **else**
38: $\quad$ **return** a new empty set

1. If q is of the form $\langle lpe, P\rangle$, then $\text{sbvars}(q) = \text{sbvars}(P)$.
2. If q is of the form $(q_1 \text{AND} q_2)$, then $\text{sbvars}(q) = \text{sbvars}(q_1) \cup \text{sbvars}(q_2)$.
3. If q is of the form $(q_1 \text{UNION} q_2)$, then $\text{sbvars}(q) = \text{sbvars}(q_1) \cap \text{sbvars}(q_2)$.
4. If q is of the form $\pi_V q'$, then $\text{sbvars}(q) = \text{sbvars}(q') \cap V$.
5. If q is of the form $(\text{SEED}\ U\ q')$, then $\text{sbvars}(q) = \text{sbvars}(q')$.
6. If q is of the form $(\text{SEED}?v\ q')$, then $\text{sbvars}(q) = \text{sbvars}(q') \cup \{?v\}$.

Lemma 4. *Let q be an LDQL query. For every finite set S of URIs, every Web of Linked Data W, and every $\mu \in [\![q]\!]^S_W$, it holds that* $\text{sbvars}(q) \subseteq \text{dom}(\mu)$.

Proof. Lemma 4 follows trivially from Definition 7 and [45, Proposition 1]. □

Given the notion of strongly bound variables, we are now ready to show the following main result related to the Web-safeness of LDQL.

Theorem 10. *An LDQL query of the form $(q_1 \text{AND} q_2 \text{AND} \ldots \text{AND}\ q_m)$ 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 Web-safe or* (ii) *q_i is of the form $(\text{SEED}?v\ q)$ where q is Web-safe and $?v \in \bigcup_{q_j \prec q_i} \text{sbvars}(q_j)$.*

Proof. We prove Theorem 10 based on Algorith 5, which is an iterative algorithm that generalizes the execution strategy outlined for query q''_{ex} in Example 6. That is, the algorithm executes the subqueries $q_1, q_2, \ldots, q_m$ sequentially in the order $\prec$ such that each iteration step (lines 2–17) executes one of the subqueries by using the solution mappings computed during the previous step (which are passed on via the sets $\Omega_0, \Omega_1, \ldots, \Omega_m$).

To prove that Algorith 5 has the properties as required in Definition 6 we have to show that the algorithm is sound and complete (i.e., for any finite set S of URIs and any Web of Linked Data W, the algorithm returns $[\![q]\!]^S_W$) and that it is guaranteed to look up a finite number of URIs only. We show these properties by induction on the m iteration steps performed by the algorithm. To this end, we assume that the indices as used for the subqueries $q_1, q_2, \ldots, q_m$ reflect the order $\prec$, that is, subquery q_1 is the first according to $\prec$, subquery q_2 is the second, and so on.

Base Case ($m = 1$): By the conditions in Theorem 10, the first subquery (according to $\prec$) must be Web-safe and, thus, cannot be of the form $(\text{SEED}\ ?v\ q')$. Hence, the algorithm enters the corresponding else-branch (line 12). Due to the Web-safeness of q_1, there exists an algorithm for subquery q_1, say A_1, that has the properties as required in Definition 6. Algorith 5 uses algorithm A_1 to obtain $\Omega_{\text{tmp}} = [\![q_1]\!]^S_W$ (where W is the queried Web of Linked Data), which requires only a finite number of URI lookups. Thereafter, Algorith 5 computes $\Omega_1 = \Omega_0 \bowtie \Omega_{\text{tmp}}$ (lines 13–17) and returns Ω_1 (line 18), which does not require any more URI lookups. Hence, for $m = 1$, the algorithm looks up a finite number of URIs (if the queried Web of Linked Data is finite). Since Ω_0 contains only the empty solution mapping $\mu_\emptyset$ (line 1), which is compatible with any other solution mapping, we have $\Omega_1 = \Omega_{\text{tmp}}$ and, thus, $\Omega_1 = [\![q_1]\!]^S_W$.

Induction Step ($m > 1$): By induction we assume that after completing the $(m-1)$th iteration, the algorithm has looked up a finite number of URIs only and the current intermediate result Ω_{m-1} covers the conjunction of subqueries $q_1, q_2, \ldots, q_{m-1}$; that is, $\Omega_{m-1} = [\![(q_1 \text{AND} q_2 \text{AND} \ldots \text{AND} q_{m-1})]\!]^S_W$. We show that the mth iteration also looks up a finite number of URIs only and that $\Omega_m = [\![(q_1 \text{AND} q_2 \text{AND} \ldots \text{AND} q_m)]\!]^S_W$.

If subquery q_m is Web-safe, it is not difficult to see these properties: Since q_m is Web-safe, there exists an algorithm for q_m, say A_m, that has the properties as required in Definition 6. Hence, by calling algorithm A_m in line 12, Algorith 5 looks up a finite number of URIs only, and the subsequent join computation in lines 13–17 does not require any more lookups. Moreover, the result of calling algorithm A_m in line 12 is $\Omega_{\text{tmp}} = [\![q_m]\!]^S_W$ and, since the subsequent join computation returns $\Omega_m = \Omega_{m-1} \bowtie \Omega_{\text{tmp}}$, we have $\Omega_m = [\![(q_1 \text{AND} q_2 \text{AND} \ldots \text{AND} q_m)]\!]^S_W$.

It remains to discuss the case of subquery q_m being of the form $(\text{SEED}\ ?v\ q')$, where, by the conditions in Theorem 10, subquery q' is Web-safe. Hence, there exists an algorithm for q', say A', that has the properties as required in Definition 6. In this case, Algorith 5 first iterates over all solution mappings in Ω_{m-1} to populate a set U_{tmp} with all URIs that any of these mappings binds to variable $?v$ (lines 6–8). Due to the finiteness assumed for all queried Webs of Linked Data (cf. Definition 6), Ω_{m-1} is finite. Hence, the resulting set U_{tmp} contains a finite number of URIs. Therefore, the subsequent loop in lines 9–10 calls algorithm A' a finite number of times and, thus, the mth iteration looks up a finite number of URIs only.

For the remaining claim, $\Omega_m = [\![(q_1 \text{AND} q_2 \text{AND} \ldots \text{AND} q_m)]\!]^S_W$, we first show $\Omega_m \subseteq [\![(q_1 \text{AND} q_2 \text{AND} \ldots \text{AND} q_m)]\!]^S_W$. Let $\mu_{\text{join}} \in \Omega_m$ be an arbitrary solution mapping in Ω_m. By lines 14–17, there exist two solution mappings, μ and μ', such that (i) $\mu \in \Omega_{m-1}$ with $\Omega_{m-1} = [\![(q_1 \text{AND} q_2 \text{AND} \ldots \text{AND} q_{m-1})]\!]^S_W$, (ii) $\mu' \in \Omega_{\text{tmp}}$ with $\Omega_{\text{tmp}} = \bigcup_{u \in U_{\text{tmp}}} [\![q']\!]^{\{u\}}_W$, (iii) μ and μ' are compatible, and (iv) $\mu_{\text{join}} = \mu \cup \mu'$. Then, we have $\mu_{\text{join}} \in [\![(q_1 \text{AND} q_2 \text{AND} \ldots \text{AND} q_m)]\!]^S_W$ and, thus, $\Omega_m \subseteq [\![(q_1 \text{AND} q_2 \text{AND} \ldots \text{AND} q_m)]\!]^S_W$.

Finally, we show $\Omega_m \supseteq [\![(q_1 \text{AND} q_2 \text{AND} \ldots \text{AND} q_m)]\!]^S_W$. Assume a mapping $\mu^* \in [\![(q_1 \text{AND} q_2 \text{AND} \ldots \text{AND} q_m)]\!]^S_W$. By Definition 5, there exist two solution mappings μ^*_1 and μ^*_2 such that (i) $\mu^*_1 \in [\![(q_1 \text{AND} q_2 \text{AND} \ldots \text{AND} q_{m-1})]\!]^S_W$, (ii) $\mu^*_2 \in [\![q_m]\!]^S_W$,

Algorithm 5 Execution of an LDQL query q of the form $(q_1 \text{AND} q_2 \text{AND} \ldots \text{AND} q_m)$ using a finite set S of seed URIs.

Require: $m \geq 1$
Require: LDQL query q is given as an array Q consisting of all subqueries of q such that the order of the subqueries in this array satisfies the conditions as given in Theorem 10.

1: Ω_0 := $\{\mu_\emptyset\}$, where $\mu_\emptyset$ is the empty solution mapping; i.e., $\text{dom}(\mu_\emptyset) = \emptyset$
2: **for** j := 1, ..., m **do**
3: $\quad\Omega_{\text{tmp}}$:= a new empty set of solution mappings
4: $\quad q_j$:= the j-th subquery in array Q
5: $\quad$**if** q_j is of the form $(\text{SEED}\ ?v\ q')$ **then**
6: $\quad\quad U_{\text{tmp}}$:= a new empty set of URIs
7: $\quad\quad$**for all** $\mu \in \Omega_{j-1}$ **do**
8: $\quad\quad\quad$**if** $\mu(?v)$ is a URI **then** $U_{\text{tmp}} := U_{\text{tmp}} \cup \{\mu(?v)\}$ **end if**
9: $\quad\quad$**for all** $u \in U_{\text{tmp}}$ **do**
10: $\quad\quad\quad\Omega_{\text{tmp}} := \Omega_{\text{tmp}} \cup \text{Exec}(q', \{u\})$ // where Exec denotes an arbitrary algorithm that can be used to compute // the $\{u\}$-based evaluation of q' over the queried Web of Linked Data
11: $\quad$**else**
12: $\quad\quad\Omega_{\text{tmp}}$:= $\text{Exec}(q_j, S)$ // where Exec is an arbitrary algorithm that can be used to compute // the S-based evaluation of q_j over the queried Web of Linked Data
13: $\quad\Omega_j$:= a new empty set of solution mappings
14: $\quad$**for all** $\mu \in \Omega_{j-1}$ **do**
15: $\quad\quad$**for all** $\mu' \in \Omega_{\text{tmp}}$ **do**
16: $\quad\quad\quad$**if** μ and μ' are compatible **then**
17: $\quad\quad\quad\quad\Omega_j := \Omega_j \cup \{\mu_{\text{join}}\}$, where $\mu_{\text{join}} = \mu \cup \mu'$
18: **return** Ω_m

(iii) μ_1^* and μ_2^* are compatible, and (iv) $\mu^* = \mu_1^* \cup \mu_2^*$. By our induction hypothesis, we have $\mu_1^* \in \Omega_{m-1}$. Then, given lines 14–17, we have to show that $\mu_2^* \in \Omega_{\text{tmp}}$ where Ω_{tmp} is the set of solution mappings computed during the mth iteration. Since q_m is of the form (SEED $?v$ q'), it holds that $\Omega_{\text{tmp}} = \bigcup_{u \in U_{\text{tmp}}} [\![q']\!]_W^{\{u\}}$ where

$$U_{\text{tmp}} = \{u \in \mathcal{U} \mid \mu(?v) = u \text{ for some } \mu \in [\![(q_1 \text{AND} q_2 \text{AND} \ldots \text{AND} q_{m-1})]\!]_W^S\}.$$

Hence, to show that $\mu_2^* \in \Omega_{\text{tmp}}$ we show that there exists a URI $u \in U_{\text{tmp}}$ such that μ_2^* is in $[\![q']\!]_W^{\{u\}}$. Since $\mu_2^* \in [\![q_m]\!]_W^S$, by Definition 5, solution mapping μ_2^* binds variable $?v$ to a URI, say u^*; i.e., $?v \in \text{dom}(\mu_2^*)$ and $\mu_2^*(?v) = u^*$ with $u^* \in \mathcal{U}$. Furthermore, by Lemma 4 and the condition in Theorem 10 (i.e., $?v \in \bigcup_{q_k \prec q_m} \text{sbvars}(q_k)$), solution mapping μ_1^* also has a binding for variable $?v$, and, since μ_1^* and μ_2^* are compatible, these bindings are the same, that is, $\mu_1^*(?v) = \mu_2^*(?v)$. Hence, for URI $u^* = \mu_2^*(?v)$ it holds that $u^* \in U_{\text{tmp}}$. Then, by Definition 5, we obtain that $\mu_2^* \in [\![q']\!]_W^{\{u\}}$, which shows that $\mu_2^* \in \Omega_{\text{tmp}}$ and, thus, we can conclude that $\Omega_m \supseteq [\![(q_1 \text{AND} q_2 \text{AND} \ldots \text{AND} q_m)]\!]_W^S$. □

With the results in this section we have all ingredients to devise a procedure to show Web-safeness for a large number of queries (including queries that are arbitrarily nested). However, as a potential limitation of such a procedure we note that Theorem 10 can be applied only in cases in which all non-Web-safe subqueries are of the form (SEED$?v$ q). For instance, the theorem cannot be applied to show that an LDQL query of the form $\big(q_1 \text{AND}(q_2 \text{UNION}(\text{SEED}?x\, q_3))\big)$ is Web-safe if $?x \in \text{sbvars}(q_1)$ and q_1, q_2 and q_3 are Web-safe. On the other hand, for the semantically equivalent query $\big((q_1 \text{AND} q_2) \text{UNION}(q_1 \text{AND}(\text{SEED}?x\, q_3))\big)$ we can show Web-safeness based on Theorem 10 (and Propositions 2–4). Fortunately, we may leverage the following fact to improve the effectiveness of applying Theorem 10 in the procedure that we aim to devise.

Fact 1. *An LDQL query q is Web-safe if there exists another LDQL query q' such that q' is Web-safe and q and q' are semantically equivalent (i.e., $q \equiv q'$).*

As a consequence of Fact 1, we may use the equivalences in Lemma 2 to rewrite a given query into an equivalent query that is more suitable for testing Web-safeness based on our results. To this end, we introduce specific normal forms for LDQL queries:

Definition 8. An LDQL query is in **UNION-free normal form** if it is of the form $(q_1 \text{AND} q_2 \text{AND} \ldots \text{AND} q_m)$ with $m \geq 1$ and each subquery q_i $(1 \leq i \leq m)$ is either (i) a basic LDQL query or (ii) of the form $\pi_V q$, (SEED U q), or (SEED $?v$ q) such that subquery q is in UNION-free normal form. An LDQL query is in **UNION normal form** if it is of the form $(q_1 \text{UNION} q_2 \text{UNION} \ldots \text{UNION} q_n)$ with $n \geq 1$ and each subquery q_i $(1 \leq i \leq n)$ is in UNION-free normal form.

The following result is an immediate consequence of Lemma 2.

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

In conjunction with Fact 1, Corollary 1 allows us to focus on LDQL queries in UNIONnormal form without losing generality. We are now ready to specify our procedure that applies the results in this paper to test a given LDQL query q for Web-safeness: First, by using the equivalences in Lemma 2, the query has to be rewritten into a semantically equivalent LDQL query q_{nf} that is in UNIONnormal form ; i.e., $q_{\text{nf}} = (q_1 \text{UNION} q_2 \text{UNION} \ldots \text{UNION} q_n)$. Next, the following test has to be repeated for every subquery q_i $(1 \leq i \leq n)$; recall that each of these subqueries is in UNION-free normal form; i.e., $q_i = (q_1^i \text{AND} q_2^i \text{AND} \ldots \text{AND} q_{m_i}^i)$. The test is to find an order for their subqueries $q_1^i, \ldots, q_{m_i}^i$ that satisfies the conditions in Theorem 10. Every top-level subquery q_i $(1 \leq i \leq n)$ for which such an order exists, is Web-safe (cf. Theorem 10). If all top-level subqueries are identified to be Web-safe by this test, then q_{nf} is Web-safe (cf. Proposition 4), 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 both sufficient and necessary.

7. 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. We prove several good properties of LDQL. Regarding expressiveness, we compare LDQL with previous formalisms and show that LDQL is strictly more expressive. Regarding complexity, we show that when the input is assumed to be a full accessible graph, the data complexity of the language is polynomial, which makes LDQL comparable with previously proposed graph query languages. We also study the notion of the Web-safeness property that ensures that a complete execution of a query is possible even if we consider the limited data access capabilities of Web clients.

Several topics remain open for future work. One of them is a theoretical complexity study of query evaluation that takes into account the limited data access capabilities of Web clients. Such study should consider a model that captures the inherent way of accessing the Web of Linked Data via HTTP requests, the overhead of data communication and transfer, the distribution of data and documents, etc.

A more practical direction for future research on LDQL is an investigation of approaches to actually execute LDQL queries. When implementing a system to this end, a number of data access specific issues must be taken into account: Some URI lookups may result in the retrieval of an unforeseeable large set of RDF triples; response times may vary significantly between different Web servers; sometimes a URI lookup may take unexpectedly long. In general, Web servers may exhibit an unexpected behavior. Furthermore, some servers enforce restrictions on clients such as serving only a limited number of requests per second. Regarding the latter it is important to emphasize that any LDQL query execution system should implement a politeness policy to avoid overloading servers. In particular, such a system should abide by the Robots Exclusion Protocol[4] and by related extensions of this protocol that allow Web sites to demand delays between subsequent requests from the same client. Even for Web sites that do not provide a robots.txt file, a minimum delay of, e.g., 500 ms [11,46] should be enforced.

Other issues that may need to be considered for building a system to execute LDQL queries over an open platform such as the WWW are coreferences and schema heterogeneity: Although URIs are used as globally unique identifiers for denoting entities in Linked Data, different publishers may use different URIs to denote the same entity. Some of the data about such a coreferenced entity will be ignored unless the coreference is detected and

[4] http://www.robotstxt.org/.

resolved. Similarly, Linked Data providers may choose different RDF vocabularies to represent their data, and these vocabularies may overlap w.r.t. the classes and properties that they define. Then, a query expressed in terms of a specific vocabulary must be rewritten to benefit from data represented using another vocabulary. Another issue is frequently changing data. In our data model and proposal we have implicitly assumed that the Web of Linked Data is static during query execution. However, in an scenario in which some data changes very frequently in the Web of Linked Data, one might need to drop this assumption. It would be interesting and challenging to deal with this dynamic scenario both from a theoretical and a practical point of view.

As a final, more practical topic that is worth investigating we refer to the possibility of integrating LDQL and LDQL query execution with other Web-based interfaces for accessing and querying Linked Data (such as SPARQL endpoints [47] and Triple Pattern Fragments [48]). Interesting problems in this context are related to discovering such interfaces and leveraging them to achieve an increase in performance.

Acknowledgments

Pérez is supported by the Millennium Nucleus Center for Semantic Web Research, Grant NC120004, and Fondecyt grant 1140790.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.websem.2016.10.001.

References

[1] T. Berners-Lee, Linked Data, 2006. At http://www.w3.org/DesignIssues/LinkedData.html.
[2] R. Fielding, J. Gettys, J.C. Mogul, H. Frystyk, L. Masinter, P.J. Leach, T. Berners-Lee, Hypertext Transfer Protocol – HTTP/1.1, June, 1999.
[3] T. Berners-Lee, R. Fielding, L. Masinter, Uniform Resource Identifier (URI): Generic Syntax, RFC 3986, Jan. 2005. Online at http://tools.ietf.org/html/rfc3986.
[4] R. Cyganiak, D. Wood, M. Lanthaler, RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation, 2014.
[5] O. Hartig, An overview on execution strategies for linked data queries, Datenbank-Spektrum 13 (2) (2013).
[6] S. Harris, A. Seaborne, E. Prud'hommeaux, SPARQL 1.1 Query Language, W3C Recommendation, Mar. 2013.
[7] P. Bouquet, C. Ghidini, L. Serafini, Querying the web of data: A formal approach, in: Proceedings of the 4th Asian Semantic Web Conference, ASWC, 2009.
[8] O. Hartig, SPARQL for a web of linked data: Semantics and computability, in: Proceedings of the 9th Extended Semantic Web Conference, ESWC, 2012.
[9] A. Harth, S. Speiser, On completeness classes for query evaluation on linked data, in: Proceedings of the 26th AAAI Conference, 2012.
[10] O. Hartig, G. Pirrò, A context-based semantics for SPARQL property paths over the web, in: Proceedings of the 12th Extended Semantic Web Conference, ESWC, 2015.
[11] J. Umbrich, A. Hogan, A. Polleres, S. Decker, Link traversal querying for a diverse web of data, Semant. Web J. 6 (6) (2015) 585–624.
[12] S. Schaffert, C. Bauer, T. Kurz, F. Dorschel, D. Glachs, M. Fernandez, The linked media framework: Integrating and interlinking enterprise media content and data, in: Proceedings of the 8th International Conference on Semantic Systems, I-Semantics, 2012.
[13] V. Fionda, G. Pirrò, C. Gutierrez, NautiLOD: A formal language for the web of data graph, ACM Trans. Web 9 (1) (2015) 5:1–5:43.
[14] O. Hartig, J. Pérez, LDQL: A query language for the web of linked data, in: Proceedings of the 14th International Semantic Web Conference, ISWC, 2015.
[15] P. Barceló Baeza, Querying graph databases, in: Proceedings of the 32nd Symposium on Principles of Database Systems, PODS, 2013.
[16] J. Clark, S. DeRose, XML Path Language (XPath), W3C Recommendation, Nov. 1999.
[17] M. Consens, A.O. Mendelzon, GraphLog: a visual formalism for real life recursion, in: Proceedings of the 9th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, PODS, 1990.
[18] J. Pérez, M. Arenas, C. Gutierrez, nSPARQL: A navigational language for RDF, J. Web Semant. 8 (4) (2010) 255–270.
[19] J. Webber, A Programmatic Introduction to Neo4j, in: SPLASH, 2012, pp. 217–218.
[20] N. Martínez-Bazan, S. Gómez-Villamor, F. Escale-Claveras, DEX: A high-performance graph database management system, in: ICDE Workshops, 2011.
[21] D. Florescu, A.Y. Levy, A.O. Mendelzon, Database techniques for the world-wide web: A survey, SIGMOD Rec. 27 (3) (1998) 59–74.
[22] D. Konopnicki, O. Shmueli, W3QS: A query system for the world-wide web, in: Proceedings of 21th International Conference on Very Large Data Bases, VLDB, 1995.
[23] D. Konopnicki, O. Shmueli, Information gathering in the world-wide web: The W3QL query language and the W3QS system, CM Trans. Database Syst. 23 (4) (1998) 369–410.
[24] G.O. Arocena, A.O. Mendelzon, G.A. Mihaila, Applications of a web query language, Comput. Netw. ISDN Syst. 29 (8-13) (1997) 1305–1316.
[25] A.O. Mendelzon, G.A. Mihaila, T. Milo, Querying the world wide web, Int. J. Digit. Libr. 1 (1) (1997) 54–67.
[26] R. Himmeröder, G. Lausen, B. Ludäscher, C. Schlepphorst, On a declarative semantics for web queries, in: Proceedings of the 5th International Conference on Deductive and Object-Oriented Databases, DOOD, 1997.
[27] A.O. Mendelzon, T. Milo, Formal models of web queries, Inf. Syst. 23 (8) (1998) 615–637.
[28] L.V.S. Lakshmanan, F. Sadri, I.N. Subramanian, A declarative language for querying and restructuring the Web, in: Proceedings of the 6th International Workshop on Research Issues in Data Engineering, RIDE, 1996.
[29] G.O. Arocena, A.O. Mendelzon, WebOQL: Restructuring documents, databases and webs, in: Proceedings of the 14th International Conference on Data Engineering, ICDE, 1998.
[30] T. Guan, M. Liu, L.V. Saxton, Structure-based queries over the world wide web, in: Proceedings of the 17th International Conference on Conceptual Modeling, ER, 1998.
[31] G. Mecca, A.O. Mendelzon, P. Merialdo, Efficient queries over web views, in: Proceedings of the 6th International Conference on Extending Database Technology, EDBT, 1998.
[32] E. Spertus, L.A. Stein, Squeal: A structured query language for the web, Comput. Netw. 33 (1–6) (2000) 95–103.
[33] M. Liu, T.W. Ling, A conceptual model and rule-based query language for HTML, World Wide Web 4 (1–2) (2001) 49–77.
[34] W.-S. Li, J. Shim, K.S. Candan, WebDB: A system for querying semi-structured data on the web, J. Vis. Lang. Comput. 13 (1) (2002) 3–33.
[35] O. Hartig, C. Bizer, J.-C. Freytag, Executing SPARQL queries over the web of linked data, in: Proceedings of the 8th International Semantic Web Conference, ISWC, 2009.
[36] G. Ladwig, D.T. Tran, Linked data query processing strategies, in: Proceedings of the 9th International Semantic Web Conference, ISWC, 2010.
[37] J. Umbrich, K. Hose, M. Karnstedt, A. Harth, A. Polleres, Comparing data summaries for processing live queries over linked data, World Wide Web 14 (5–6) (2011).
[38] S. Muñoz, J. Pérez, C. Gutierrez, Simple and efficient minimal RDFS, J. Web Semant. 7 (3) (2009) 220–234.
[39] M. Schneider, OWL 2 Web Ontology Language, RDF-Based Semantics, second ed., W3C Recommendation, Dec. 2012.
[40] J. Pérez, M. Arenas, C. Gutierrez, Semantics and complexity of SPARQL, ACM Trans. Database Syst. 34 (2009).
[41] M. Arenas, C. Gutierrez, J. Pérez, On the semantics of SPARQL, in: Semantic Web Information Management—A Model-Based Perspective, Springer, 2009, pp. 281–307. (Chapter 13).
[42] M. Arenas, S. Conca, J. Pérez, Counting beyond a yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard, in: Proceedings of the 21th World Wide Web Conference, WWW, 2012.
[43] O. Hartig, G. Pirrò, SPARQL with property paths on the Web, Semantic Web J. (2017) in press.
[44] M.Y. Vardi, The complexity of relational query languages (extended abstract), in: Proceedings of the 14th Annual ACM Symposium on Theory of Computing, 1982.
[45] C. Buil-Aranda, M. Arenas, O. Corcho, Semantics and optimization of the SPARQL 1.1 federation extension, in: Proceedings of the 8th Extended Semantic Web Conference, ESWC, 2011.
[46] A. Hogan, A. Harth, J. Umrich, S. Kinsella, A. Polleres, S. Decker, Searching and browsing linked data with SWSE: the semantic web search engine, J. Web Semant. 9 (4) (2012).
[47] L. Feigenbaum, G.T. Williams, K.G. Clark, E. Torres, SPARQL 1.1 protocol, W3C Recommendation, Mar. 2013.
[48] R. Verborgh, M. Vander Sande, O. Hartig, J. Van Herwegen, L. De Vocht, B. De Meester, G. Haesendonck, P. Colpaert, Triple pattern fragments: a low-cost knowledge graph interface for the web, J. Web Semant. 37–38 (2016) 184–206.